

User and Reference Manual



Copyright ©2016 Altova GmbH. All rights reserved. Use of this software is governed by an Altova license agreement. XMLSpy, MapForce, StyleVision, SchemaAgent, UModel, DatabaseSpy, DiffDog, Authentic, MissionKit, FlowForce, RaptorXML, MobileTogether, and Altova as well as their respective logos are either registered trademarks or trademarks of Altova GmbH. Protected by U.S. Patents 7,739,292, 7,200,816, and other pending patents. This software contains third party software or material that is protected by copyright and subject to other terms and conditions as detailed on the Altova website at http://www.altova.com/legal_3rdparty.html.

Altova UModel 2016 User & Reference Manual

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Published: 2016

© 2016 Altova GmbH

Table of Contents

1	UModel	3
2	Introducing UModel	6
3	UModel tutorial	8
3.1	Starting UModel	10
3.2	Use cases	14
3.3	Class Diagrams	22
3.3.1	Creating derived classes	29
3.4	Object Diagrams	34
3.5	Component Diagrams	39
3.6	Deployment Diagrams	44
3.7	Round-trip engineering (model - code - model)	48
3.8	Round-trip engineering (code - model - code)	54
4	UModel User Interface	64
4.1	Model Tree	65
4.2	Diagram Tree	70
4.3	Favorites	72
4.4	Properties	74
4.5	Styles	76
4.6	Hierarchy	79
4.7	Overview	82
4.8	Documentation	83
4.9	Layer	84
4.9.1	Layer Context menu	88
4.10	Messages	90
4.11	Diagram pane	91
4.11.1	Diagram properties	95
4.11.2	Cut, copy and paste in UModel Diagrams	97

4.12	Adding/Inserting model elements	100
4.13	Hyperlinking modeling elements	102
4.14	Bank samples	107

5 UModel Command Line Interface 110

5.1	File: New / Load / Save options	114

6 Projects and code engineering 116

6.1	Minimalist UModel project - starting from scratch	119
6.2	Importing source code into projects	124
6.3	Importing Java, C# and VB binaries	129
6.4	Synchronizing the Model and Source Code	135
6.4.1	Synchronization Tips	138
6.4.2	Refactoring Code and Synchronization	140
6.5	Forward engineering prerequisites	142
6.6	Java code to/from UModel elements	144
6.7	C# code to/from UModel elements	145
6.8	XML Schema to/from UModel elements	146
6.9	VB.NET code to/from UModel elements	147
6.10	Database to/from UModel elements	148
6.11	Including other UModel projects	149
6.12	Merging UModel projects	151
6.12.1	2-way Project merge	152
6.12.2	3-way Project merge	153
6.12.3	Manual 3-way project merge example	155
6.13	Sharing Packages and Diagrams	158
6.14	UML templates	161
6.14.1	Template signatures	163
6.14.2	Template binding	164
6.14.3	Template usage in operations and properties	165
6.15	Project Settings	166
6.16	Enhancing performance	167

7 Transforming UML models 170

7.1	Transformation example - Java to C#	171
7.2	C# to Java - model to model	177

7.3	Transformation settings	184
7.4	Automatically update transformation	185
8	Creating model relationships	188
8.1	Showing model relationships	191
8.2	Associations, realizations and dependencies	193
9	Generating UML documentation	198
9.1	Supplied SPS stylesheet	205
9.2	User-defined Stylesheets	206
10	UML Diagrams	208
10.1	Behavioral Diagrams	209
10.1.1	Activity Diagram	210
	<i>Inserting Activity Diagram elements</i>	211
	<i>Creating branches and merges</i>	213
	<i>Activity Diagram elements</i>	215
10.1.2	State Machine Diagram	225
	<i>Inserting state machine diagram elements</i>	226
	<i>Creating states, activities and transitions</i>	226
	<i>Composite states</i>	233
	<i>Code generation from State Machine diagrams</i>	235
	<i>Working with state machine code</i>	238
	<i>State Machine Diagram elements</i>	243
10.1.3	Protocol State Machine	246
	<i>Inserting Protocol State Machine elements</i>	246
	<i>Protocol State Machine Diagram elements</i>	248
10.1.4	Use Case Diagram	250
10.1.5	Communication Diagram	251
	<i>Inserting Communication Diagram elements</i>	251
10.1.6	Interaction Overview Diagram	254
	<i>Inserting Interaction Overview elements</i>	255
10.1.7	Sequence Diagram	259
	<i>Inserting sequence diagram elements</i>	260
	Lifeline	261
	Combined Fragment	262
	Interaction Use	265
	Gate	265
	State Invariant	266

	Messages	266
	<i>Generating Sequence Diagrams from source code</i>	271
	<i>Generate Sequence diagrams from properties</i>	275
	<i>Generate code from sequence diagram</i>	278
	Adding code to sequence diagrams.....	281
10.1.8	Timing Diagram	285
	<i>Inserting Timing Diagram elements</i>	286
	<i>Lifeline</i>	286
	<i>Tick Mark</i>	288
	<i>Event/Stimulus</i>	289
	<i>DurationConstraint</i>	289
	<i>TimeConstraint</i>	290
	<i>Message</i>	291
10.2	Structural Diagrams	293
10.2.1	Class Diagram	294
	<i>Customizing Class Diagrams</i>	294
	<i>Overriding Base Class Operations and Implementing</i> <i>Interface Operations</i>	301
	<i>Creating Getter and Setter Methods</i>	301
	<i>Ball and Socket Notation</i>	303
	<i>Adding Raised Exceptions to Methods of a Class</i>	304
	<i>Generating Class Diagrams</i>	305
10.2.2	Composite Structure Diagram	309
	<i>Inserting Composite Structure Diagram elements</i>	309
10.2.3	Component Diagram	311
10.2.4	Deployment Diagram	312
10.2.5	Object Diagram	313
10.2.6	Package Diagram	314
	<i>Inserting Package Diagram elements</i>	315
	<i>Generating Package Diagrams While Importing Code or</i> <i>Binaries</i>	317
10.2.7	Profile Diagram and stereotypes	320
	<i>Adding Stereotypes and defining tagged values</i>	322
	<i>Stereotypes and enumerations</i>	326
	<i>User-defined stereotype styles</i>	328
	<i>Custom stereotype icons - assigning</i>	330
10.3	Additional Diagrams	334
10.3.1	XML Schema Diagrams	335
	<i>Importing XML Schema(s)</i>	335
	<i>Inserting XML Schema elements</i>	341
	<i>Creating and generating an XML Schema</i>	345

10.3.2	Business Process Modeling Notation 1.0 / 2.0	347
	<i>Flow objects</i>	348
	Expanded Sub Processes	353
	Collapsed Sub Processes	354
	<i>Connecting objects</i>	356
	<i>Pools / Swimlanes</i>	358
	<i>Artifacts</i>	358
	<i>Choreography diagram</i>	359
	Choreography Tasks	360
	Tasks and Subprocesses	363
	Data Objects	364
	<i>Collaboration diagram</i>	364
	Conversations.....	365
	Tasks and Subprocesses	366
	Data Objects	367
	<i>Standard Business Process diagram BPMN 2.0</i>	367
	Tasks and Subprocesses	369
	Data Objects	370
10.3.3	SysML Diagrams	371
	<i>Structure diagrams</i>	371
	Block Definition diagram.....	372
	Internal Block diagram.....	375
	Parametric diagram.....	378
	Package diagram.....	380
	<i>Requirement diagram</i>	382
	<i>Behavior diagrams</i>	383
	Activity diagram.....	383
	Sequence diagram.....	385
	State Machine diagram.....	386
	Use Case diagram.....	387
11	UModel and Databases	390
11.1	Connecting to a Database	392
11.1.1	Starting the Database Connection Wizard	394
11.1.2	Database Drivers Overview	395
11.1.3	Setting up an ADO Connection	398
	<i>Connecting to an Existing Microsoft Access Database</i>	400
	<i>Setting up the SQL Server Data Link Properties</i>	400
	<i>Setting up the Microsoft Access Data Link Properties</i>	401
11.1.4	Setting up an ODBC Connection	404
	<i>Viewing the Available ODBC Drivers</i>	406
11.1.5	Setting up a JDBC Connection	407
	<i>Configuring the CLASSPATH</i>	408
11.1.6	Setting up a SQLite Connection	410

	<i>Connecting to an Existing SQLite Database.....</i>	410
11.1.7	Examples	411
	<i>Connecting to Firebird (ODBC).....</i>	411
	<i>Connecting to Firebird (JDBC).....</i>	414
	<i>Connecting to IBM DB2 (ODBC).....</i>	415
	<i>Connecting to IBM DB2 for i (ODBC).....</i>	421
	<i>Connecting to IBM Informix (JDBC).....</i>	424
	<i>Connecting to Microsoft Access (ADO).....</i>	425
	<i>Connecting to Microsoft SQL Server (ADO).....</i>	428
	<i>Connecting to Microsoft SQL Server (ODBC).....</i>	431
	<i>Connecting to MySQL (ODBC).....</i>	433
	<i>Connecting to Oracle (ODBC).....</i>	436
	<i>Connecting to PostgreSQL (ODBC).....</i>	441
	<i>Connecting to Sybase (JDBC).....</i>	443
11.2	Importing SQL databases into UModel.....	446
11.3	Viewing database modeling elements	450
11.4	Adding columns to a table	454
11.5	Adding a new table to model / database	457

12 XMI - XML Metadata Interchange 464

13 Teamwork support for UModel projects 468

13.1	Creating and editing subproject files	469

14 UModel Plug-in for Visual Studio 476

14.1	Installing the UModel Plug-in for Visual Studio	478
14.2	Adding UModel Support to Visual Studio Projects	480
14.3	Loading/Unloading UModel Projects	484
14.4	Synchronizing the Model and Code	485

15 UModel Plug-in for Eclipse 488

15.1	Installing the UModel Plug-in for Eclipse	491
15.2	The UModel Perspective	496
15.3	Adding UModel Support to Eclipse Projects	498
15.4	Importing Existing UModel Projects	500
15.5	Loading/Unloading UModel Projects	502

15.6	How Automatic Synchronization Works	503
15.7	Example: Setting up Automatic Synchronization	504

16 Source Control 518

16.1	Setting Up Source Control	520
16.2	Supported Source Control Systems	521
16.3	Source Control Commands	523
16.3.1	Open from Source Control	524
16.3.2	Enable Source Control	527
16.3.3	Get Latest Version	528
16.3.4	Get	529
16.3.5	Get Folder(s)	530
16.3.6	Check Out	531
16.3.7	Check In	533
16.3.8	Undo Check Out...	534
16.3.9	Add to Source Control	535
16.3.10	Remove from Source Control	538
16.3.11	Share from Source Control	539
16.3.12	Show History	540
16.3.13	Show Differences	542
16.3.14	Show Properties	544
16.3.15	Refresh Status	545
16.3.16	Source Control Manager	546
16.3.17	Change Source Control	547
16.4	Source Control with Git	548
16.4.1	Enabling Git Source Control with GIT SCC Plug-in	549
16.4.2	Adding a Project to Git Source Control	550
16.4.3	Cloning a Project from Git Source Control	552

17 UModel Diagram icons 556

17.1	Activity Diagram	557
17.2	Class Diagram	558
17.3	Communication diagram	559
17.4	Composite Structure Diagram	560
17.5	Component Diagram	561
17.6	Deployment Diagram	562
17.7	Interaction Overview diagram	563
17.8	Object Diagram	564

17.9	Package diagram	565
17.10	Profile Diagram	566
17.11	Protocol State Machine	567
17.12	Sequence Diagram	568
17.13	State Machine Diagram	569
17.14	Timing Diagram	570
17.15	Use Case diagram	571
17.16	XML Schema diagram	572
17.17	Business Process Modeling Notation	573
17.18	Business Process Modeling Notation 2.0	574
17.19	Database Modeling	575

18 Menu Reference 578

18.1	File	579
18.2	Edit	584
18.3	Project	587
18.4	Layout	594
18.5	View	595
18.6	Tools	596
18.6.1	Spelling...	597
18.6.2	Spelling options...	601
18.6.3	Scripting Editor	603
18.6.4	Macros	604
18.6.5	User-defined Tools	605
18.6.6	Customize...	606
	<i>Commands</i>	606
	<i>Toolbars</i>	606
	<i>Tools</i>	607
	<i>Keyboard</i>	610
	<i>Menu</i>	610
	<i>Macros</i>	611
	<i>Plug-Ins</i>	611
	<i>Options</i>	611
18.6.7	Restore Toolbars and Windows	612
18.6.8	Options	613
18.7	Window	619
18.8	Help	620

19 Code Generator 622

19.1	The way to SPL (Spy Programming Language)	623
19.1.1	Basic SPL structure	624
19.1.2	Variables	625
19.1.3	Operators	633
19.1.4	Conditions	634
19.1.5	Collections and foreach	635
19.1.6	Subroutines	637
	<i>Subroutine declaration</i>	637
	<i>Subroutine invocation</i>	638
19.2	Error Codes	639

20 UModel Programmer's Reference 642

20.1	Release Notes	644
20.2	Scripting Editor	648
20.2.1	Overview	650
	<i>Scripting Projects in UModel</i>	651
	<i>The Scripting Editor GUI</i>	652
	<i>Components of a Scripting Project</i>	657
20.2.2	Creating a Scripting Project	659
20.2.3	Global Declarations	661
20.2.4	Forms	662
	<i>Creating a New Form</i>	662
	<i>Form Design and Form Objects</i>	663
	<i>Form Events</i>	665
20.2.5	Events	668
20.2.6	Macros	669
	<i>Creating and Editing a Macro</i>	669
	<i>Running a Macro</i>	673
	<i>Debugging a Macro</i>	674
20.2.7	Programming Points	676
	<i>Built-in Commands</i>	677
	Form usage and commands	684
20.2.8	Usage example (UModel Find sample)	685
20.3	IDE PlugIns	689
20.3.1	How to create a plug-in in MS Visual Studio .NET	690
	<i>Add reference to plugin library</i>	690
	<i>Add reference to UModel type library</i>	691

	<i>Change compiler settings - make COM-visible.....</i>	692
	<i>Change compiler settings - expose COM wrapper.....</i>	693
	<i>How to sign your plug-in with a strong name (optional).....</i>	693
	<i>Implement IUModelPlugIn.....</i>	694
	<i>Test UModel plug-in.....</i>	696
20.3.2	Deployment of PlugIns	697
20.3.3	Configuration XML	699
20.3.4	ActiveX Controls	702
20.3.5	IUModelPlugIn	704
	<i>OnInitialize.....</i>	706
	<i>OnRunning.....</i>	706
	<i>OnShutdown.....</i>	706
	<i>GetUIModifications.....</i>	707
	<i>GetDescription.....</i>	708
	<i>OnCommand.....</i>	708
	<i>OnUpdateCommand.....</i>	708
20.4	The UModel API	710
20.4.1	Overview	711
	<i>Object model.....</i>	711
	Object model UMLData.....	712
	Object model UMLData styles.....	714
	Graphical Objects.....	715
	<i>How to.....</i>	716
	How to create Sequence Diagrams.....	716
	How to generate Sequence Diagrams from code.....	716
	How to create Sequence diagrams manually.....	717
	Undo / Redo and UMLData transaction handling.....	721
	How to use predefined UModel elements.....	722
	How to work with Stereotypes and Tagged Values.....	722
	How to use UMLData events and eventfilters.....	723
	How to create and use Hyperlinks.....	725
	<i>Error handling.....</i>	726
	<i>Programming languages.....</i>	728
	C#	729
	Java	734
	Example Java Project.....	735
	JScript	739
	Start application.....	740
	Document Access.....	741
	Generate documentation.....	742
	Generate code.....	743
	Update Documentation.....	748
20.4.2	Usage Examples	757
	<i>Styles (C#).....</i>	757
	SetupStyles (Deployment).....	763
	<i>CSharpDelegate (C#).....</i>	763

	<i>DefaultPrefix (C#)</i>	768
	<i>CreateCode (JScript)</i>	776
	<i>UpdateDocumentation (JScript)</i>	781
	<i>StatisticsActiveX (C#)</i>	790
20.5	UModel API Reference	797
20.5.1	UModel Plugins	798
	<i>UModelAPI - IUModelPlugIn</i>	798
	<i>UModelAPI - UModelUpdateAction</i>	799
20.5.2	UModel API	800
	<i>UModelAPI - IApplication</i>	801
	<i>UModelAPI - IBinaryTypeEntries</i>	804
	<i>UModelAPI - IBinaryTypeEntry</i>	805
	<i>UModelAPI - ICollectionTemplate</i>	806
	<i>UModelAPI - ICollectionTemplates</i>	807
	<i>UModelAPI - IDiagramWindow</i>	808
	<i>UModelAPI - IDiagramWindows</i>	811
	<i>UModelAPI - IDialog</i>	812
	<i>UModelAPI - IDialogs</i>	813
	<i>UModelAPI - IDocument</i>	816
	<i>UModelAPI - IExportXMIFileDlg</i>	824
	<i>UModelAPI - IFocusedUMLDataNotifier</i>	825
	<i>UModelAPI - IGenerateDocumentationDlg</i>	826
	<i>UModelAPI - IGenerateSequenceDiagramDlg</i>	833
	<i>UModelAPI - IGenerateStateMachineCodeDlg</i>	835
	<i>UModelAPI - IImportBinaryTypesDlg</i>	837
	<i>UModelAPI - IImportDatabaseDlg</i>	841
	<i>UModelAPI - IImportSourceDirectoryDlg</i>	843
	<i>UModelAPI - IImportSourceDlg</i>	844
	<i>UModelAPI - IImportSourceProjectDlg</i>	847
	<i>UModelAPI - IImportXMLSchemaDirectoryDlg</i>	848
	<i>UModelAPI - IImportXMLSchemaFileDlg</i>	849
	<i>UModelAPI - IIncludeSubprojectDlg</i>	850
	<i>UModelAPI - IKindSelection</i>	851
	<i>UModelAPI - IKindSelectionList</i>	851
	<i>UModelAPI - ILocalOptions</i>	852
	<i>UModelAPI - ILocalOptionsCodeEngineering</i>	853
	<i>UModelAPI - ILocalOptionsDiagramEditing</i>	856
	<i>UModelAPI - ILocalOptionsEditing</i>	859
	<i>UModelAPI - ILocalOptionsFile</i>	860
	<i>UModelAPI - ILocalOptionsView</i>	862
	<i>UModelAPI - IMatchRenamedDlg</i>	864

<i>UModelAPI - IMatchRenamedEntries</i>	865
<i>UModelAPI - IMatchRenamedEntry</i>	866
<i>UModelAPI - IModelTransformationDlg</i>	867
<i>UModelAPI - IModelTransformationTypeMapping</i>	869
<i>UModelAPI - IModelTransformationTypeMappings</i>	870
<i>UModelAPI - IProjectSettingsDlg</i>	871
<i>UModelAPI - ISaveAllDiagramsAsImagesDlg</i>	872
<i>UModelAPI - ISynchronizationSettingsDlg</i>	873
<i>UModelAPI - ITransactionNotifier</i>	874
<i>UModelAPI - IURLDlg</i>	875
<i>Events</i>	876
UModelAPI - _IApplicationEvents.....	876
UModelAPI - _IDiagramWindowEvents.....	877
UModelAPI - _IDocumentEvents.....	877
UModelAPI - _IFocusedUMLDataEvents.....	879
UModelAPI - _ISynchronizationEvents.....	879
UModelAPI - _ITransactionEvents.....	879
<i>Enumerations</i>	880
UModelAPI - ENUMApplicationFrameTitle.....	880
UModelAPI - ENUMApplicationStatus.....	880
UModelAPI - ENUMAutolayoutGrowDirectionKind.....	881
UModelAPI - ENUMCodeLang.....	881
UModelAPI - ENUMCodeLangVersion.....	882
UModelAPI - ENUMDiagramLayoutKind.....	883
UModelAPI - ENUMDocumentationFilePathKind.....	883
UModelAPI - ENUMDocumentationFontSetting.....	884
UModelAPI - ENUMDocumentationImageFormat.....	884
UModelAPI - ENUMDocumentationOutputFormat.....	885
UModelAPI - ENUMExportXMIType.....	885
UModelAPI - ENUMOpenMessageWindow.....	885
UModelAPI - ENUMOutputImageFormat.....	886
UModelAPI - ENUMSynchronizationDeleteKind.....	886
UModelAPI - ENUMSynchronizationKind.....	886
UModelAPI - ENUMSyntaxCheckKind.....	887
<i>UMLData</i>	887
UModelAPI - IUMLData.....	888
UModelAPI - IUMLDataList.....	890
IUMLElement.....	895
UModelAPI - IUMLAbstraction.....	895
UModelAPI - IUMLAcceptEventAction.....	896
UModelAPI - IUMLAction.....	897
UModelAPI - IUMLActionExecutionSpecification.....	899
UModelAPI - IUMLActionInputPin.....	900
UModelAPI - IUMLActivity.....	900
UModelAPI - IUMLActivityEdge.....	902
UModelAPI - IUMLActivityFinalNode.....	904
UModelAPI - IUMLActivityGroup.....	905
UModelAPI - IUMLActivityNode.....	906
UModelAPI - IUMLActivityParameterNode.....	907
UModelAPI - IUMLActivityPartition.....	908

UModelAPI - IUMLActor.....	910
UModelAPI - IUMLAnyReceiveEvent.....	910
UModelAPI - IUMLArtifact.....	911
UModelAPI - IUMLAssociation.....	913
UModelAPI - IUMLAssociationClass.....	914
UModelAPI - IUMLBehavior.....	915
UModelAPI - IUMLBehavioralFeature.....	917
UModelAPI - IUMLBehavioredClassifier.....	918
UModelAPI - IUMLBehaviorExecutionSpecification.....	920
UModelAPI - IUMLCallAction.....	920
UModelAPI - IUMLCallBehaviorAction.....	922
UModelAPI - IUMLCallEvent.....	922
UModelAPI - IUMLCallOperationAction.....	923
UModelAPI - IUMLCentralBufferNode.....	924
UModelAPI - IUMLChangeEvent.....	925
UModelAPI - IUMLClass.....	926
UModelAPI - IUMLClassifier.....	929
UModelAPI - IUMLClassifierTemplateParameter.....	932
UModelAPI - IUMLCollaboration.....	933
UModelAPI - IUMLCollaborationUse.....	934
UModelAPI - IUMLCombinedFragment.....	935
UModelAPI - IUMLComment.....	936
UModelAPI - IUMLCommentTextHyperlink.....	937
UModelAPI - IUMLComponent.....	938
UModelAPI - IUMLComponentRealization.....	940
UModelAPI - IUMLConnectableElement.....	941
UModelAPI - IUMLConnectionPointReference.....	942
UModelAPI - IUMLConnector.....	943
UModelAPI - IUMLConnectorEnd.....	944
UModelAPI - IUMLConstraint.....	945
UModelAPI - IUMLControlFlow.....	947
UModelAPI - IUMLControlNode.....	948
UModelAPI - IUMLDataStoreNode.....	948
UModelAPI - IUMLDataType.....	949
UModelAPI - IUMLDecisionNode.....	950
UModelAPI - IUMLDependency.....	951
UModelAPI - IUMLDeployedArtifact.....	951
UModelAPI - IUMLDeployment.....	952
UModelAPI - IUMLDeploymentTarget.....	953
UModelAPI - IUMLDevice.....	954
UModelAPI - IUMLDirectedRelationship.....	954
UModelAPI - IUMLDuration.....	955
UModelAPI - IUMLDurationConstraint.....	956
UModelAPI - IUMLDurationInterval.....	957
UModelAPI - IUMLDurationObservation.....	958
UModelAPI - IUMLElement.....	958
UModelAPI - IUMLElementImport.....	962
UModelAPI - IUMLElementEncapsulatedClassifier.....	963
UModelAPI - IUMLEnumeration.....	964
UModelAPI - IUMLEnumerationLiteral.....	965
UModelAPI - IUMLEvent.....	966
UModelAPI - IUMLExceptionHandler.....	967
UModelAPI - IUMLExecutableNode.....	968
UModelAPI - IUMLExecutionEnvironment.....	969

UModelAPI - IUMLExecutionSpecification.....	970
UModelAPI - IUMLExpansionNode.....	971
UModelAPI - IUMLExpansionRegion.....	972
UModelAPI - IUMLExpression.....	973
UModelAPI - IUMLExtend.....	974
UModelAPI - IUMLExtensionPoint.....	976
UModelAPI - IUMLFeature.....	976
UModelAPI - IUMLFinalNode.....	977
UModelAPI - IUMLFinalState.....	978
UModelAPI - IUMLFlowFinalNode.....	979
UModelAPI - IUMLForkNode.....	979
UModelAPI - IUMLFunctionBehavior.....	980
UModelAPI - IUMLGate.....	981
UModelAPI - IUMLGeneralization.....	981
UModelAPI - IUMLHyperlink.....	982
UModelAPI - IUMLHyperlink2File.....	983
UModelAPI - IUMLHyperlink2GuiElement.....	984
UModelAPI - IUMLHyperlink2Model.....	985
UModelAPI - IUMLInclude.....	986
UModelAPI - IUMLInformationFlow.....	987
UModelAPI - IUMLInitialNode.....	989
UModelAPI - IUMLInputPin.....	990
UModelAPI - IUMLInstanceSpecification.....	991
UModelAPI - IUMLInstanceValue.....	993
UModelAPI - IUMLInteraction.....	994
UModelAPI - IUMLInteractionConstraint.....	995
UModelAPI - IUMLInteractionFragment.....	997
UModelAPI - IUMLInteractionOperand.....	997
UModelAPI - IUMLInteractionUse.....	998
UModelAPI - IUMLInterface.....	999
UModelAPI - IUMLInterfaceRealization.....	1002
UModelAPI - IUMLInterruptibleActivityRegion.....	1003
UModelAPI - IUMLInterval.....	1004
UModelAPI - IUMLIntervalConstraint.....	1005
UModelAPI - IUMLInvocationAction.....	1006
UModelAPI - IUMLJoinNode.....	1007
UModelAPI - IUMLLifeline.....	1008
UModelAPI - IUMLLiteralBoolean.....	1009
UModelAPI - IUMLLiteralInteger.....	1009
UModelAPI - IUMLLiteralNull.....	1010
UModelAPI - IUMLLiteralSpecification.....	1011
UModelAPI - IUMLLiteralString.....	1011
UModelAPI - IUMLLiteralUnlimitedNatural.....	1012
UModelAPI - IUMLManifestation.....	1012
UModelAPI - IUMLMergeNode.....	1013
UModelAPI - IUMLMessage.....	1014
UModelAPI - IUMLMessageEnd.....	1016
UModelAPI - IUMLMessageEvent.....	1017
UModelAPI - IUMLMessageOccurrenceSpecification.....	1017
UModelAPI - IUMLModel.....	1018
UModelAPI - IUMLMultiplicityElement.....	1018
UModelAPI - IUMLNamedElement.....	1020
UModelAPI - IUMLNamespace.....	1022
UModelAPI - IUMLNode.....	1024

UModelAPI - IUMLObjectFlow.....	1025
UModelAPI - IUMLObjectNode.....	1026
UModelAPI - IUMLObservation.....	1028
UModelAPI - IUMLOccurrenceSpecification.....	1029
UModelAPI - IUMLOpaqueAction.....	1030
UModelAPI - IUMLOpaqueBehavior.....	1031
UModelAPI - IUMLOpaqueExpression.....	1031
UModelAPI - IUMLOperation.....	1032
UModelAPI - IUMLOutputPin.....	1034
UModelAPI - IUMLPackage.....	1035
UModelAPI - IUMLPackageableElement.....	1037
UModelAPI - IUMLPackageImport.....	1038
UModelAPI - IUMLPackageMerge.....	1039
UModelAPI - IUMLParameter.....	1040
UModelAPI - IUMLParameterableElement.....	1041
UModelAPI - IUMLPin.....	1042
UModelAPI - IUMLPort.....	1043
UModelAPI - IUMLPrimitiveType.....	1044
UModelAPI - IUMLProfile.....	1045
UModelAPI - IUMLProfileApplication.....	1046
UModelAPI - IUMLProperty.....	1047
UModelAPI - IUMLProtocolStateMachine.....	1050
UModelAPI - IUMLProtocolTransition.....	1051
UModelAPI - IUMLPseudostate.....	1052
UModelAPI - IUMLRealization.....	1053
UModelAPI - IUMLRedefinableElement.....	1054
UModelAPI - IUMLRedefinableTemplateSignature.....	1054
UModelAPI - IUMLRegion.....	1055
UModelAPI - IUMLRelationship.....	1056
UModelAPI - IUMLSendSignalAction.....	1057
UModelAPI - IUMLSignal.....	1058
UModelAPI - IUMLSignalEvent.....	1059
UModelAPI - IUMLSlot.....	1059
UModelAPI - IUMLState.....	1061
UModelAPI - IUMLStateInvariant.....	1064
UModelAPI - IUMLStateMachine.....	1065
UModelAPI - IUMLStereotype.....	1066
UModelAPI - IUMLStereotypeApplication.....	1067
UModelAPI - IUMLStructuralFeature.....	1068
UModelAPI - IUMLStructuredActivityNode.....	1069
UModelAPI - IUMLStructuredClassifier.....	1071
UModelAPI - IUMLTemplateableElement.....	1072
UModelAPI - IUMLTemplateBinding.....	1073
UModelAPI - IUMLTemplateParameter.....	1074
UModelAPI - IUMLTemplateParameterSubstitution.....	1076
UModelAPI - IUMLTemplateSignature.....	1077
UModelAPI - IUMLTimeConstraint.....	1078
UModelAPI - IUMLTimeEvent.....	1078
UModelAPI - IUMLTimeExpression.....	1079
UModelAPI - IUMLTimeInterval.....	1081
UModelAPI - IUMLTimeObservation.....	1081
UModelAPI - IUMLTransition.....	1082
UModelAPI - IUMLTrigger.....	1084
UModelAPI - IUMLType.....	1085

UModelAPI - IUMLTypedElement.....	1086
UModelAPI - IUMLUsage.....	1086
UModelAPI - IUMLUseCase.....	1087
UModelAPI - IUMLValuePin.....	1089
UModelAPI - IUMLValueSpecification.....	1091
UModelAPI - IUMLValueSpecificationAction.....	1093
UModelAPI - IUMLVertex.....	1094
IUMLGuiElement.....	1095
UModelAPI - IUMLGuiActivityDiagram.....	1096
UModelAPI - IUMLGuiAttachedNode.....	1096
UModelAPI - IUMLGuiBehaviorDiagram.....	1097
UModelAPI - IUMLGuiBPMN2ChoreographyDiagram.....	1098
UModelAPI - IUMLGuiBPMN2CollaborationDiagram.....	1098
UModelAPI - IUMLGuiBPMN2Diagram.....	1099
UModelAPI - IUMLGuiBPMNDiagram.....	1099
UModelAPI - IUMLGuiClassDiagram.....	1100
UModelAPI - IUMLGuiCommunicationDiagram.....	1100
UModelAPI - IUMLGuiCommunicationLink.....	1101
UModelAPI - IUMLGuiComponentDiagram.....	1102
UModelAPI - IUMLGuiCompositeStructureDiagram.....	1102
UModelAPI - IUMLGuiConstrainedNode.....	1103
UModelAPI - IUMLGuiContainmentLink.....	1104
UModelAPI - IUMLGuiDeploymentDiagram.....	1105
UModelAPI - IUMLGuiDiagram.....	1105
UModelAPI - IUMLGuiDiagramLayer.....	1109
UModelAPI - IUMLGuiElement.....	1110
UModelAPI - IUMLGuiEndWaypoint.....	1111
UModelAPI - IUMLGuiExtensionDiagram.....	1111
UModelAPI - IUMLGuiInteractionDiagram.....	1112
UModelAPI - IUMLGuiInteractionOverviewDiagram.....	1112
UModelAPI - IUMLGuiLabeledRelativeNodeLink.....	1113
UModelAPI - IUMLGuiLineConnectionWaypoint.....	1114
UModelAPI - IUMLGuiLineLink.....	1115
UModelAPI - IUMLGuiLink.....	1117
UModelAPI - IUMLGuiMiddleWaypoint.....	1118
UModelAPI - IUMLGuiNodeLink.....	1119
UModelAPI - IUMLGuiNote.....	1121
UModelAPI - IUMLGuiNoteLink.....	1122
UModelAPI - IUMLGuiObjectDiagram.....	1123
UModelAPI - IUMLGuiPackageDiagram.....	1123
UModelAPI - IUMLGuiProfileDiagram.....	1124
UModelAPI - IUMLGuiProtocolStateMachineDiagram.....	1125
UModelAPI - IUMLGuiRelativeNodeLink.....	1125
UModelAPI - IUMLGuiRootElement.....	1126
UModelAPI - IUMLGuiSeparatedNodeLink.....	1127
UModelAPI - IUMLGuiSeparatedNodeLink2D.....	1129
UModelAPI - IUMLGuiSequenceDiagram.....	1130
UModelAPI - IUMLGuiStateMachineDiagram.....	1131
UModelAPI - IUMLGuiStructureDiagram.....	1131
UModelAPI - IUMLGuiStyle.....	1132
UModelAPI - IUMLGuiStyles.....	1133
UModelAPI - IUMLGuiSubDiagramNode.....	1134
UModelAPI - IUMLGuiSysMLActivityDiagram.....	1135
UModelAPI - IUMLGuiSysMLBlockDefinitionDiagram.....	1136

UModelAPI - IUMLGuiSysMLInternalBlockDiagram.....	1137
UModelAPI - IUMLGuiSysMLPackageDiagram.....	1137
UModelAPI - IUMLGuiSysMLParametricDiagram.....	1138
UModelAPI - IUMLGuiSysMLRequirementDiagram.....	1139
UModelAPI - IUMLGuiSysMLSequenceDiagram.....	1139
UModelAPI - IUMLGuiSysMLStateMachineDiagram.....	1140
UModelAPI - IUMLGuiSysMLUseCaseDiagram.....	1140
UModelAPI - IUMLGuiTextHyperlink.....	1141
UModelAPI - IUMLGuiTextLabel.....	1143
UModelAPI - IUMLGuiTextLabelWaypoint.....	1144
UModelAPI - IUMLGuiTickMark.....	1145
UModelAPI - IUMLGuiTimingDiagram.....	1146
UModelAPI - IUMLGuiTimingDiagramLifeline.....	1148
UModelAPI - IUMLGuiTimingDiagramMessage.....	1150
UModelAPI - IUMLGuiUseCaseDiagram.....	1151
UModelAPI - IUMLGuiVisibleElement.....	1151
UModelAPI - IUMLGuiWaypoint.....	1152
UModelAPI - IUMLGuiXMLSchemaDiagram.....	1154
Events.....	1154
UModelAPI - _IUMLDataEvents.....	1154
Enumerations.....	1155
UModelAPI - ENUMUMLAggregationKind.....	1155
UModelAPI - ENUMUMLCallConcurrencyKind.....	1156
UModelAPI - ENUMUMLConnectorKind.....	1156
UModelAPI - ENUMUMLDataEventFilter.....	1157
UModelAPI - ENUMUMLDBDataSourceMethod.....	1157
UModelAPI - ENUMUMLExpansionKind.....	1158
UModelAPI - ENUMUMLGuiStyleKind.....	1158
UModelAPI - ENUMUMLGuiTextLabelKind.....	1159
UModelAPI - ENUMUMLInteractionOperatorKind.....	1160
UModelAPI - ENUMUMLMessageKind.....	1160
UModelAPI - ENUMUMLMessageSort.....	1161
UModelAPI - ENUMUMLObjectNodeOrderingKind.....	1161
UModelAPI - ENUMUMLParameterDirectionKind.....	1161
UModelAPI - ENUMUMLPredefinedElement.....	1162
UModelAPI - ENUMUMLPseudostateKind.....	1163
UModelAPI - ENUMUMLTransitionKind.....	1163
UModelAPI - ENUMUMLVisibilityKind.....	1164
UModelAPI - IUMLDataAll.....	1164
UModelAPI - UMLData.....	1232

21 License Information 1236

21.1 Electronic Software Distribution.....	1237
21.2 Software Activation and License Metering.....	1238
21.3 Intellectual Property Rights.....	1239
21.4 Altova End User License Agreement.....	1240

Index

Chapter 1

UModel

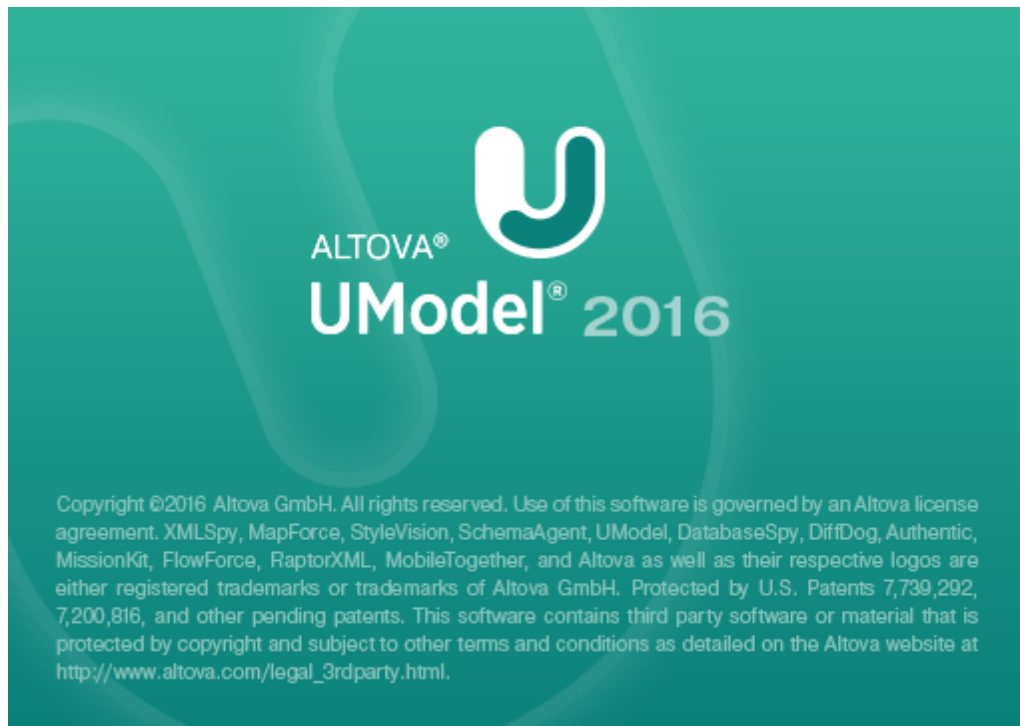
1 UModel

UModel® 2016 Enterprise Edition is an affordable UML modeling application with a rich visual interface and superior usability features to help level the UML learning curve, and includes many high-end functions to empower users with the most practical aspects of the UML 2.4 specification. UModel is a 32/64-bit Windows application that runs on Windows Server 2003/2008/2012, Windows XP, Windows Vista, Windows 7, Windows 8, and Windows 10. 64-bit support is available for the Enterprise and Professional editions.

UModel® 2016 supports:

- All 14 [UML 2.4 modeling diagrams](#)
- Support for the Model Driven Architecture (MDA) which allows conversion between different programming languages (Enterprise edition only)
- Ability to import and export [SQL databases](#) into UModel (Enterprise/Professional editions only)
- [Teamwork](#) support - allowing concurrent project editing
- [3-way](#) project merging
- [Protocol State](#) Machines
- [SysML](#) 1.2 diagrams (Enterprise/Professional Edition only)
- [Sequence Diagram](#) generation directly from source code
- Code generation from [State Machine diagrams](#)
- UModel API and plug-in (Enterprise/Professional Edition only)
- Built-in [Scripting](#) environment and Form editor (Enterprise/Professional Edition only)
- Visual Studio integration (Enterprise/Professional Edition only)
- Eclipse integration (Enterprise/Professional Edition only)
- Support for [version control](#) systems
- [XML Schema](#) diagrams
- Business Process Modeling Notation 1.0 and 2.0 (Enterprise/Professional Edition only)
- Multiple layers per UML diagram (Enterprise/Professional Edition only)
- Import of [Java, C# and Visual Basic binaries](#)
- [Hyperlinking](#) of diagrams and modeling elements
- Syntax coloring in diagrams
- Cascading styles
- Unlimited Undo and Redo
- Java, C# and Visual Basic [code generation](#) from models
- [Reverse engineering](#) of existing Java, C#, and Visual Basic source code
- Complete round-trip processing allowing code and model merging
- [XML version 2.4](#) for UML 2.0, 2.1, & 2.1.2, 2.2, 2.3, 2.4 - model import and export
- Generation of UModel project [documentation](#)

These capabilities allow developers, including those new to software modeling, to quickly leverage UML to enhance productivity and maximize their results.



UML®, OMG™, Object Management Group™, and Unified Modeling Language™ are either registered trademarks or trademarks of Object Management Group, Inc. in the United States and/or other countries.

Last updated: 03 February 2016

Chapter 2

Introducing UModel

2 Introducing UModel

Altova web site:  [Introduction to Altova UModel](#)

The UML is a complete modeling language but does not discuss, or prescribe, the methodology for the development, code generation and round-trip engineering processes. UModel has therefore been designed to allow complete flexibility during the modeling process:

- UModel diagrams can be created in any order, and at any time; there is no need to follow a prescribed sequence during modeling.
- Code, or model merging can be achieved at the project, package, or even class level. UModel does not require that pseudo-code, or comments in the generated code be present, in order to accomplish round-trip engineering.
- Code generation in UModel is based on Spy Programming Language (SPL) templates and is completely customizable. Customizations are automatically recognized during code generation.
- Code generation and reverse-engineering currently supports the following languages:

Language	Version
C#	1.2, 2.0, 3.0, 4.0, 5.0, 6.0
Java	1.4, 5.0 (1.5), 6, 7, 8
Visual Basic .NET	7.1, 8.0, 9.0

- A single project can support Java, C#, or VB.NET code simultaneously.
- Support for UML templates and generics.
- XML Metadata Interchange (XMI 2.1 for UML 2.0, 2.1.2, 2.2, 2.3, and XMI 2.4 for UML 2.4.1)
- When adding properties or operations, UModel provides in-place entry helpers to choose types, protection levels, and other properties that are also available in industrial-strength IDEs such as XMLSpy, Visual Studio or Eclipse.
- Syntax-coloring in diagrams makes UML diagrams more attractive and intuitive.
- Modeling elements and their properties (font, colors, borders etc.) are completely customizable in an hierarchical fashion at the project, node/line, element family and element level.
- Customizable actors can be defined in use-case diagrams to depict terminals, or any other symbols.
- Modeling elements can be searched for by name in the Diagram tab, Model Tree pane, Messages and Documentation windows.
- Class, or object associations, dependencies, generalizations etc. can be found/ highlighted in model diagrams through the context menu.
- The unlimited levels of Undo/Redo track not only content changes, but also all style changes made to any model element.

Please note:

This document does not attempt to describe, or explain, the Unified Modeling Language (UML); it describes how to use the UModel modeling application, to model code and achieve round-trip engineering results.

Chapter 3

UModel tutorial

3 UModel tutorial

This tutorial describes, and follows, the general sequence used when creating a modeling project in UModel.

The major portion of the tutorial deals with the forward-engineering process, i.e. using UModel to create UML diagrams and generate code as the precursor to the round-trip engineering sections that follow. The round-trip engineering sections describe the process from both code and model vantage points.

The tutorial describes the following UML diagrams, and how to manipulate the various modeling elements within them. The following diagrams and follow-on tasks are discussed:

Forward engineering process:

- Use cases
- Class diagrams
- Object diagrams
- Component diagrams
- Deployment diagrams

Round-trip process (model - code - model)

- Code generation from UModel
- Add a new operation to the external code
- Merge the external code back into UModel.

Round-trip process (code - model - code)

- Import code produced by XMLSpy from a directory (or from a project file)
- Add a new class to the generated model in UModel
- Merge the updated project with the external code.

All the files used in this tutorial are initially available in the **C:\Documents and Settings\All Users\Application Data\Altova** folder. When any single user starts the application for the first time, the **example** files for that user are copied to **C:\Documents and Settings\<username>\My Documents\Altova\UModel2016\UModelExamples** folder. Therefore do not move, edit, or delete the example files in the initial ...All Users\.... directory.

BankView-start.ump

is the UModel project file that constitutes the initial state of the tutorial sample. Several model diagrams as well as classes, objects, and other model elements exist at this stage. Working through the tutorial adds new packages, model diagrams and many other elements that will acquaint you with the ease with which you can model applications using UModel. Please note that the syntax check function reports errors and warnings on this file, the tutorial shows you how to resolve these issues.

BankView-finish.ump

is the UModel project file that constitutes final state of the tutorial sample, if you have worked through it step by step. This project file is the one used when generating code and synchronizing it with UModel.

- The **OrgChart.zip** file supplied in the folder is used for the round-trip engineering process. Please unzip it in the ...**UModelExamples** folder before starting the section.

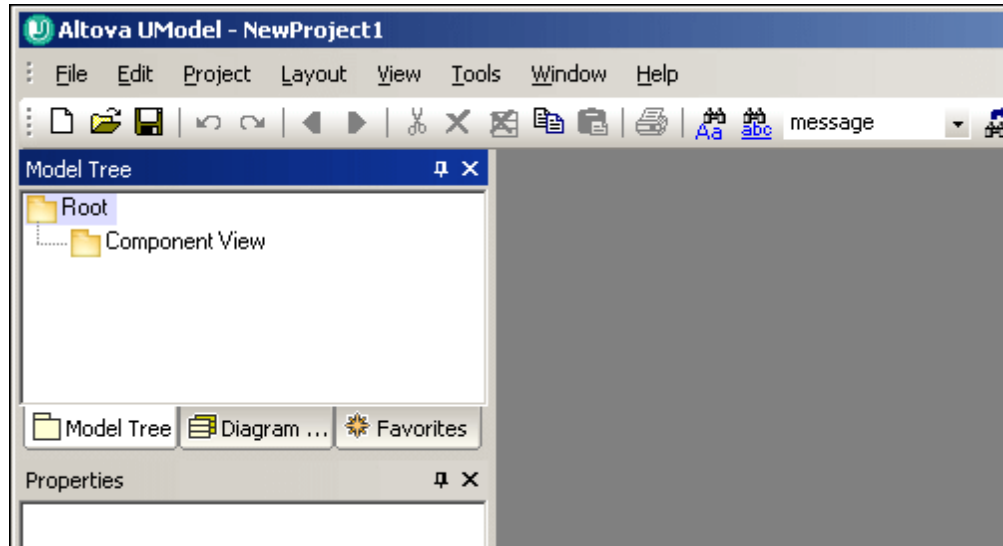
Additional example files for both **Java** and **C#** programming languages are also available in the same directory, i.e. **Bank_Java.ump**, **Bank_CSharp.ump** and **Bank_MultiLanguage.ump**. These project files also contain [Sequence diagrams](#) which are described later in this documentation.

A section describing how to start a [project from scratch](#) and generate code, is included in the Projects and code engineering section.

3.1 Starting UModel

Having installed UModel on your computer:

1. Start UModel by double-clicking the UModel icon on your desktop, or use the **Start | All Programs** menu to access the UModel program.
UModel is started with a default project "NewProject1" visible in the interface.

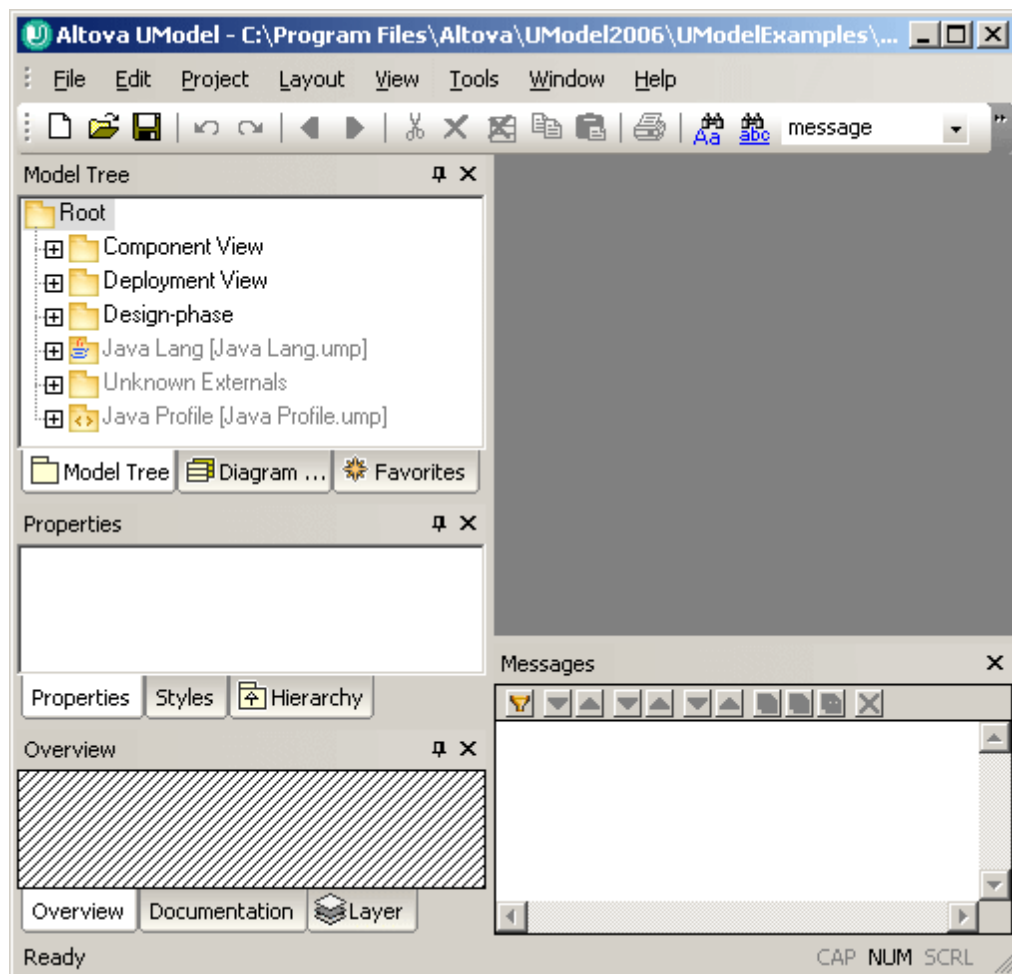


Note the major parts of the user interface: the three panes on the left hand side and the empty diagram pane at right.

Two default packages are visible in the Model Tree tab, "Root" and "Component View". These two packages cannot be deleted or renamed in a project.

To open the BankView-start project:

1. Select the menu option **File | Open** and navigate to the ...\\UModelExamples\\Tutorial folder of UModel. Note that you can also open a *.ump file through a URL, please see [Switch to URL](#) for more information.
2. Open the **BankView-start.ump** project file.
The project file is now loaded into UModel. Several predefined packages are now visible under the Root package. Note that the main window is empty at the moment.



The Model Tree pane displays various views of your modeling project:

- The **Model Tree** tab contains and displays all modeling elements of your UModel project. Elements can be directly manipulated in this tab using the standard editing keys as well as drag and drop.
- The **Diagram Tree** tab allows you quick access to the modeling diagrams of you project wherever they may be in the project structure. Diagrams are grouped according to their diagram type.
- The **Favorites** tab is a user-definable repository of modeling elements. Any type of modeling element can be placed in this tab using the "Add to Favorites" command of the context menu.

The Properties pane displays two views of specific model properties:

- The **Properties** tab displays the properties of the currently selected element in the Model Tree pane or in the Diagram tab. Element properties can be defined or updated in this tab.
- The **Styles** tab displays attributes of diagrams, or elements that are displayed in the Diagram view. These style attributes fall into two general groups: Formatting and display settings.
- The **Hierarchy** tab displays all relations of the currently selected modeling item, in two

different views. The modeling element can be selected in a modeling diagram, the Model Tree, or in the Favorites tab.

The Overview pane displays various tabs:

- The **Overview** tab which displays an outline view of the currently active diagram.
- The **Documentation** tab which allows you to document your classes on a per-class basis.
- The **Layer** tab allows you to define multiple layers for any UModel diagram. Single, as well as multiple, layers can be shown, locked and hidden. Layers allow you to make logical groupings of modeling elements on a diagram.

Modeling element icon representation in the Model Tree

Package types:

-  UML Package
-  Java namespace root package
-  C# namespace root package
-  Visual Basic root package
-  XML Schema root package
-  Java, C#, VB code package (package declarations are created when code is generated)

Diagram types:

 Activity diagram	 Package diagram
 Class diagram	 Profile diagram
 Communication diagram	 Sequence diagram
 Component diagram	 State Machine diagram
 Composite Structure diagram	 Timing diagram
 Deployment diagram	 Use Case diagram
 Interaction Overview diagram	 XML Schema diagram
 Object diagram	 Business Process Modeling Notation

Element types:



An element that is currently visible in the active diagram is displayed with a blue dot at its base. In this case a class element.

-  Class Instance/Object
 -  Class instance slot

-  Class
 -  Property
 -  Operation
 -  Parameter

-  Actor (visible in active use case diagram)
-  Use Case

-  Component
-  Node
-  Artifact
-  Interface

-  Relations (/package)
-  Constraints

3.2 Use cases

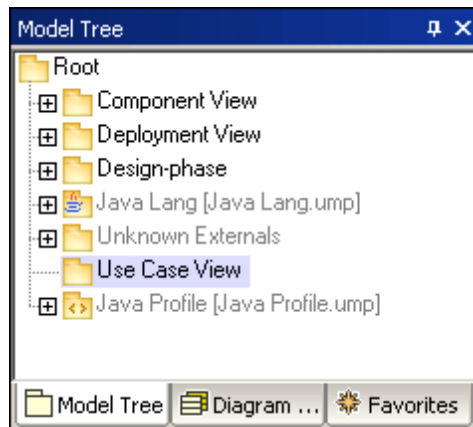
Altova web site:  [UML Use Case diagrams](#)

The aim of this tutorial section is to:

- Add a new **package** to the project
- Add a new Use Case **diagram** to the project
- Add use case **elements** to the diagram, and define the dependencies amongst them
- Align and size elements in the diagram tab.

To add a new package to a project:

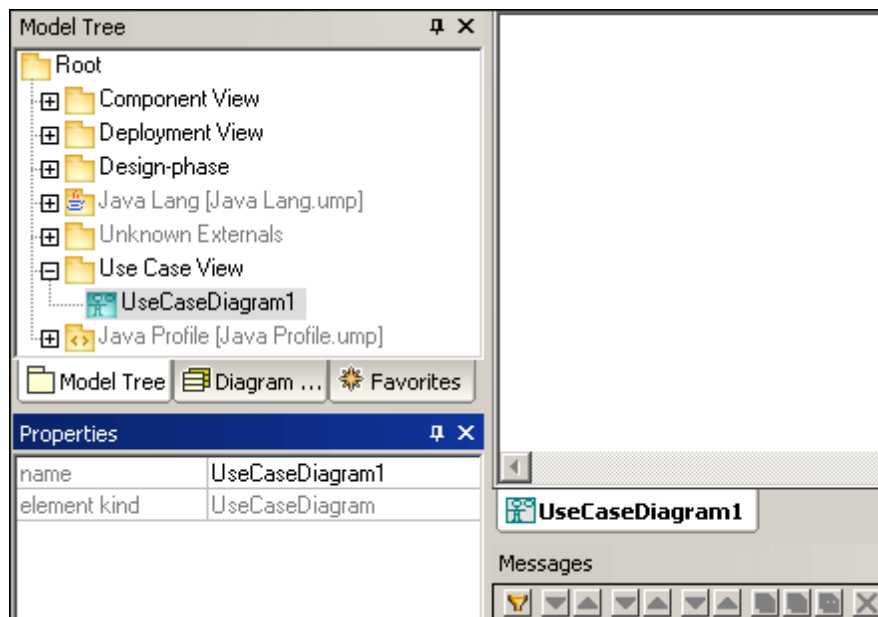
1. Right click the **Root** package in the Model Tree tab, and select **New Element | Package**.
2. Enter the name of the new package e.g. **Use Case View**, and press Enter.



Please see [Packages](#) for more information on packages and their properties.

Adding a diagram to a package:

1. Right click the previously created Use Case View package.
2. Select **New Diagram | UseCase Diagram**.



A Use Case diagram has now been added to the package in the Model Tree view, and a diagram tab has been created in the diagram pane. A default name has been provided automatically.

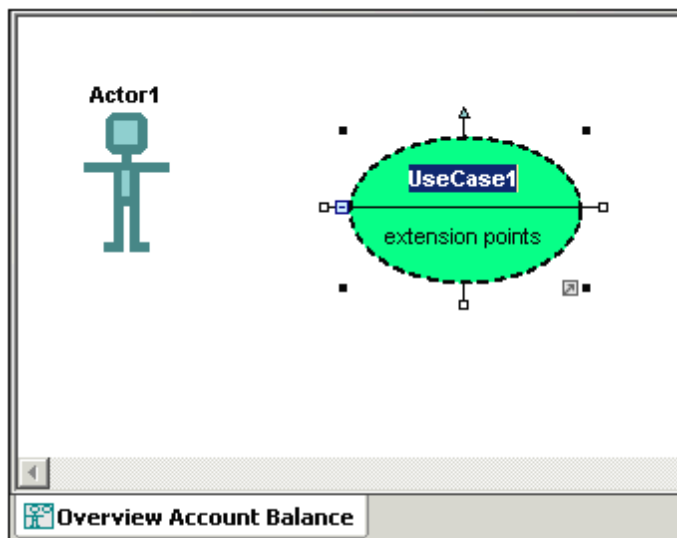
3. Double click the supplied name, in the Model Tree tab, change it to "Overview Account Balance", and press Enter to confirm.



Please see [Diagrams](#) for more information on diagrams and their properties.

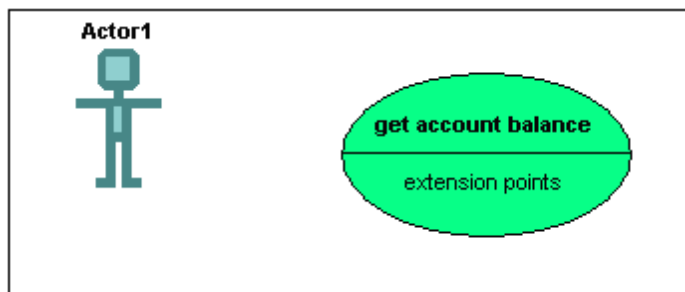
Adding Use case elements to the Use Case diagram:

1. Right click in the newly created diagram and select **New | Actor**.
The actor element is inserted at the click position.
2. Click the Use Case icon  in the icon bar and click in the diagram tab to insert the element.
A UseCase1 element is inserted. Note that the element, and its name, are currently selected, and that its properties are visible in the Properties tab.



3. Change the title to "get account balance", press Enter to confirm. Double click the title if it is deselected.

Note that the use case is automatically resized to adjust to the text length.

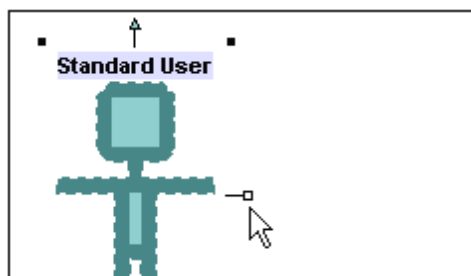


Model elements have various connection handles and other items used to manipulate them.

Note: Use **Ctrl+Enter** to create a **multi-line** use case name.

Manipulating UModel elements: handles and compartments

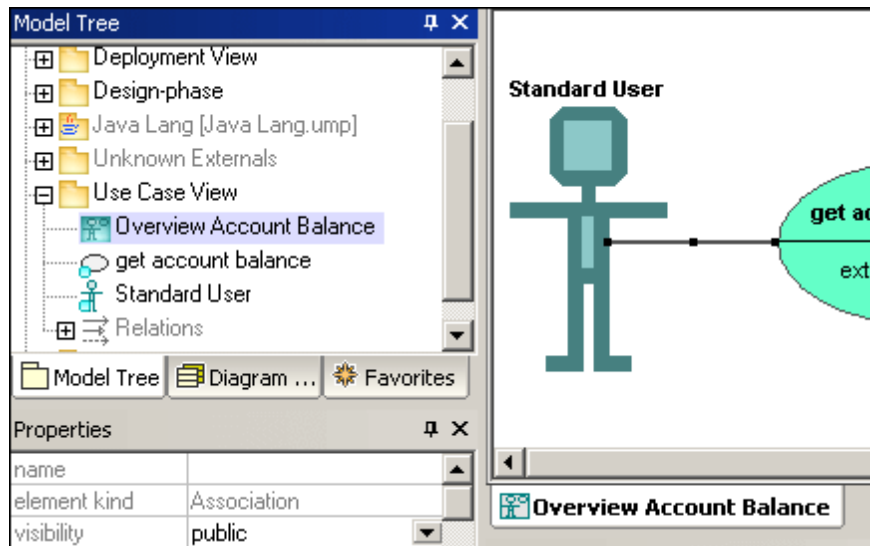
1. Double click the Actor1 text, of the Actor element, change the name to "Standard User" and press Enter to confirm.
2. Place the mouse cursor over the **"handle"** to the right of the actor. A tooltip containing "Association" appears.



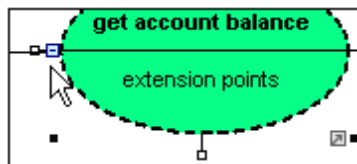
3. Click the handle, drag the Association line to the right, and drop it on the "get account balance" use case.

An association has now been created between the actor and the use case. The

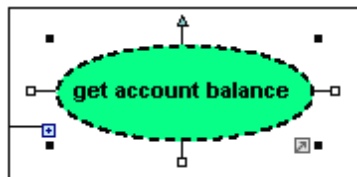
association properties are also visible in the Properties tab. The new association has been added to Model Tree under the Relations item of the Use Case View package.



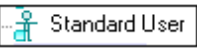
- Click the use case and drag it to the right to reposition it.
The association properties are visible on the association object.
- Click the use case to select it, then click the **collapse icon** on the left hand edge of the use case ellipse.



The **extension points** compartment is now hidden.



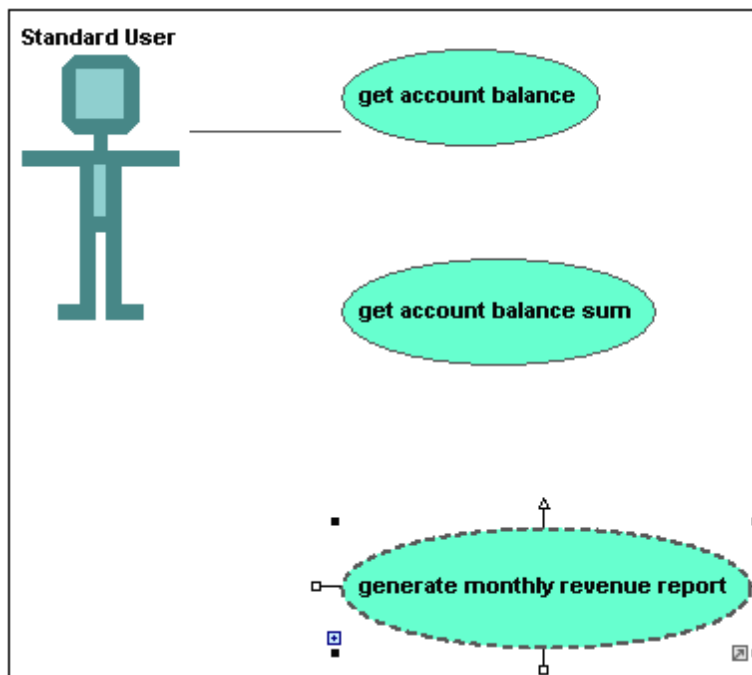
Please note:

A blue dot next to an element icon , in the Model Tree tab, signifies that the element is visible in the current diagram tab. Resizing the actor adjusts the text field which can be multi line. A line break can be inserted into the text using **Ctrl+Enter**.

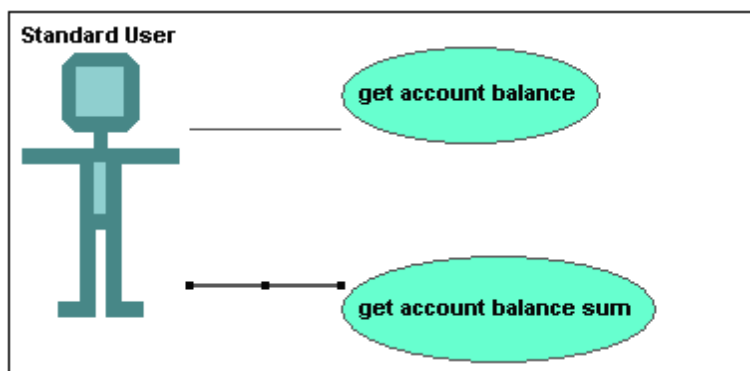
Finishing up the use case diagram:

Using the methods discussed above:

- Click the Use Case icon in the icon bar and **simultaneously** hold down the **Ctrl** key.
- Click at two different vertical positions in the diagram tab to add two more use cases, then release the **Ctrl** key.
- Name the first use case "get account balance sum" and the second, "generate monthly revenue report".
- Click on the collapse icon of each use case to hide the extensions compartment.

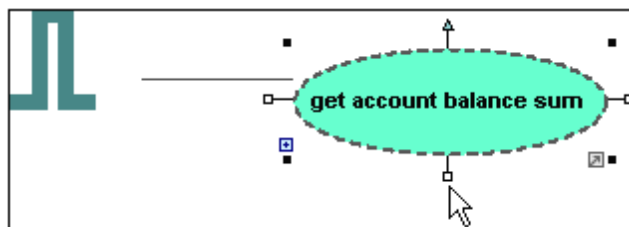


5. Click the actor and use the association handle to create an association between Standard user and "get account balance sum".

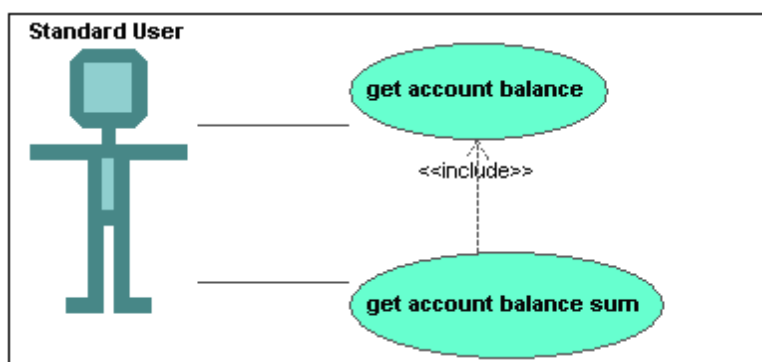


To create an "Include" dependency between use cases (creating a subcase):

1. Click the **Include** handle of the "get account balance sum" use case, at the bottom of the ellipse, and drop the dependency on "get account balance".



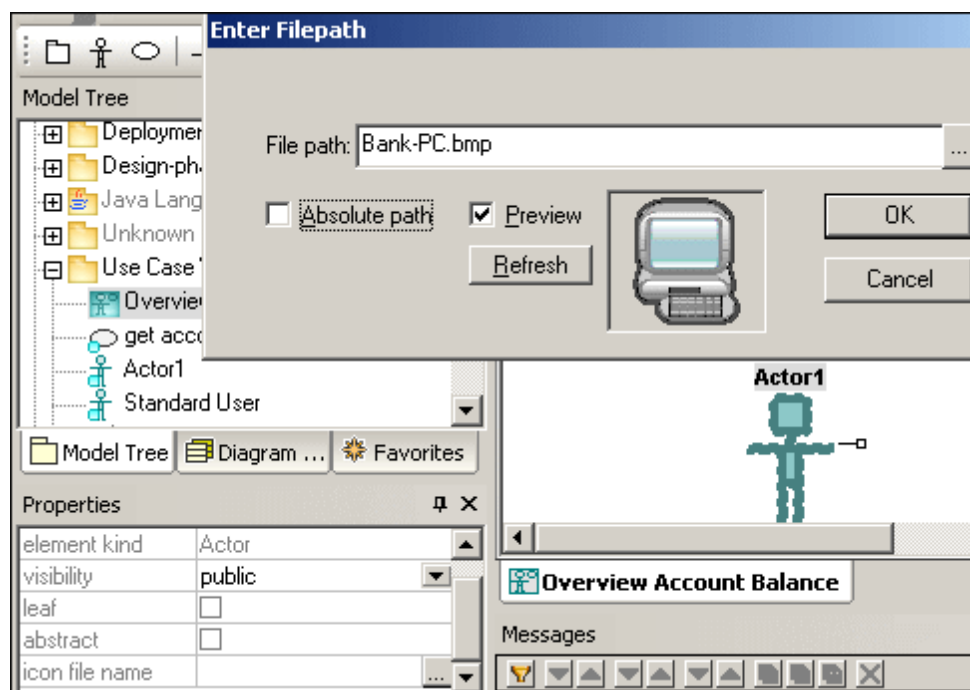
An "include" dependency is created, and the include stereotype is displayed on the dotted arrow.



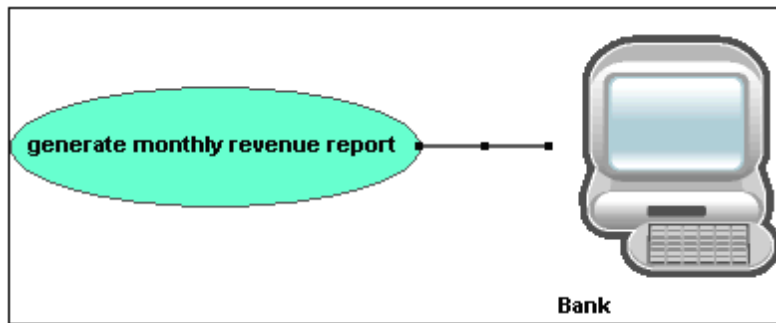
Inserting user-defined / customized actors:

The actor in the "generate monthly revenue report" use case is not a person, but an automated batch job run by a Bank computer.

1. Insert an actor into the diagram using the Actor icon in the icon bar.
2. Rename the actor to Bank.
3. Move the cursor over to the Properties tab, and click the browse  icon next to the "icon file name" entry.
4. Click the Browse icon to select the user-defined bitmap, Bank-PC.bmp.
5. Deselect the "Absolute Path" check box to make the path relative. Preview displays a preview of the selected file in the dialog box.



6. Click OK to confirm the settings and insert the new actor.
7. Move the new Bank actor to the right of the lowest use case.
8. Click the Association icon  in the icon bar and drag from the Bank actor to the "generate monthly revenue report" use case. This is an alternative method of creating an association.

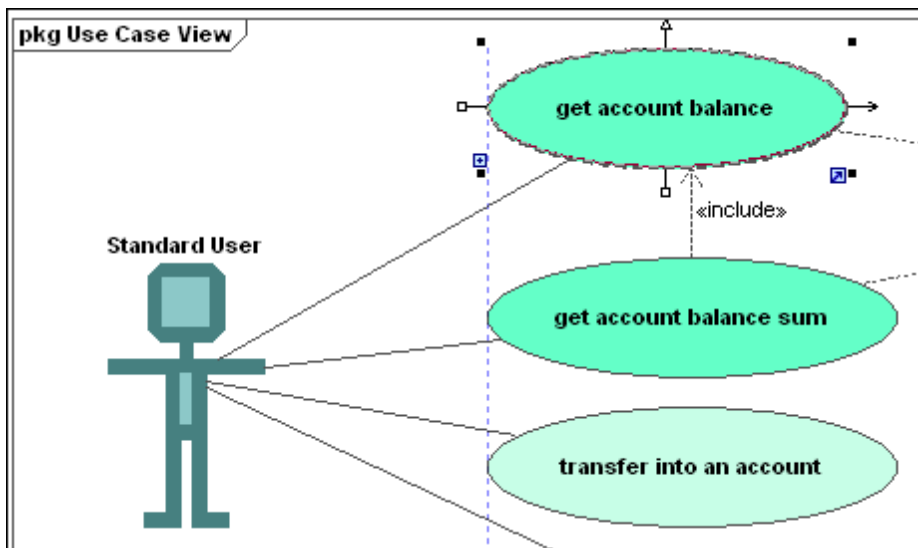


Please note:

The background color used to make the bitmap transparent has the RGB values 82.82.82.

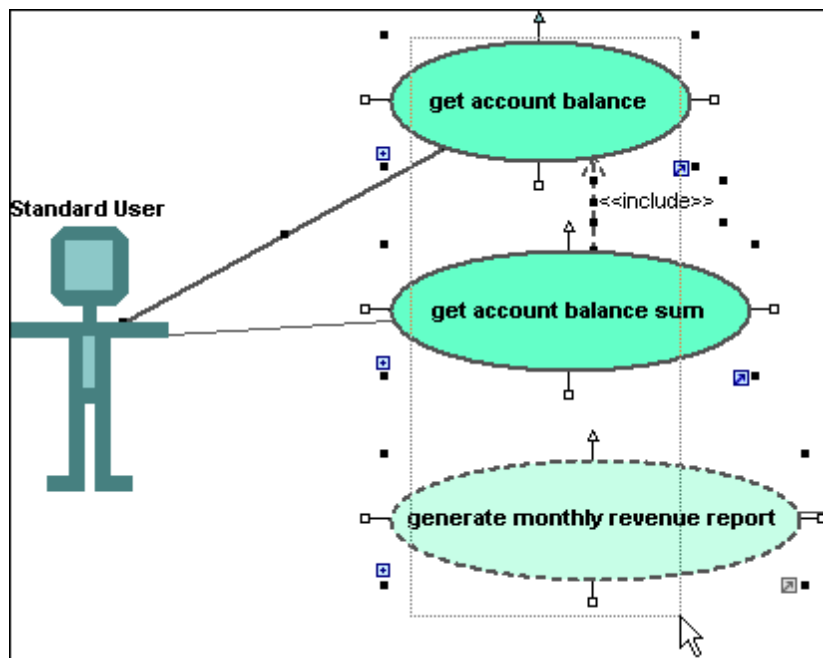
Dragging elements - Snap lines

When dragging components in a diagram, guide lines appear allowing you to align an element to any other element in the diagram. This option can be enabled/disabled using the menu option **Tools | Options | View - Alignment group "Enable Snap lines"**.



Aligning and adjusting the size of elements:

1. Create a selection marquee by dragging on the diagram background, making sure that you encompass all three use cases starting from the top.
Note that the last use case to be marked, is shown in a dashed outline in the diagram, as well as in the Overview window.

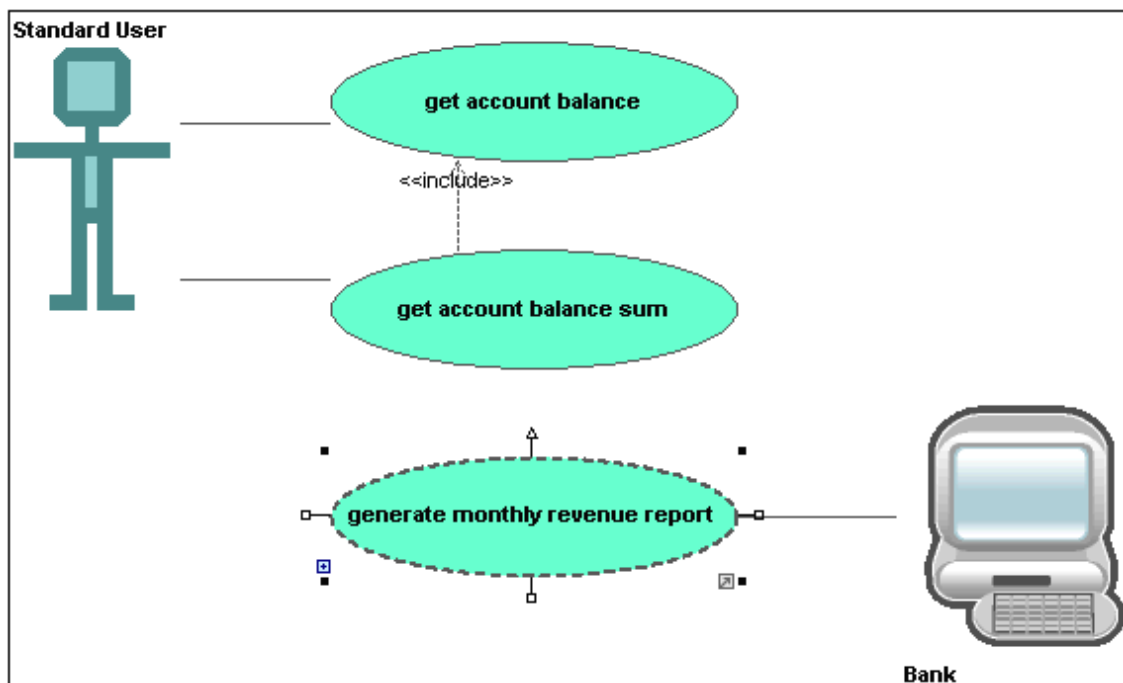


All use cases are selected, with the lowest being the basis for the following adjustments.

2. Click the Make same size icon  in the title bar.
3. Click the **Center Horizontally** icon  to line up all the ovals.
The use case elements are all centered and of the same size.

Please note:

You can also use the **Ctrl** key to select multiple elements.



3.3 Class Diagrams

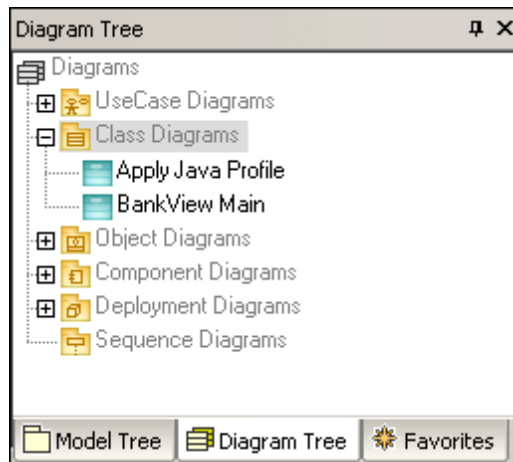
Altova web site:  [UML Class diagrams](http://www.altova.com/uml-class-diagrams)

The aim of this tutorial section is to:

- Add a new abstract class called Account, as well as attributes and operations
- Create a **composite** association from Bank to Account

To open a different diagram in UModel:

1. Click the Diagram Tree tab.
2. Expand the Class Diagrams package to see its contents.



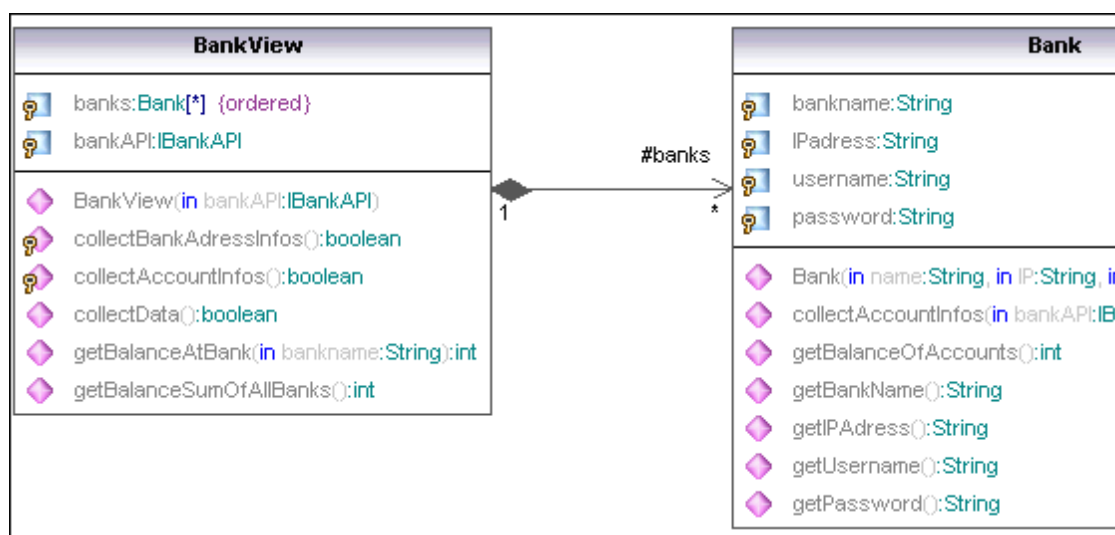
All class diagrams contained in the project are displayed.

3. Double click the  **BankView Main** diagram icon.
The Class diagram appears as a tab in the working area.

Please note:

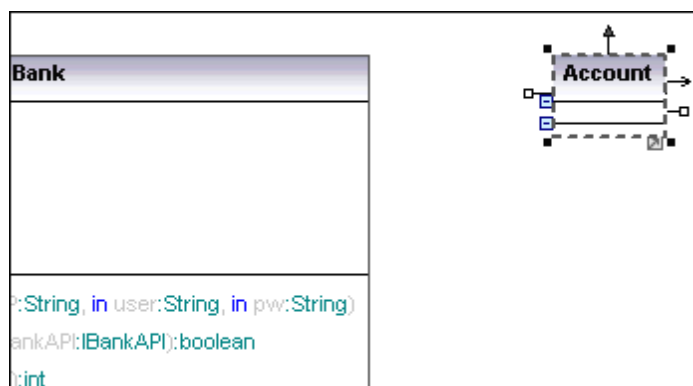
You could of course, double click the Class diagram icon in the Model Tree tab below the BankView package to achieve the same thing.

Two concrete classes with a composite association between them, are visible in the class diagram.



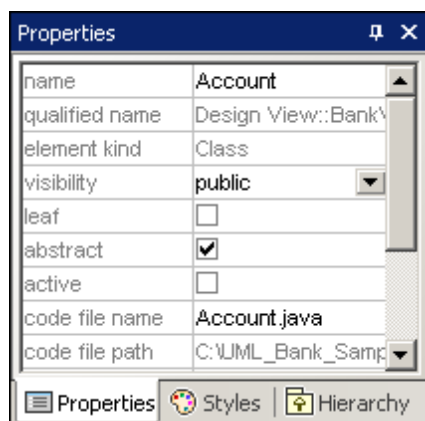
To add a new class and define it as abstract:

1. Click the class icon  in the icon bar, then click to the right of the Bank class to insert it.
2. Change the Class1 name to e.g. "**Account**", press Enter to confirm, (double click the name if it becomes deselected).

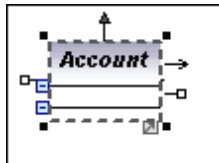


Note that the Properties tab displays the current class properties.

3. Click the "**abstract**" check box in the Properties pane to make the class abstract.
4. Click in the "code file name" text box, and enter **Account.java** to define the Java class.

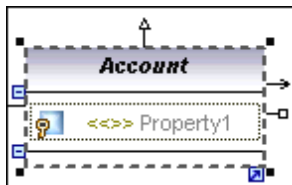


The class title is now displayed in italic, which is the identifying characteristic of abstract classes.

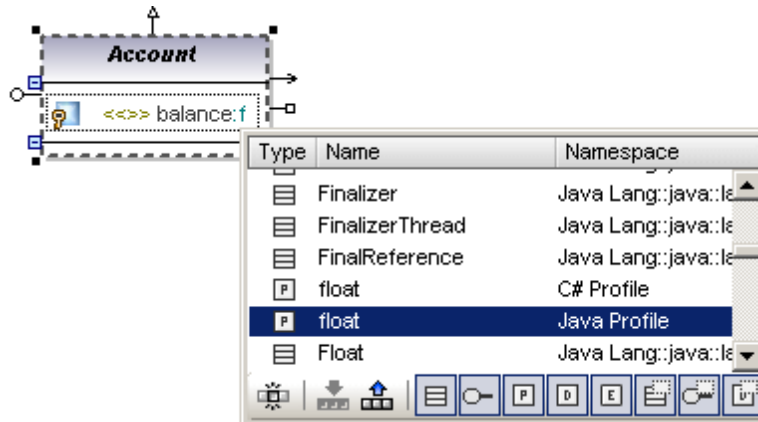


To add properties to a class:

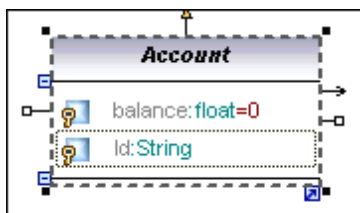
1. Right click the Account class and select **New | Property**, or press the **F7** key. A default property "Property1" is inserted with stereotype identifiers <<>>.



2. Enter the Property name "**balance**", and then add a colon character ":".
3. Enter the "**f**" character through the keyboard, and press Enter to insert the return value datatype "float". Please note that drop-down lists are case sensitive!



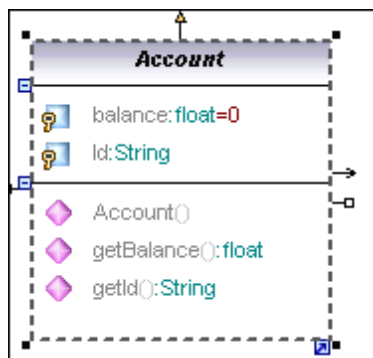
4. Continue on the same line by appending "**=0**" to define the default value.
5. Press the **F7** keyboard key to add a second property to the class.
6. Enter **Id:** and select **String** from the drop-down list.



To add operations to a class:

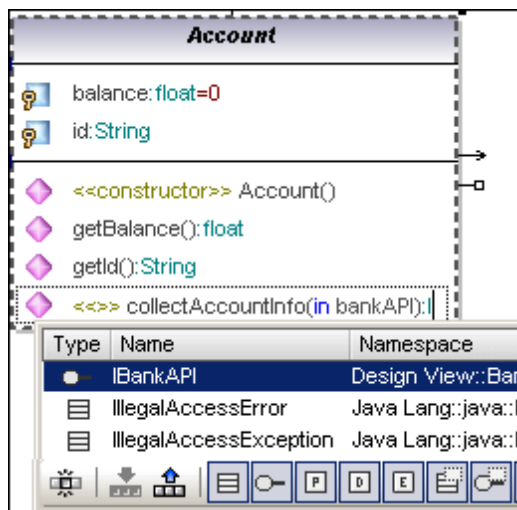
1. Right click the Account class and select **New | Operation**, or press the **F8** key.
2. Enter **Account()** as the constructor. Using the method described above:

3. Add two more operations namely **getBalance:float** and **getId:String**.

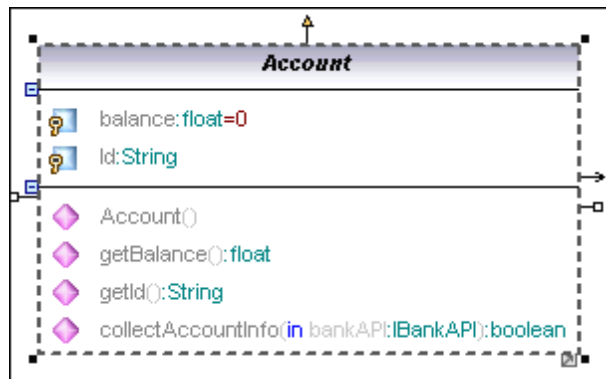


Using the **autocomplete function** while defining operations:

4. Create another operation, using F8, **collectAccountInfo** and enter the open parenthesis character "(".
5. Select "in" from the drop-down list, enter a "space" character, and continue editing on the same line.
6. Enter "bankAPI" and then a colon.
7. Select **IBankAPI** from the drop-down list, add the close parenthesis character ")", and enter a colon ":".



8. Press the "b" key to select the Boolean data type, then Enter to insert it.
9. Press Enter to end the definition.



Please note:

Clicking the **visibility icon** to the left of an operation , or property , opens a drop-down list enabling you to change the visibility status. Note that these visibility icons can be changed to the UML conformant symbols.

Deleting class properties and operations from a Class Diagram:

1. Press F8 then Enter, to add a default operation "Operation1" in the Account class.
2. Click Operation1 and press the **Del** key to delete it.

A delete prompt appears asking if you want to delete the element from the project. Click Yes to delete Operation1 from the **class** as well as from the **project**.

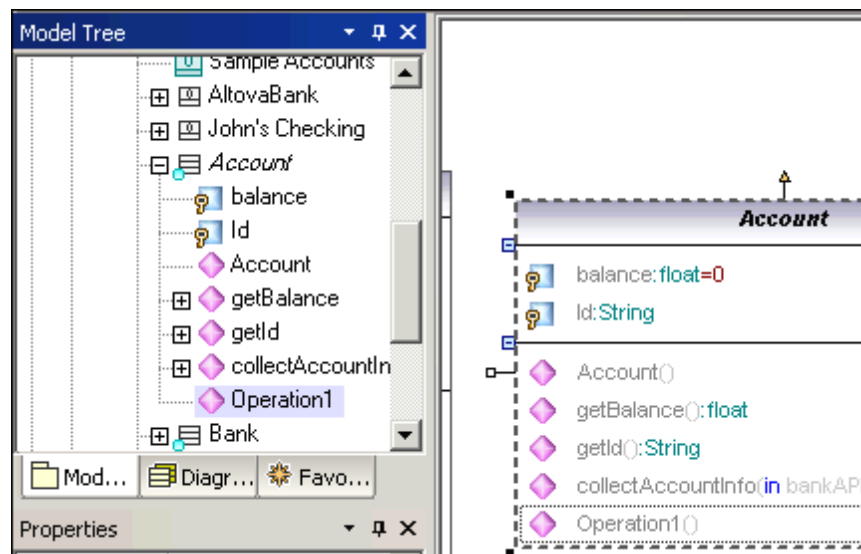
Please note:

If you only want to delete the operation from the class in the diagram, but **not** from the **project**, press the **Ctrl+Del**. key. You can also enable a prompt that queries you when deleting objects, please see "[Tools | Editing](#)" for more information.

Finding (deleting) class properties and options from the Model Tree:

Properties and options can also be deleted directly from the Model Tree. To do this safely, it is important to first find the correct property. Assuming you have inserted "Operation1" in the Account class (press F8, then Enter to insert):

1. Right click Operation1 in the Account class.
2. Select the option "**Select in Model Tree**" or press F4.
The Operation1 item is now highlighted under *Account* in the Model Tree tab.



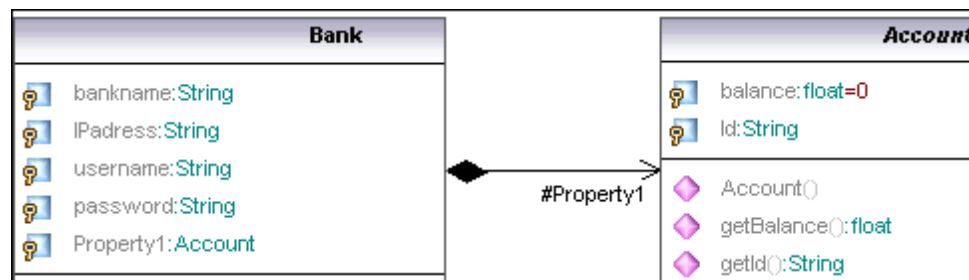
3. Press the **Del** key to delete the operation from the **class** and **project**!
Note that almost any modeling element can be found in the Model Tree when pressing F4.

Please note:

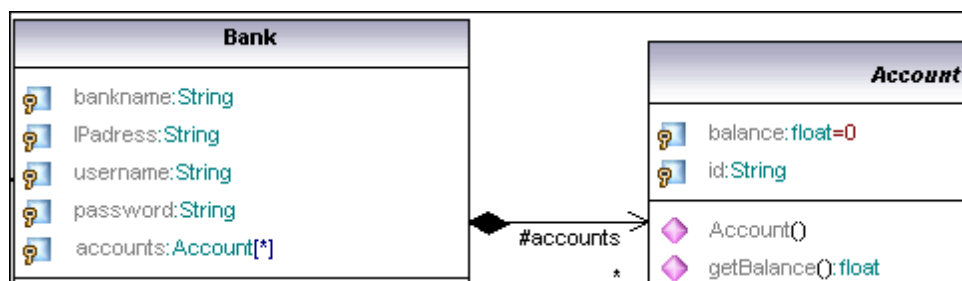
It is also possible to navigate from the Properties pane to the Model Tree when viewing an attributes properties, please see: the [Properties](#) in the User Interface section.

Creating an composition association between the Bank and Account classes:

1. Click the Composition icon  in the title bar, then drag from the **Bank** class to the **Account** class. The class is highlighted when the association can be made. A new property (**Property1:Account**) is created in the Bank class, and an composite association arrow joins the two classes.



2. Double click the new **Property1** entry in the Bank class and change it to "**accounts**", being sure not to delete the Account type definition (displayed in teal/green).
3. Press the End keyboard key to place the text cursor at the end of the line, and
4. Enter the open square bracket character "[" and select "*" from the dropdown list, to define the **multiplicity**, and press Enter to confirm.



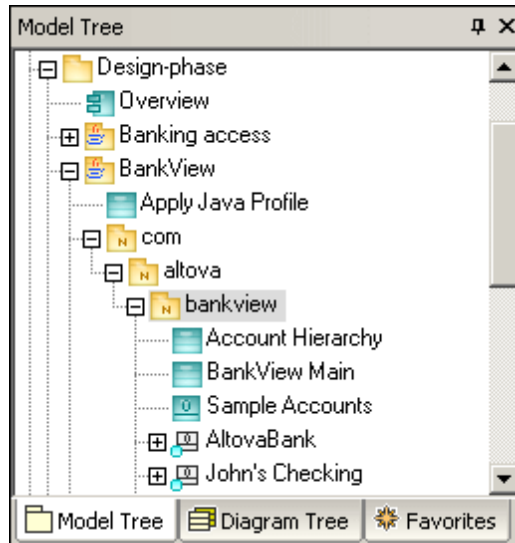
3.3.1 Creating derived classes

The aim of this tutorial section is to:

- Add a new **Class diagram** called Account Hierarchy to the project
- Insert existing classes, and create a new Savings account class
- Create three **derived** classes of the abstract base class Account, using Generalizations

To create a new Class Diagram:

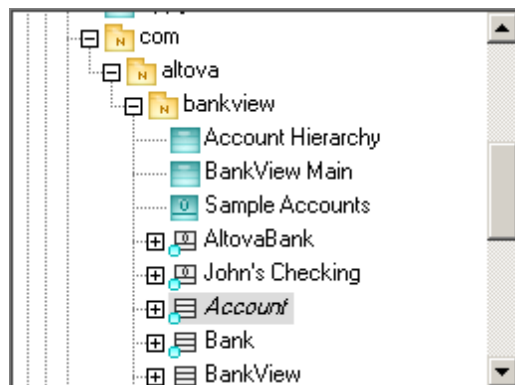
1. Right click the bankview **package** (under **Design-phase | BankView | com | altova**) in the Model Tree tab, and select **New Diagram | Class Diagram**.
2. Double click the new ClassDiagram1 entry and rename it to "Account Hierarchy", and press Enter to confirm.



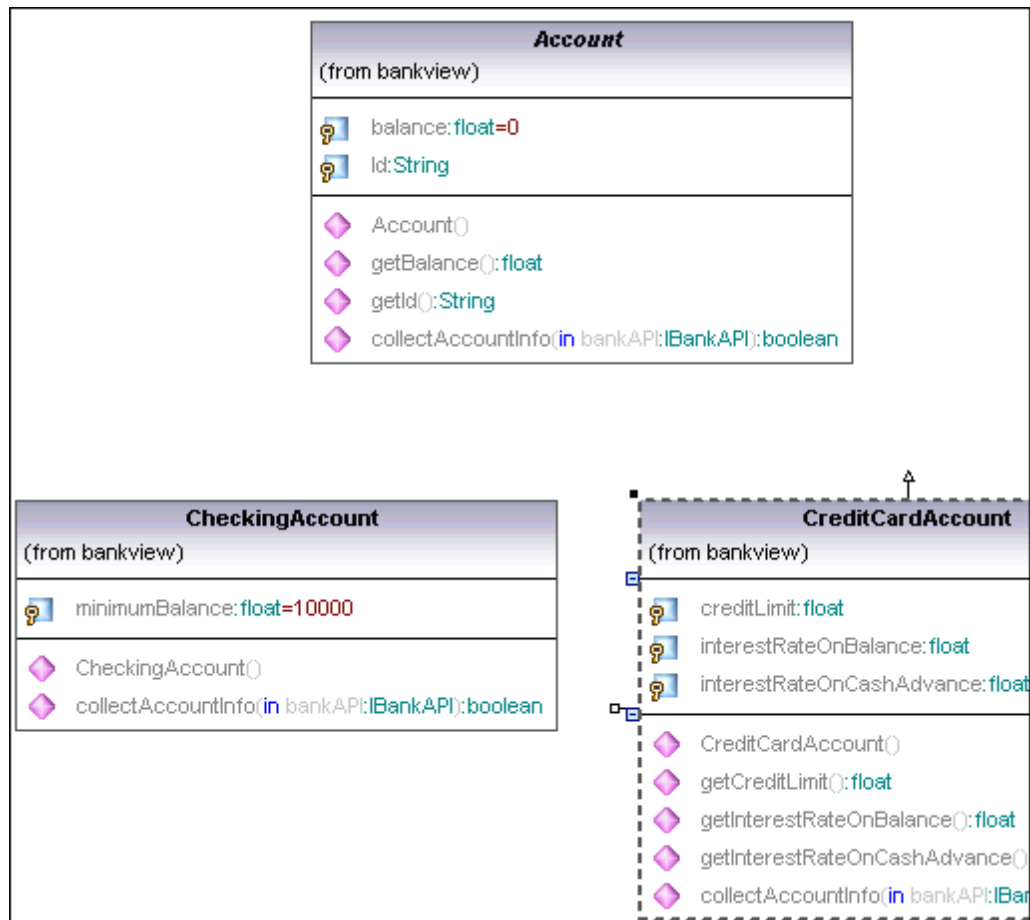
The Account Hierarchy tab is now visible in the working area.

Inserting existing classes into a diagram:

1. Click the **Account class** in the bankview **package** (under **com | altova | bankview**), and

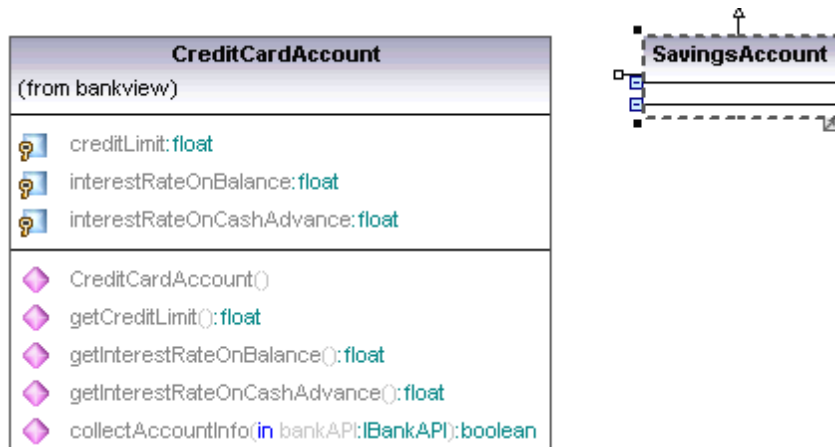


2. Drag it into the Account Hierarchy tab.
3. Click the **CheckingAccount** class (of the same package) and drag it into the tab.
4. Place the class below and to the left of the Account class.
5. Use the same method to insert the **CreditCardAccount** class. Place it to the right of the CheckingAccount class.



Adding a new class:

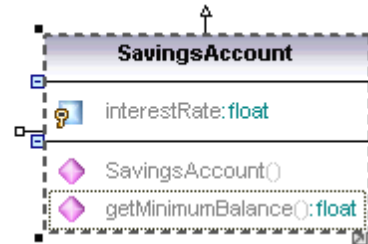
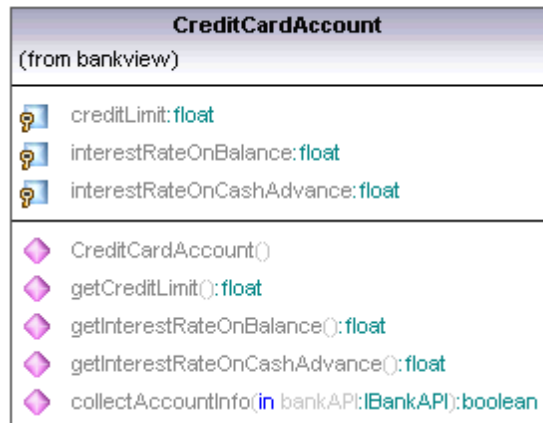
1. Right click the diagram background (to the right of CreditAccountClass) and select **New | Class**.
A new class is automatically added to the correct package, i.e. bankview which contains the current class diagram **Account Hierarchy**.
2. Double click the class name and change it to **SavingsAccount**.



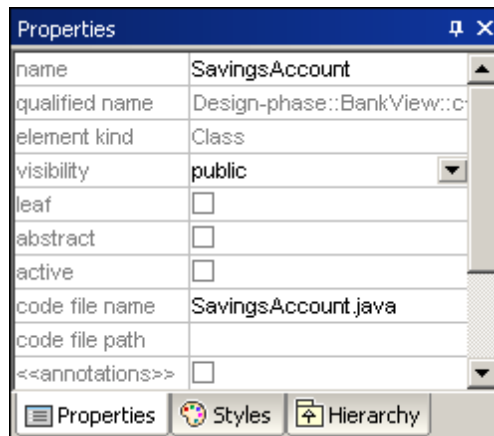
3. Press the **F7** key to add a new **property**.
4. Enter "**interestRate**", then a colon, and press "**f**" to select the float datatype from the

dropdown list and press Enter twice to select and confirm the entry.

5. Press **F8** and add the **operation/constructor SavingsAccount()**.
6. Use the same method, F8, to add the operation **getMinimumBalance:float**.



7. Click in the "code file name" text box, in the Properties tab, and enter **SavingsAccount.java** to define the Java code class.



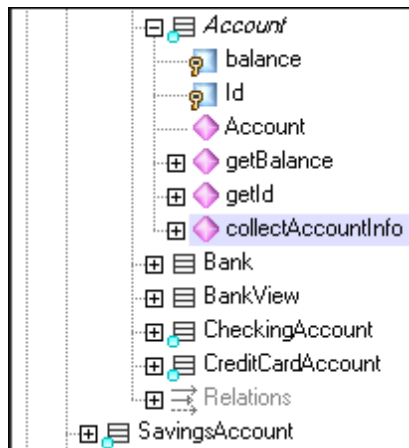
Reusing/copying existing Properties/Operations:

Properties and operations can be directly copied, or moved, from one class to another. This can be achieved using drag and drop, as well as the standard keyboard shortcuts:

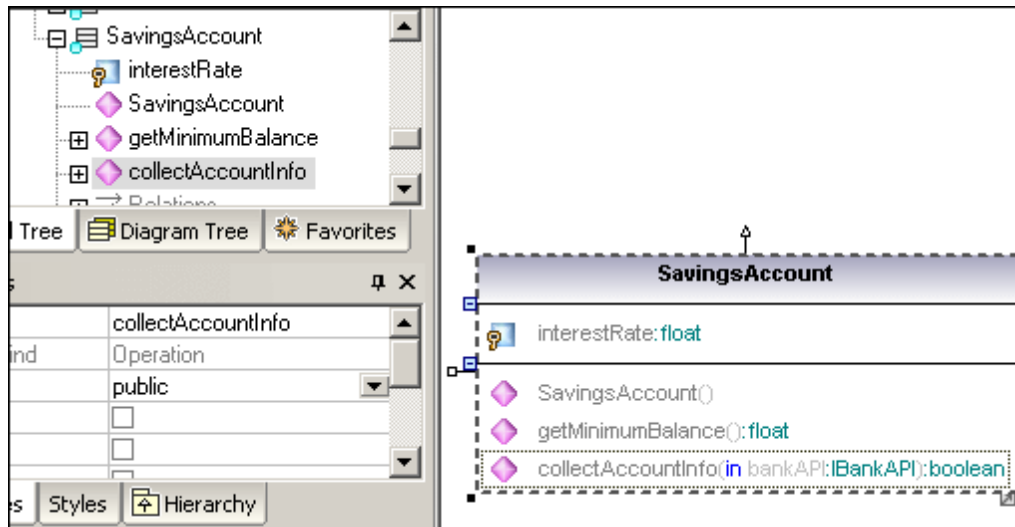
- within a **class** in the diagram tab
- between **different** classes in the diagram tab
- in the Model Tree view
- between different UML diagrams, by dropping the copied data onto a different diagram tab.

Please see "[Cut, copy and paste in UModel Diagrams](#)" for more information.

1. Expand the **Account** class in the **Model Tree**.
2. Right click the **collectAccountInfo** operation and select **Copy**.



3. Right click the **SavingsAccount** class in the Model Tree and select **Paste**. The operation is copied into the SavingsAccount class, which is automatically expanded to display the new operation.



The new operation is now also visible in the SavingsAccount class in the Class Diagram.

Please note:

You can use the Copy/Paste keyboard shortcuts (**Ctrl + C**, **Ctrl + V**), as well as drag and drop in the Model Tree to achieve the same effect. You might have to disable the [sort options](#) to drop the operation between specific items.

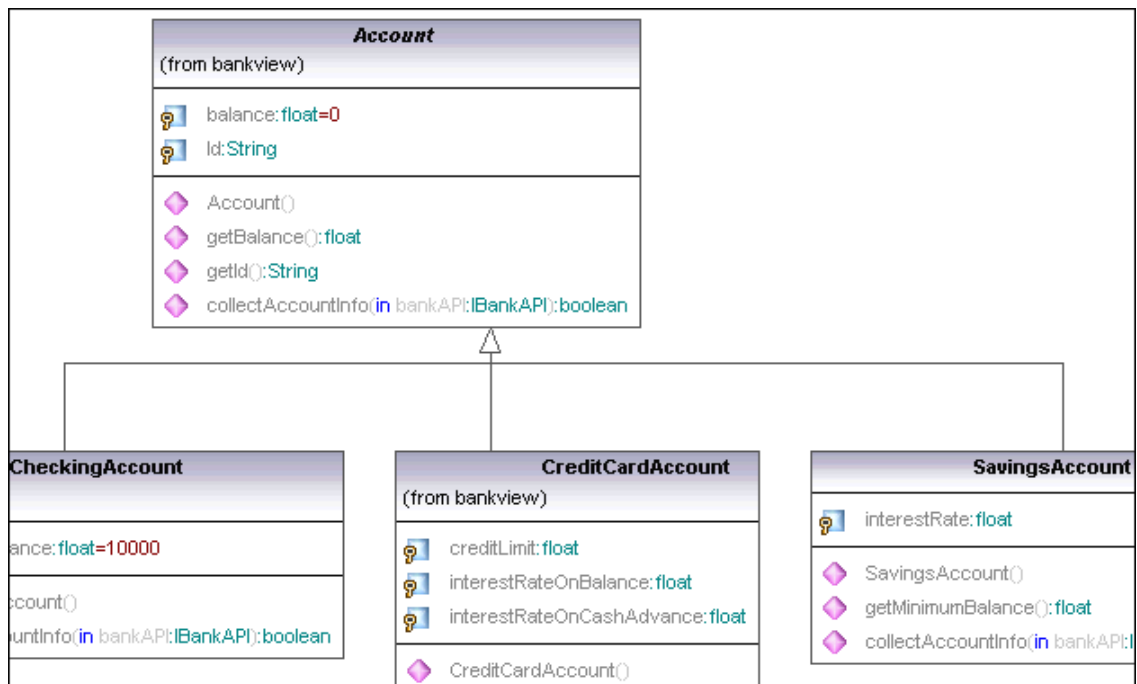
Creating derived classes - Generalization/Specialization:

At this point the class diagram contains the abstract class, Account, as well as three specific Account classes. We now want to define, or create a generalization/specialization relationship between Account and the specific classes i.e. to create three derived concrete classes.

1. Click the Generalization icon  in the icon bar and **hold down** the **Ctrl** key.
2. Drag from **CreditCardAccount** (the class in the middle) and drop on the Account class.
3. Drag from the **CheckingAccount** class and drop the **arrowhead** of the previously created generalization.
4. Drag from the **SavingsAccount** class and drop the arrowhead of the previously created

generalization: release the Ctrl key at this point.

- Generalization arrows are created between the three subclasses, and the Account superclass.



3.4 Object Diagrams

Altova web site:  [UML Object diagrams](#)

The aim of this tutorial section is to:

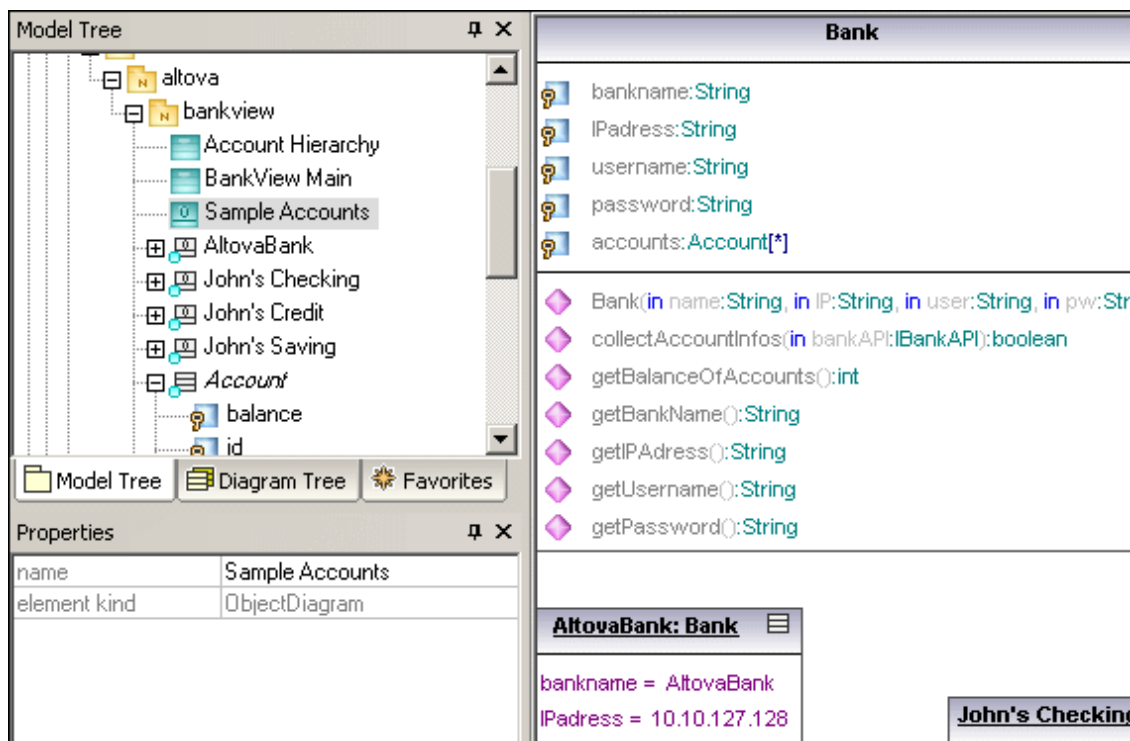
- Show how **class** and **object** diagrams can be combined in one diagram, to give you a snapshot of the objects at a given point of time
- Create **Objects/Instances** and define the relationships between them
- Format association/links
- Enter real-life data into objects/instances

To open the Object diagram:

1. Double click the **Sample Accounts** diagram icon under the **bankview** package (or under Object Diagrams in the Diagram Tree tab).

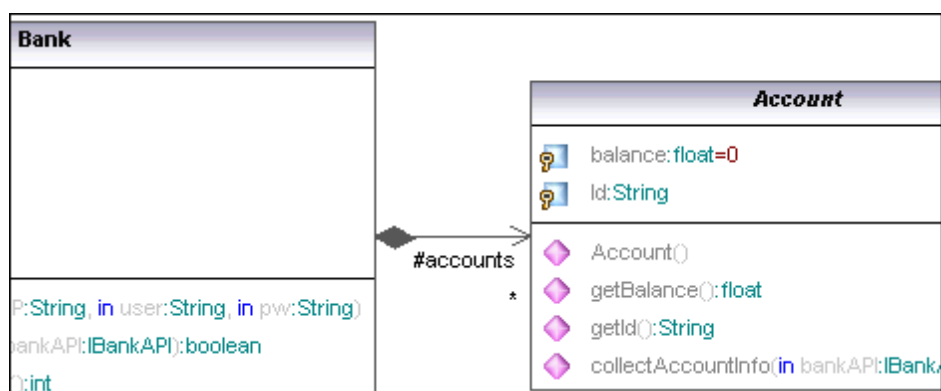
The Bank class and two related objects/instances are displayed in the object diagram.

AltovaBank:Bank is the object/instance of the Bank class, while **John's checking: CheckingAccount** is an instance of the class CheckingAccount.



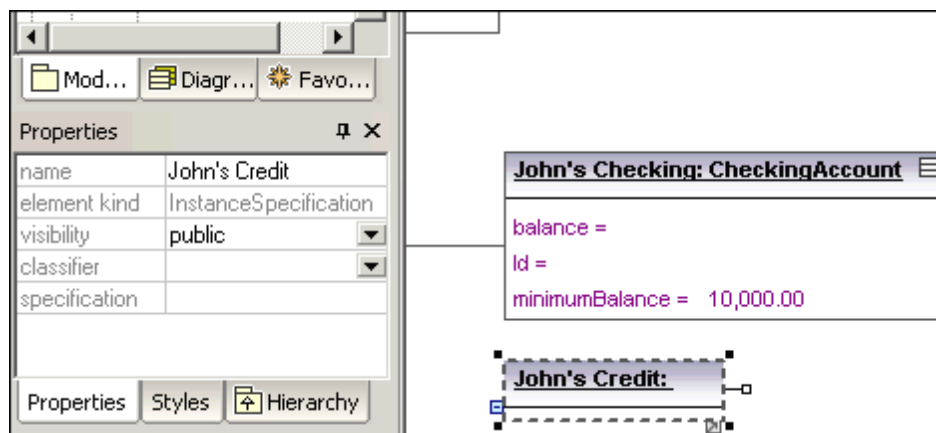
Inserting a class into an Object diagram:

- Click the **Account** class icon  in the **Model Tree**, and drag it into the "Sample Accounts" tab. The composite association defined previously, in BankView Main diagram, is automatically created.



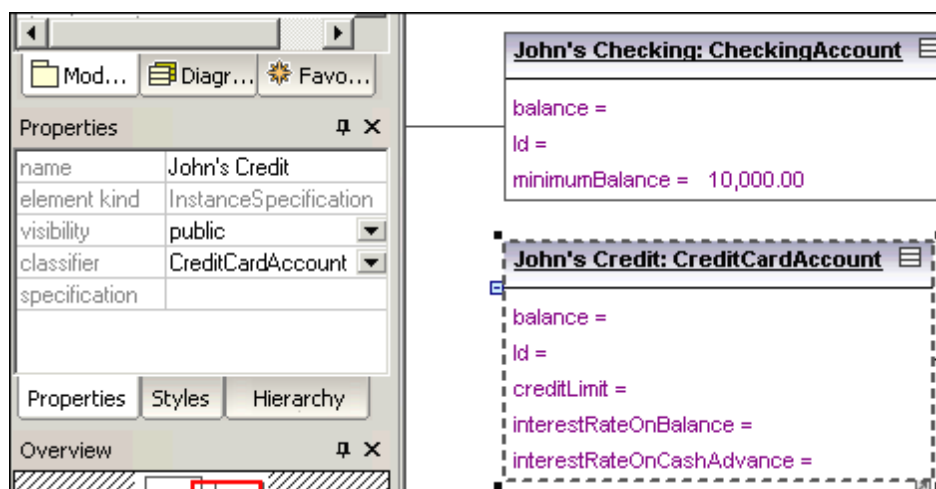
To add a new object/instance by selecting its type:

1. Click the **InstanceSpecification** icon  in the icon bar, then click under the John's Checking object in the diagram tab.
2. Change the name of the instance to **John's Credit**, and press Enter.



While the instance is active, all its properties are visible in the Properties tab.

3. Click the **classifier** combo box and select the entry **CreditCardAccount** from the drop-down list.

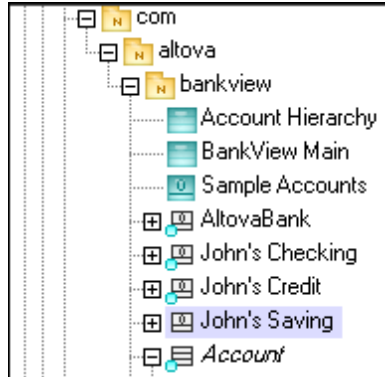


Note that right clicking an instance specification and selecting [Show/Hide Node](#)

[content](#) allows you show/hide object content.

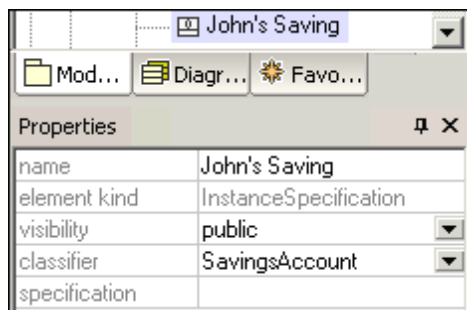
To add a new object in the Model Tree view (then insert it into a diagram):

1. Right click the **bankview** package in the **Model Tree** tab, and select **New element | InstanceSpecification**.
2. Change the default object name to **John's Saving**, and press Enter to confirm.
The new object is added to the package and sorted accordingly.

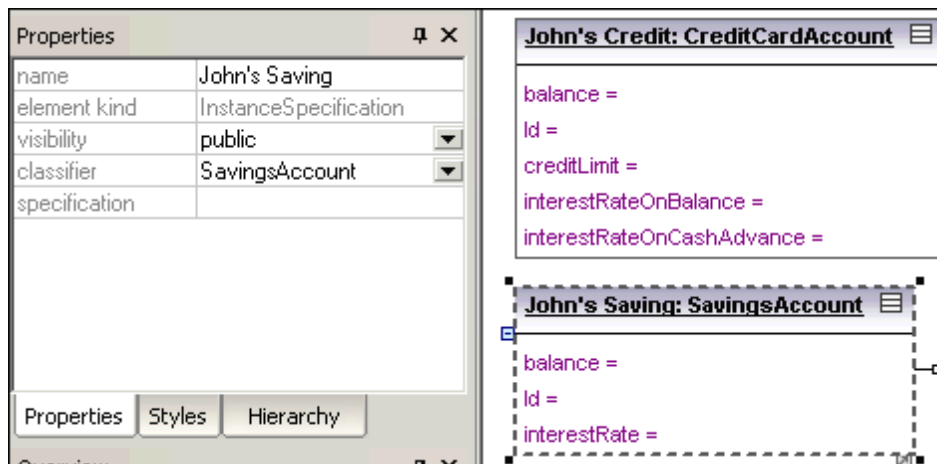


While the object is still selected in the Model Tree tab,

3. Click the **classifier** combo box, in the **Properties** tab, and select **SavingsAccount**.



4. Drag the John's Saving object/instance from the Model Tree tab, into the Sample Accounts tab, placing it below John's credit.



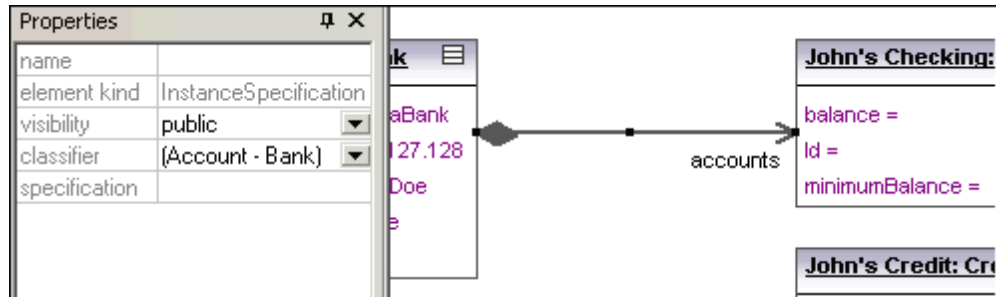
Creating "links" between objects:

Links are the instances of class associations, and describe the relationships between objects/

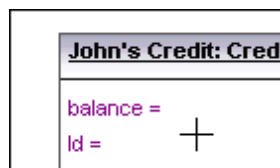
instances at a fixed moment in time.

1. Click the existing link (association) between the **AltovaBank** and John's Checking.
2. In the Properties tab, click the **classifier** combo box and select the entry **Account - Bank**.

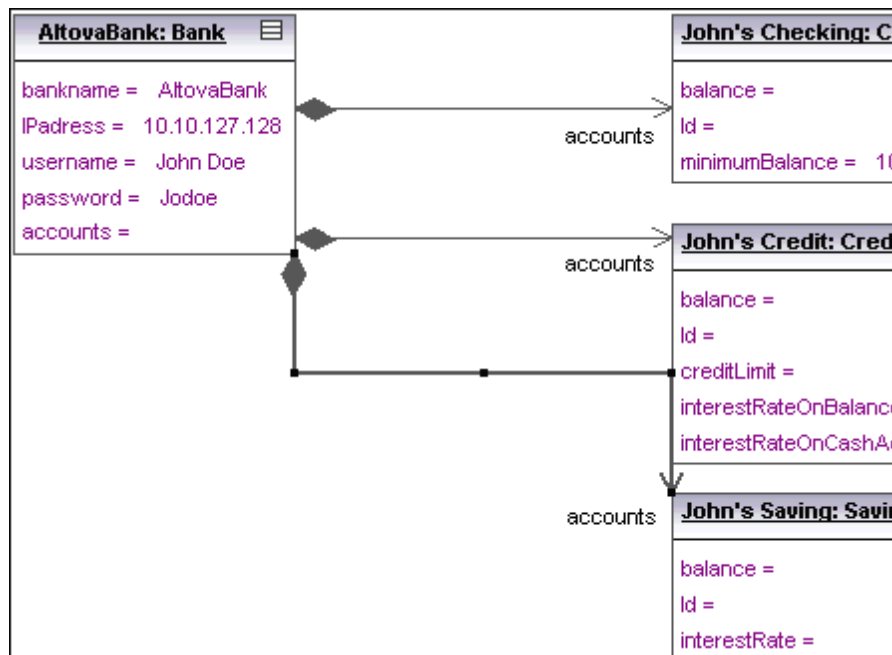
The link now changes to a composite association, in accordance with the class definitions.



3. Click the **InstanceSpecification** icon  in the icon bar, and position the cursor over the John's Credit class.
The cursor now appears as a + sign.



4. Drag from **John's Credit** object to AltovaBank to create a link between the two.
5. Use the **classifier** combo box in the Properties tab to change the link type to **Account - Bank**.
6. Use the method outlined above to create a link between **John's Saving** and AltovaBank.



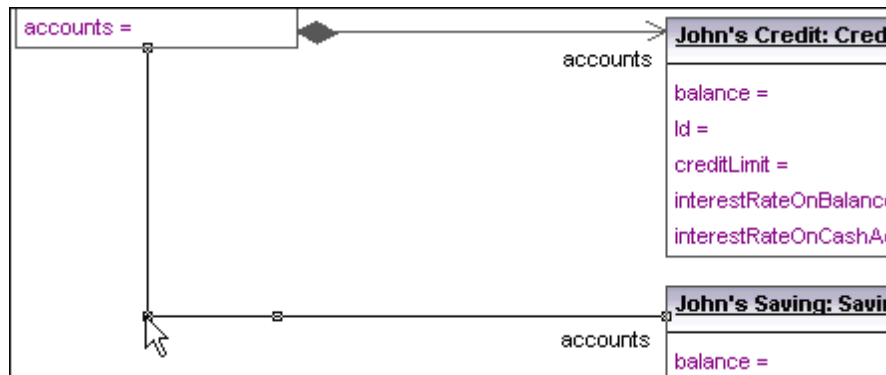
Please note:

Changes made to the association type in any class diagram, are now automatically updated in the object diagram.

Formatting association/link lines in a diagram:

1. Click the lowest link in the diagram, if not active, and drag the corner connector to the left.

This allows you to reposition the line both horizontally and vertically.

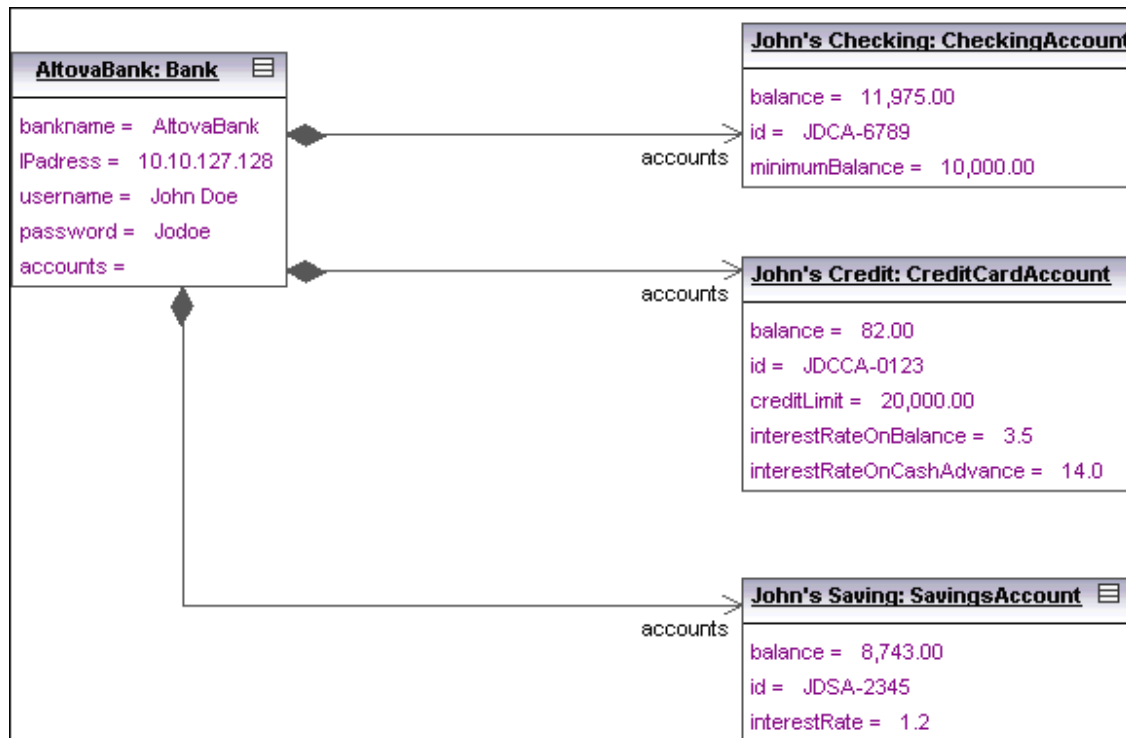


Use this method to reposition links in the diagram tab.

Entering sample data into objects:

The instance value of an Attribute/Property in an object is called a **slot**.

1. Click in the respective slots of each object and enter sample data.
2. E.g. in **John's Checking** object, double click in the **balance** slot and enter 11,975.00 as the balance.
3. Fill in the rest of the data to give yourself an idea of the current instance state.



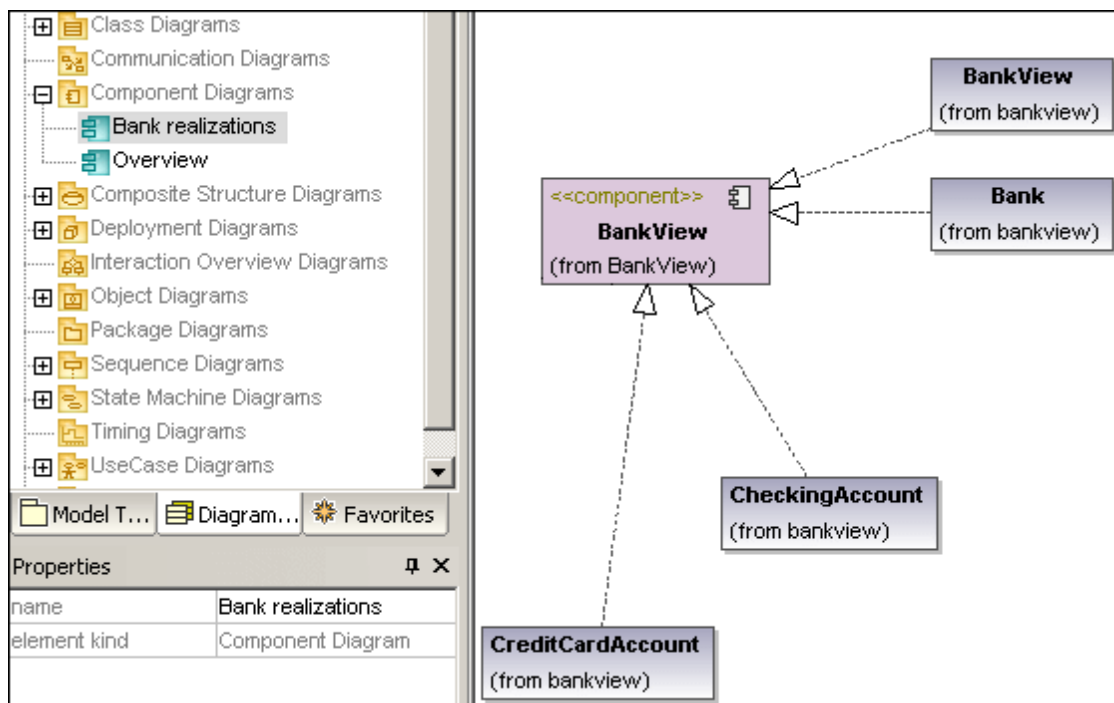
3.5 Component Diagrams

The aim of this tutorial section is to:

- Show how to insert classes into a component diagram
- Create realization dependencies between the classes and the BankView component
- Show how to change line properties
- Insert components into a component diagram, and create usage dependencies to an interface

To open the component diagram:

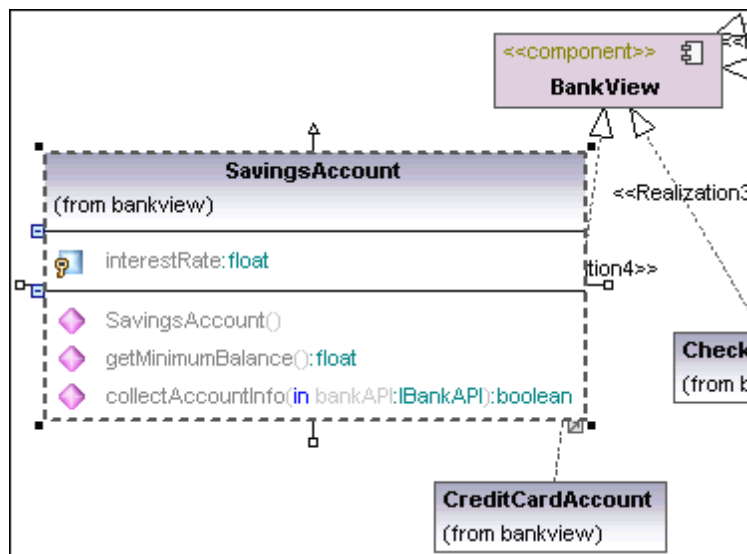
1. Click the Diagram Tree tab, expand the **Component Diagrams** component and double click the "BankView realization" diagram icon.
The "BankView realization" component diagram is displayed.



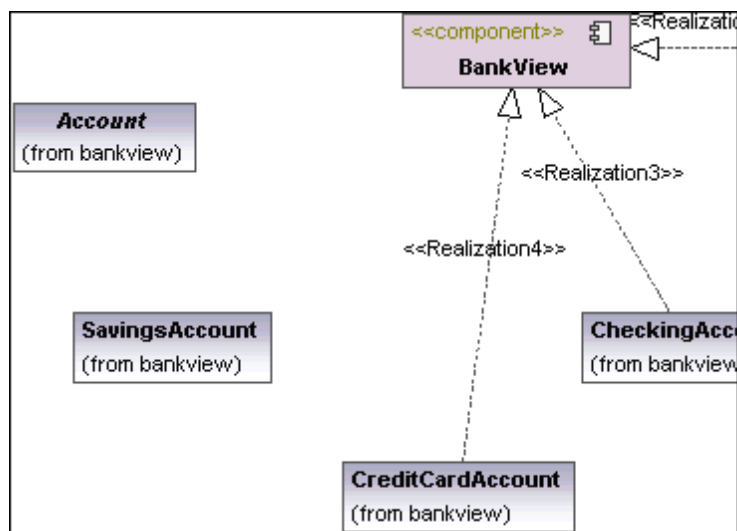
2. Switch back to the Model Tree tab by clicking that tab.

To insert (existing) classes into a component diagram:

1. Locate the **SavingsAccount** class  under the bankview package.
2. Drag it into the component diagram.
The class is displayed with all its compartments.



3. Click both collapse icons to end up with the only the class name compartment.
4. Use the same method to insert the abstract class **Account**.

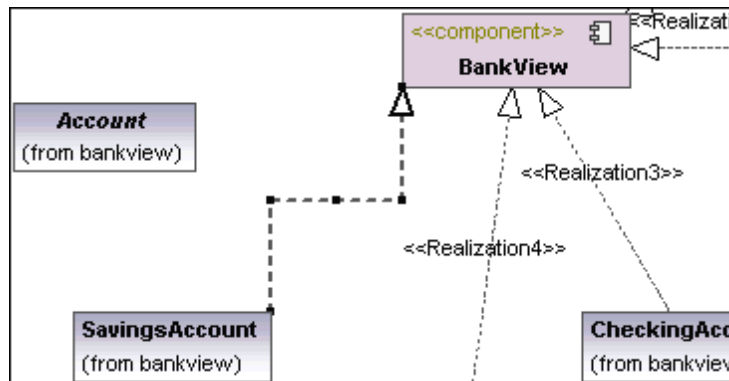


Please note:

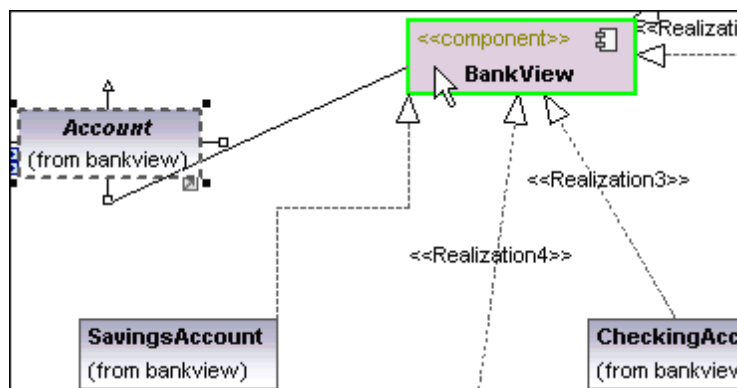
The package containing the inserted class, is displayed in the name compartment in the form "from bankview".

To create Realization dependencies between a class and component:

1. Click the Realization icon  in the icon bar.
2. Drag from **SavingsAccount**, and drop the arrow on the **BankView** component.



3. Click the **ComponentRealization** handle of the Account class (at the base), and drop it on the BankView component.

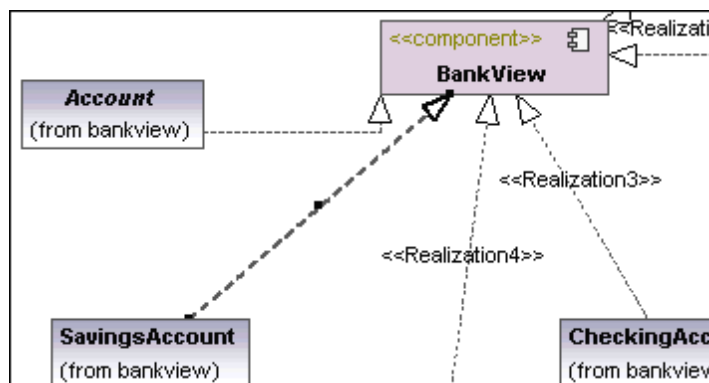


Both of these methods can be used to create realization dependencies. There is another method that allows you to create realization dependencies solely in the Model Tree, please see [Round-trip engineering \(code - model - code\)](#) for more information.

Changing (Realization) line characteristics:

Clicking a dependency or any other type of line in a UModel diagram, activates the line drawing icons in the Layout icon bar.

1. Click the realization line between **SavingsAccount** and BankView.
2. Click the line type icon **Direct line**  in the Layout toolbar.



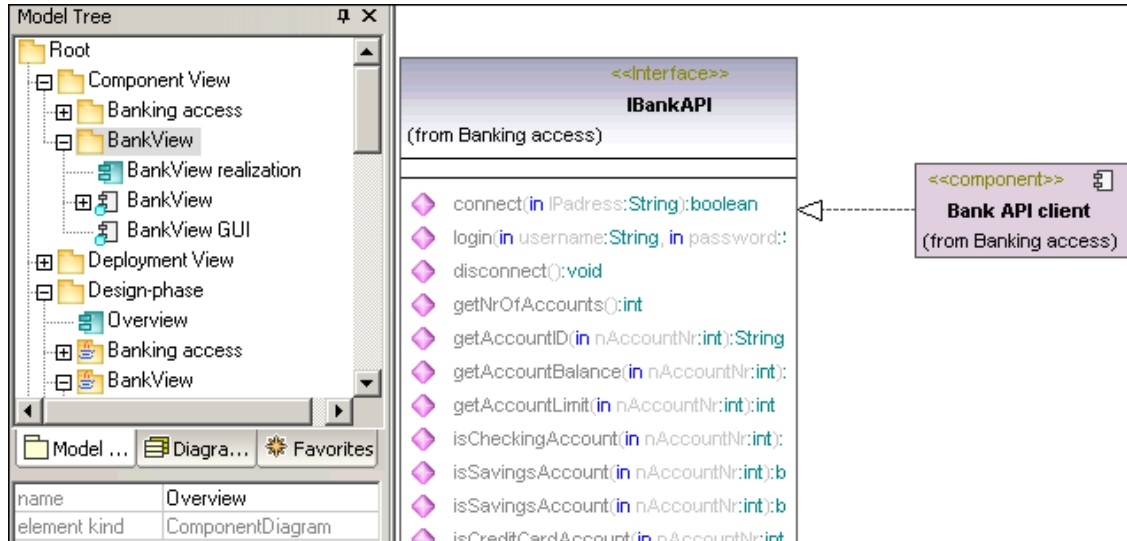
The line properties are immediately altered. Lines have small icons along them called **waypoints**. Waypoints can be clicked and moved to alter line characteristics. Change

the line properties to suit your needs.

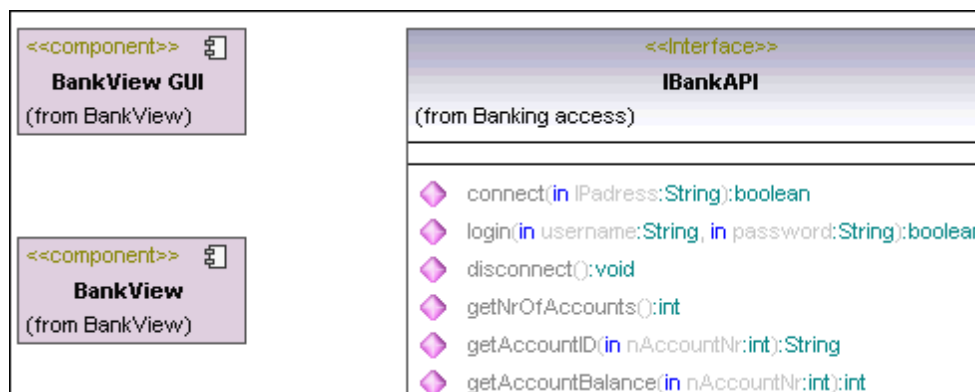
Inserting components and creating usage dependencies:

1. Double click the **Overview** diagram icon directly under the **Design-phase** package in the Model Tree.

The Overview component diagram is opened and displays the currently defined system dependencies between components and interfaces.



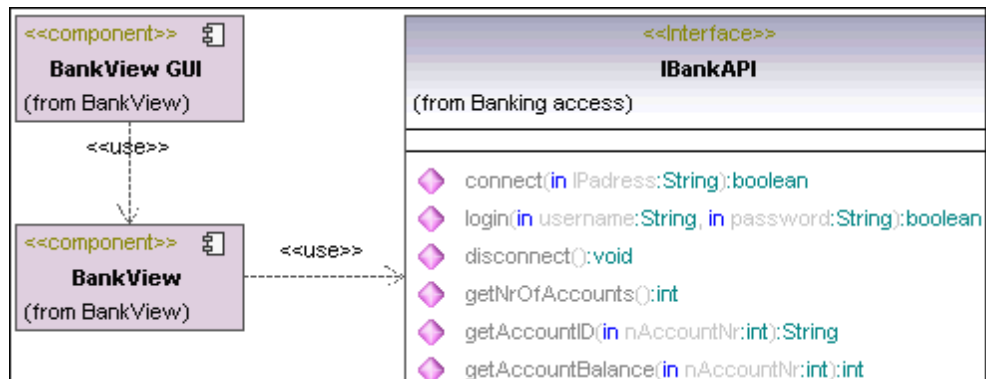
2. Click the **BankView GUI** component under the **Component View | BankView** package in the Model Tree, and drag it into the **Overview** diagram tab.
The package containing the inserted component is displayed in the name compartment, "from BankView".
3. Use the same method to insert the **BankView** component under the same package.



The BankView component is the component produced by the "forward-engineering" process described in this tutorial.

To create a usage dependency between interfaces and components:

1. Click the Usage icon  in the icon bar.
2. Drag from the **BankView GUI** component to the **BankView** component.
3. Click the Usage icon again, and drag from the **BankView** component to the **IBankAPI** interface.



The usage dependency (`<<use>>`) connects a **client** element to a **supplier** element. In this case the **IBankInterfaceAPI** interface uses the services of components **BankView** and **BankView GUI**.

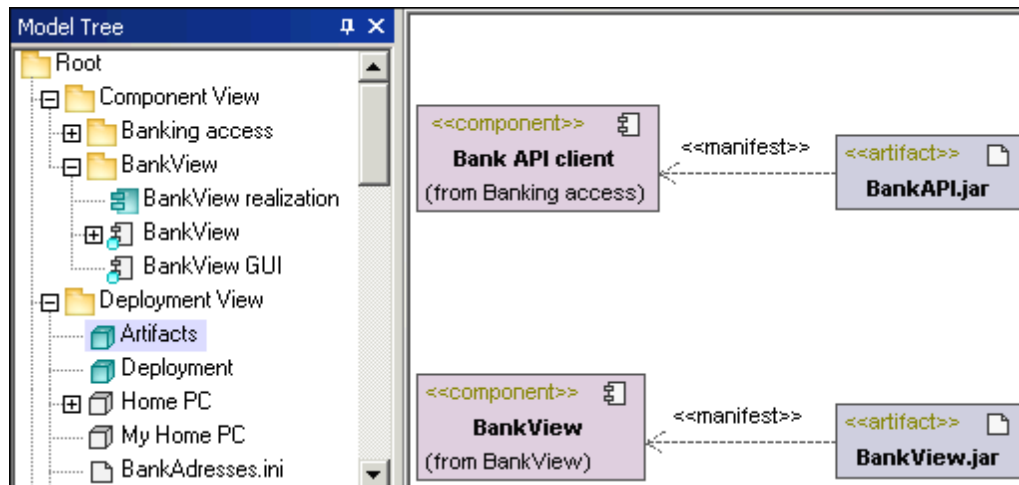
3.6 Deployment Diagrams

The aim of this tutorial section is to:

- Show the artifact manifestation of components
- Add a new node and dependency to a Deployment diagram
- Add artifacts to a node and create relationships between them

To open the Deployment (Artifacts) diagram:

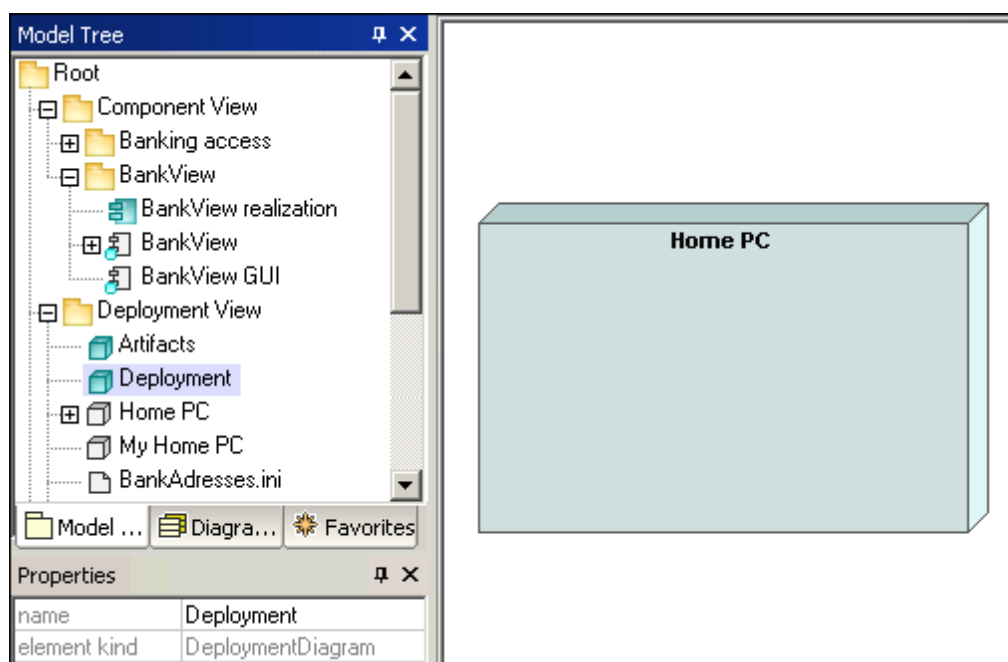
1. Click the Model Tree tab, expand the **Deployment View** diagram package, then double click the **Artifacts** icon.



This diagram shows the manifestation of the **Bank API client** and the **BankView** components, to their respective compiled Java **.jar** files.

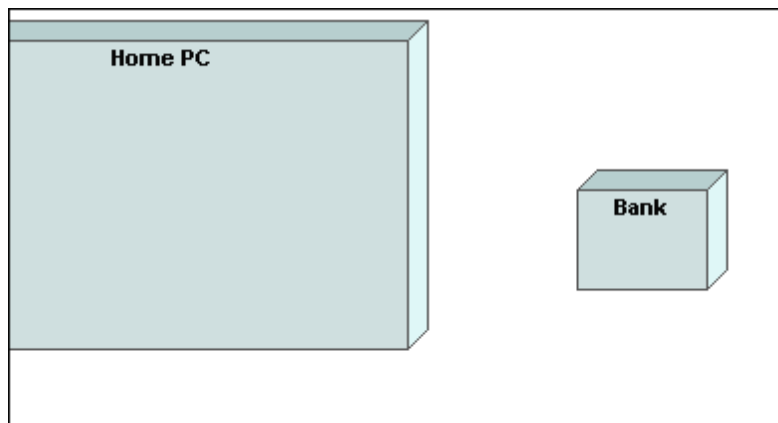
To open the Deployment diagram:

1. Double click the **Deployment** icon under the Deployment View package.
The Deployment diagram is opened and displays the physical architecture of the system, which currently only comprises of the Home PC node.



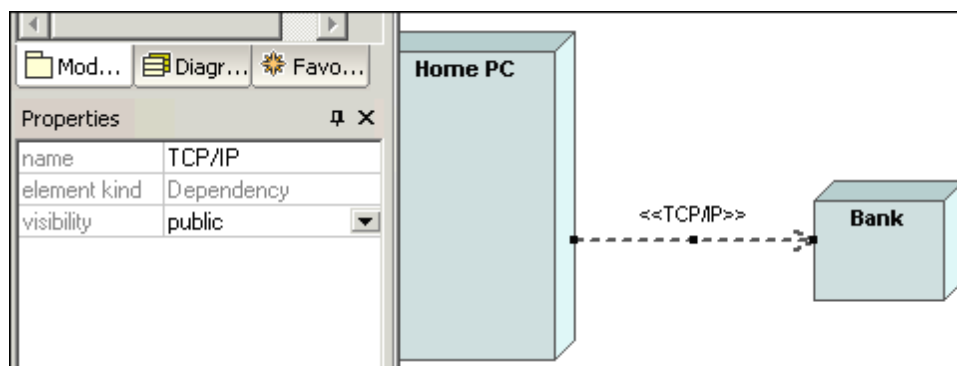
To add a Node to a Deployment diagram:

1. Click the Node icon  in the icon bar, and click right of the Home PC node to insert it.
2. Rename the node to Bank, and drag on one of its edges to enlarge it.



To create a dependency between two nodes:

1. Click the dependency icon , then drag from the **Home PC** node to the **Bank** node. This creates a dependency between the two nodes.
2. Click into the **name** field of the Properties tab, change it to **TCP/IP**, and press Enter to confirm. The dependency name appears above the dependency line.



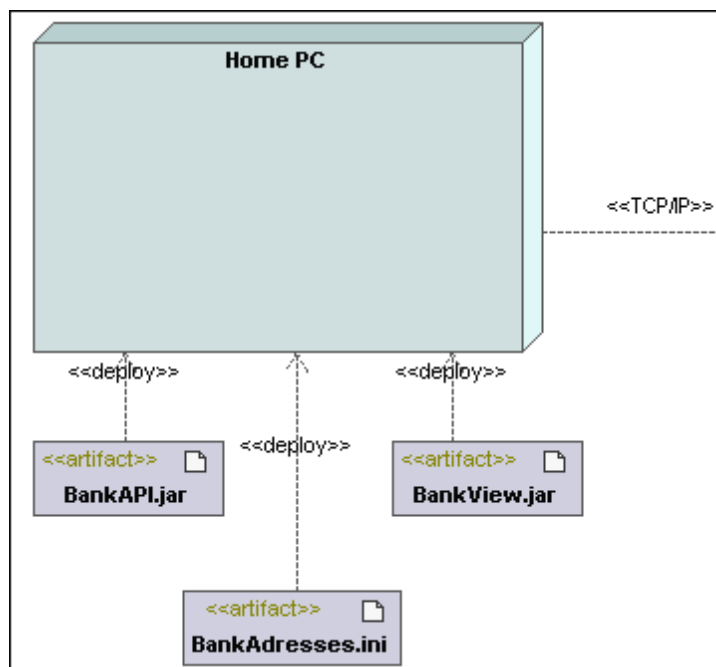
Please note:

Having clicked the dependency arrow (or any named elements), you can also enter text directly through the keyboard, without having to first click into the name field.

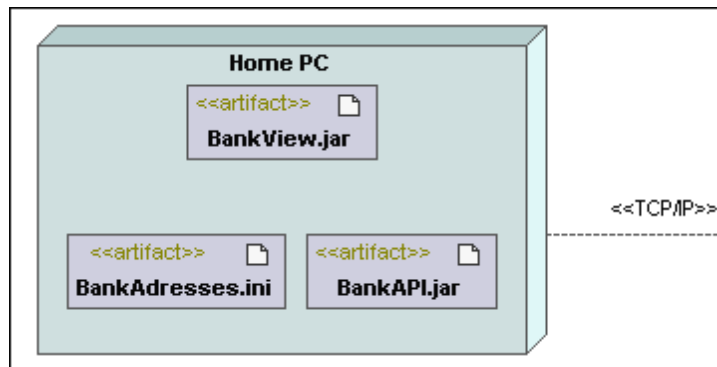
Adding artifacts to a node and creating dependencies between them:

Expand the Deployment View package, in the Model Tree, to see its contents:

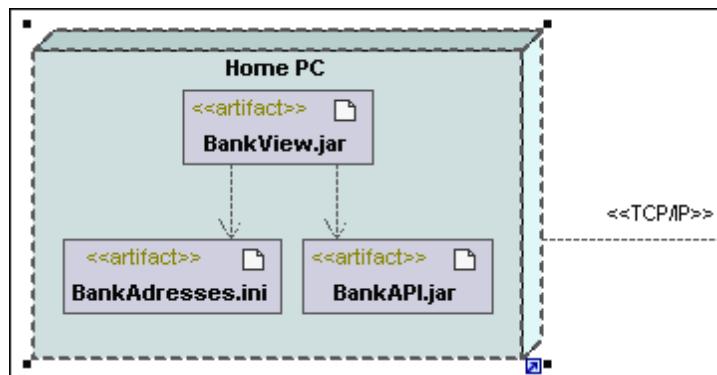
1. Click each of the **BankAddresses.ini**, **BankAPI.jar** and **BankView.jar** artifacts individually, and place them on the diagram background (Deployment dependencies are displayed for each artifact).



2. Click the **BankView.jar** artifact and drag it onto the **Home PC** node. The node is highlighted when the drop action will be successful.
3. Use the same method to drag the other artifacts onto the Home PC node. The artifacts are now part of the node and move with it when it is repositioned.



4. Click the Dependency icon  in the icon bar, and hold down the **Ctrl** key.
5. Drag from the **BankView.jar** artifact to the **BankAddresses.ini** artifact; still holding down the **Ctrl** key.
6. Drag from the **BankView.jar** artifact to the **BankAPI.jar** artifact.



Please note:

Dragging an artifact out of a node onto the diagram background, automatically creates a Deployment dependency.

To delete an artifact from a node and the project:

- Click the artifact you want to delete and press the **Del** keyboard key.
The artifact and any dependencies are deleted from the **node** as well as the **project**.

To remove an artifact from a node and its diagram:

1. Use drag and drop to place the artifact onto the diagram background.
2. Hold down the **Ctrl** key and press **Del**.

The artifact and any dependencies are deleted from the current **diagram** and not from the project.

To create operations / properties, or nesting artifacts:

1. **Right** click the artifact in the **Model Tree**.
2. Select the appropriate action from the **context** menu, e.g. **New Element | Operation**, **Property**, or **Artifact**.

The new element will appear below the selected artifact in the Model Tree.

3.7 Round-trip engineering (model - code - model)

Altova web site:  [UML code generation](#) and [UML Round-trip engineering](#)

The aim of this tutorial section is to:

- Perform a project syntax check
- Generate project code
- Add a new method external code i.e. to the SavingsAccount class
- Synchronize the UModel model with the new code

Packages and Code / model synchronization:

Code can be merged/synchronized at different levels:

- Project, Root package level (menu item)
- Package level (multiple package selection / generation is possible)
- Class level (multiple class selection / generation is possible)
- Component level

The BankView realization diagram depicts how the BankView component is realized by its six constituent classes. This is the component that is produced when the forward-engineering section of the tutorial is complete.

To be able to produce code:

- The component must be **realized** by one or more classes.
- The component must have a **physical location**, i.e. directory, assigned to it. The generated code is then placed in this directory.
- Components must be individually set to be **included** in the code engineering process.
- A Java, C#, or VB namespace root package must be defined.

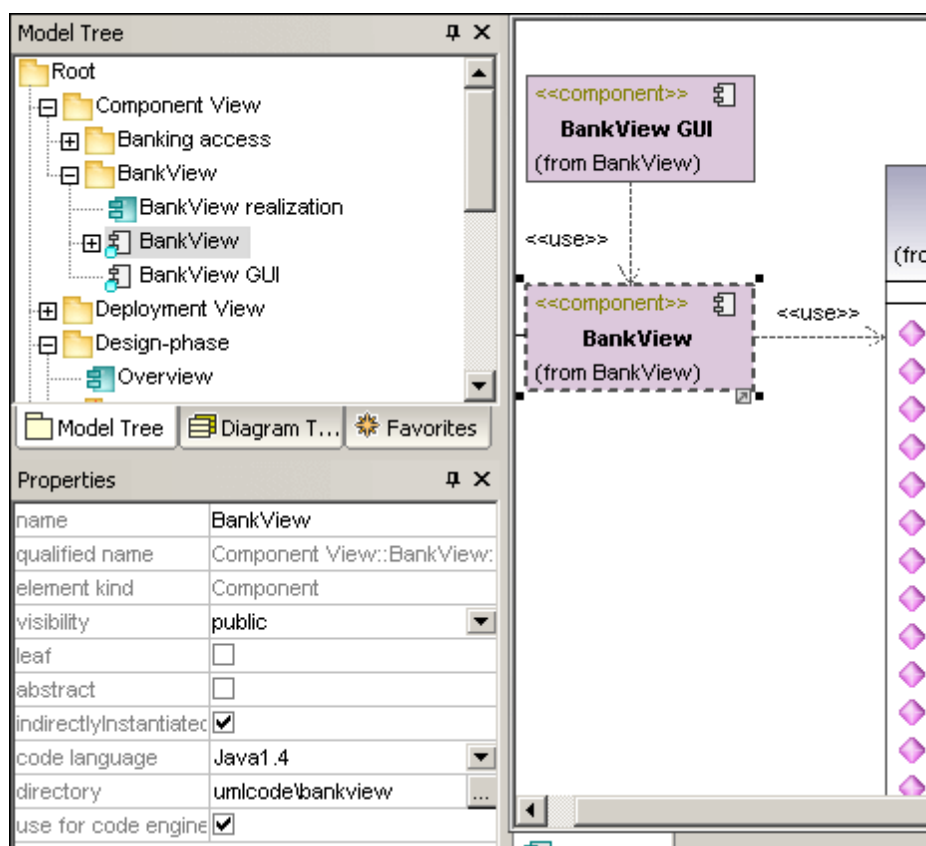
Please note:

The Java namespace root has been set on the **Design-phase | BankView | com** package in the Model Tree.

Java, C# or VB code can be combined in one project and are automatically handled during the round-trip engineering process. The Bank_MultiLanguage.ump file in the ... \UModelExamples folder is an example of a project for Java and C# types of code.

To define a code generation target directory:

1. Double click the  **Overview** icon under the **Design-phase** package to switch into the component overview.
2. Click the **BankView** component, in the diagram, and note the current settings in the Properties tab.
3. Click the browse button , to the right of the directory field.
4. Enter/select the target directory in the dialog box (the supplied example is defined as **InstallationDir\UModelExamples\Tutorial\umlcode\bankview**), or click the "Make New Folder" button to create a new folder.
The path now appears in the directory field.



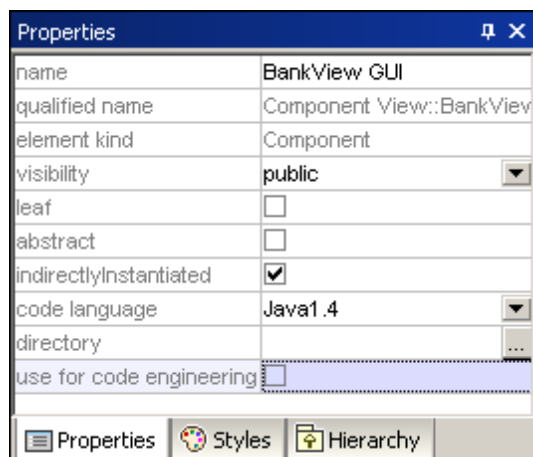
Please note:

UModel Java usually follows the convention of creating code in directories according to their namespace e.g. ...code\namespace1\C1.java.

If you want to use this directory naming convention for C# and VB .NET, select the menu option **Tools | Options | Code Engineering** and select the relevant option in the "Use namespace for code file path" group.

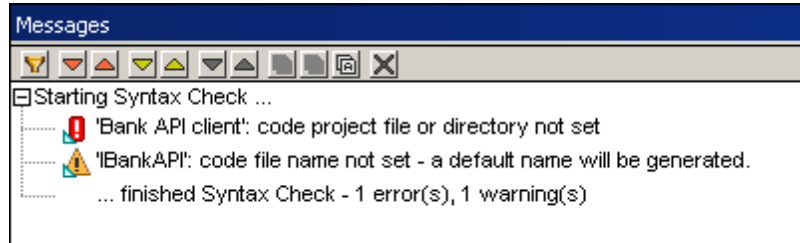
To include/exclude components from code generation:

1. Click the **BankView GUI** component.
2. **Uncheck** the "use for code engineering" check box (if not already unchecked).

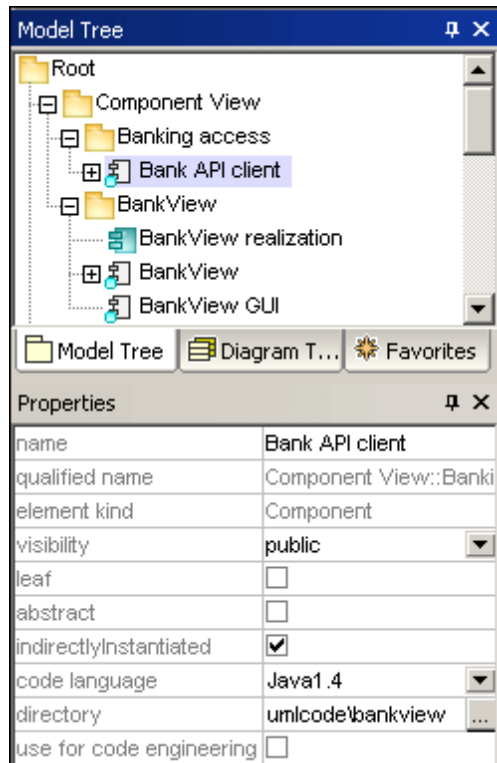


Checking project syntax prior to code generation:

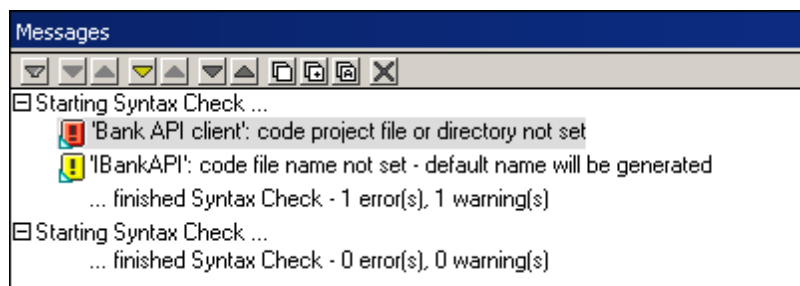
1. Select the menu option **Project | Check project syntax**.
2. A syntax check is performed, and messages appear in the Messages window, "Bank API-client: **code project file or directory not set**" - "IBankAPI: code file name not set".



3. Click the first message in the messages window.
4. The Bank API client package is highlighted in the Model Tree view, with its properties visible in the Properties tab.
5. **Uncheck** the "use for code engineering" check box for the Bank API client component.



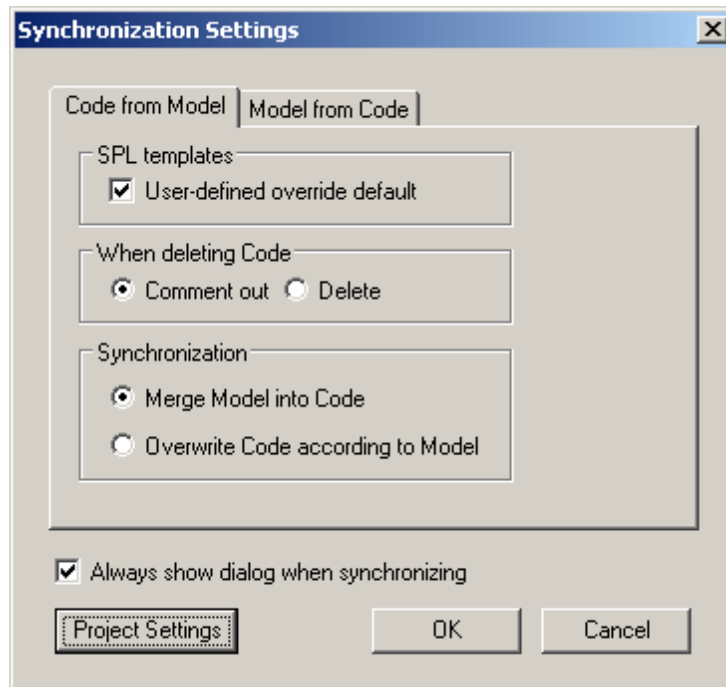
6. Check the project syntax again using **Project | Check project syntax**.



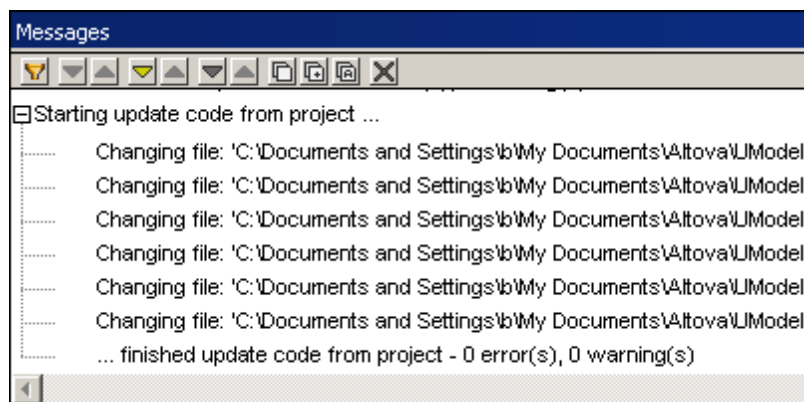
No errors are reported this time around. We can now generate program code for this project. Please see [Check Project syntax](#) for more information.

To generate project code:

1. Click the **BankView** package to select it.
2. Select the menu option **Project | Merge Program Code from UModel project**.
3. Select your synchronization options from the dialog box, and press OK to proceed (no changes needed for the tutorial; see "[Merge Program Code from UModel project](#)" for more information).



The message pane displays the outcome of the code generation process.



4. Navigate to the target directory.
Six **.Java** files have been created for the project.

Synchronizing the UModel model having updated Java code externally:

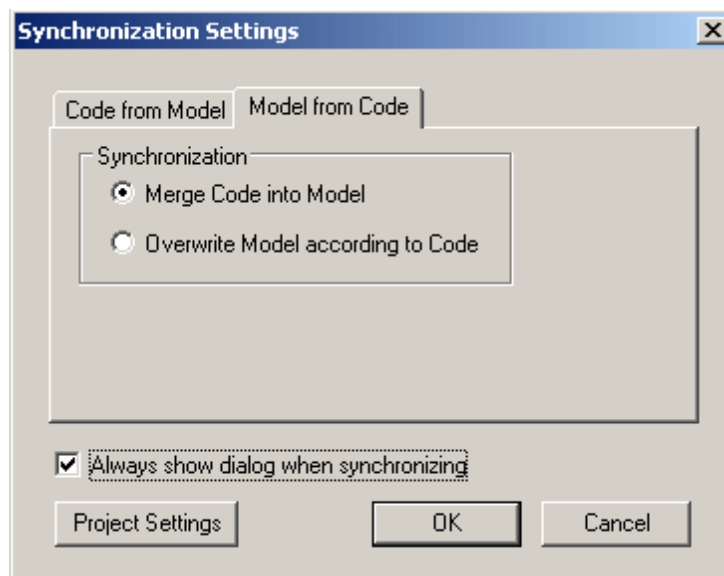
1. Open the **SavingsAccount.java** file in the text editor of your choice, XMLSpy for example.

2. Add the new **method** to the generated code "**public float getInterestRate() {}**", and save the file.

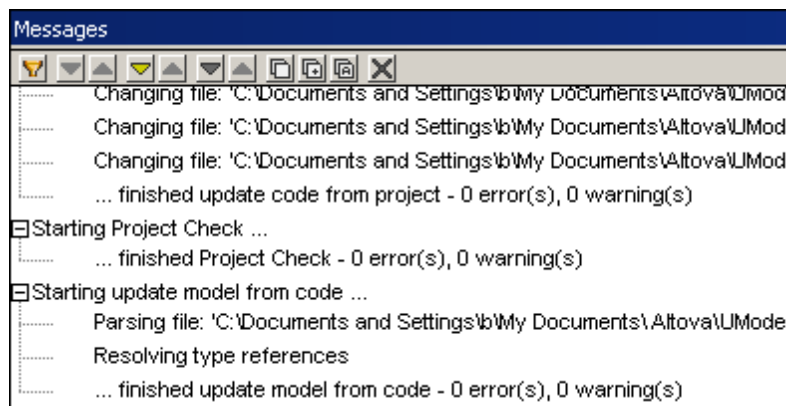
```
1
2 public class SavingsAccount extends Account
3 {
4
5     protected float interestRate;
6
7     public SavingsAccount()
8     {
9     }
10
11     public float getMinimumBalance()
12     {
13     }
14
15     public float getInterestRate()
16     {
17     }
18
19     public boolean collectAccountInfo(IBankAPI bankAPI)
20     {
21     }
22 }
23
```

3. Switch to UModel and right click the **SavingsAccount** class  under the BankView package.
4. Select the option **Code Engineering | Merge UModel Class from Program Code**.

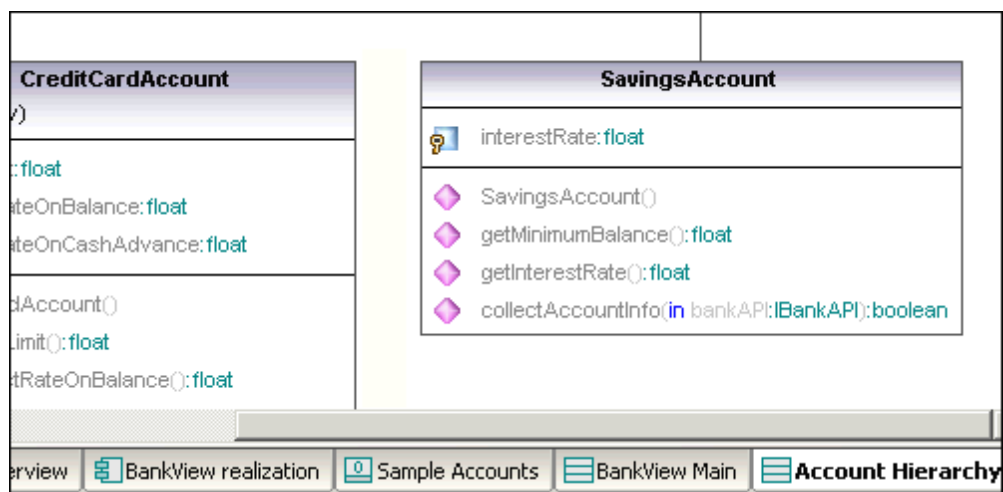
This opens the Synchronization Settings dialog box with the "Model from Code" tab being active. No changes are needed for the tutorial; see "[Merge UModel project from code](#)" for more information)



5. Click OK to merge the model from the code.



6. Click the **Account Hierarchy** tab to see the outcome of the merge process.



The new method added to the code, (getInterestRate...) generates a new **operation** in the SavingsAccount **class** of UModel.

3.8 Round-trip engineering (code - model - code)

Altova web site:  [Reverse engineering - Java, C#, VB .NET code](#)

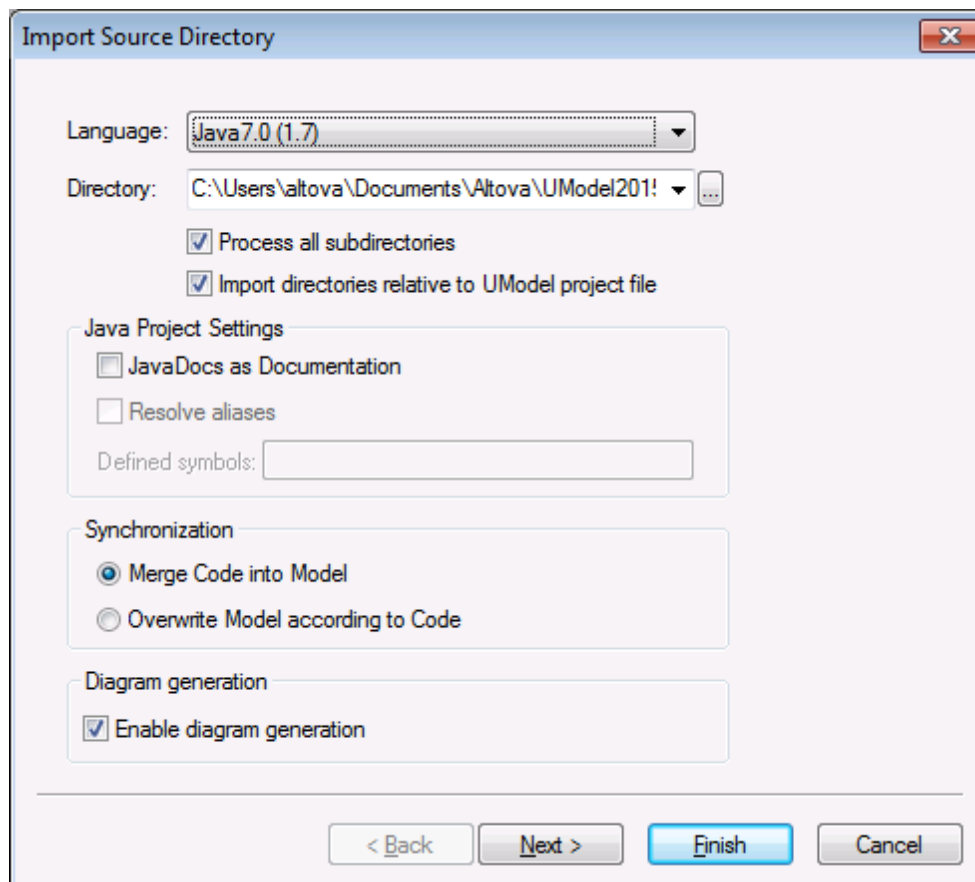
The aim of this tutorial section is to:

- Import a directory containing Java code generated by XMLSpy
- Add a new class to the project in UModel
- Merge to the program code from a UModel package

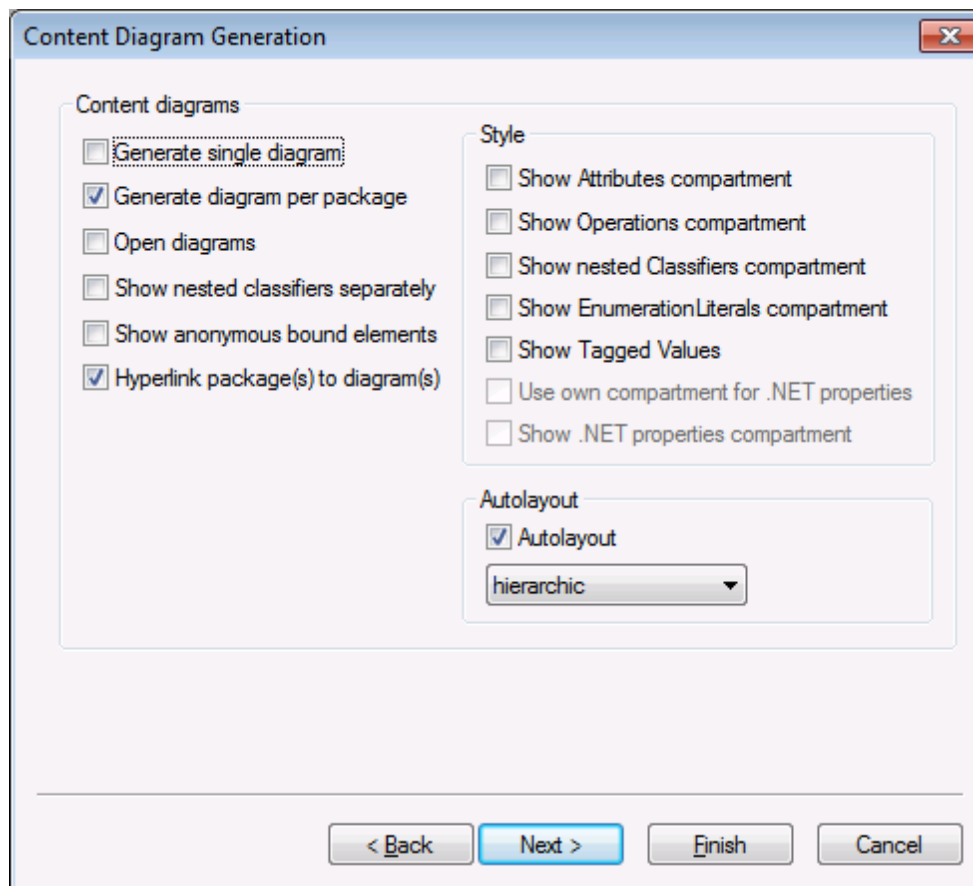
The files used in this example are available as the **OrgChart.zip** file under ...**UModelExamples** folder of your installation. Please unzip the **OrgChart.zip** file into the ...**UModelExamples** folder before you start this section. This creates the **OrgChart** directory which will then be used to import the existing code.

Importing existing code from a directory

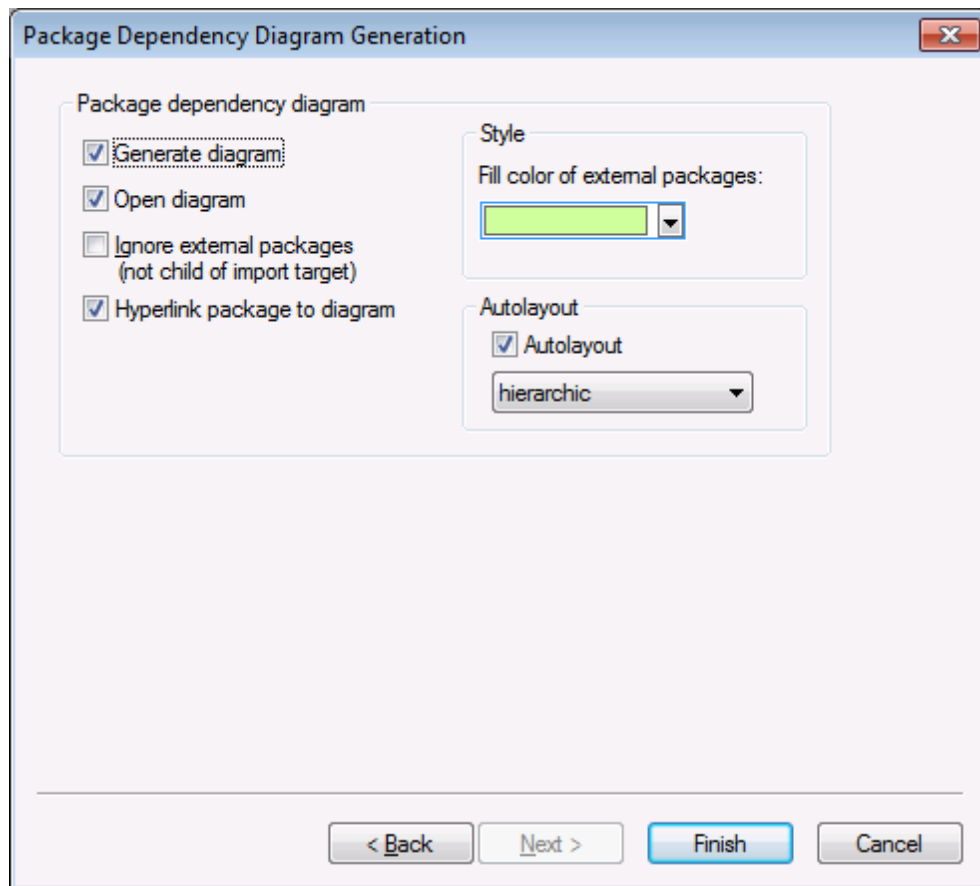
1. On the **File** menu, click **New**.
2. On the **Project** menu, click **Import Source Directory**.
3. Select the C#, Java, or VB version of the source code.
4. Click the Browse button  and select the **OrgChart** directory in the ...**UModelExamples** folder.



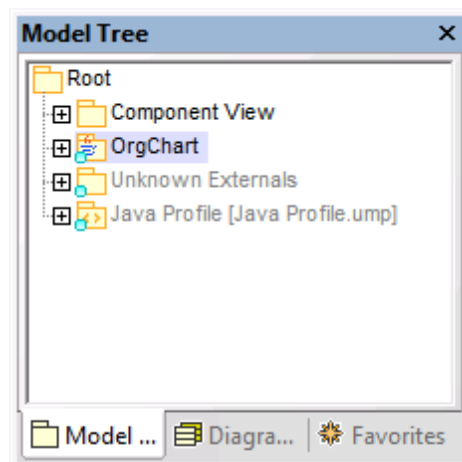
5. To instruct UModel to generate [Class Diagrams](#) and [Package Diagrams](#) from the source code, select the "**Enable diagram generation**" check box, and click Next.
6. When prompted, select the **Generate single diagram** and/or **Generate diagram per package** options. The diagram styling options can be changed later if necessary.



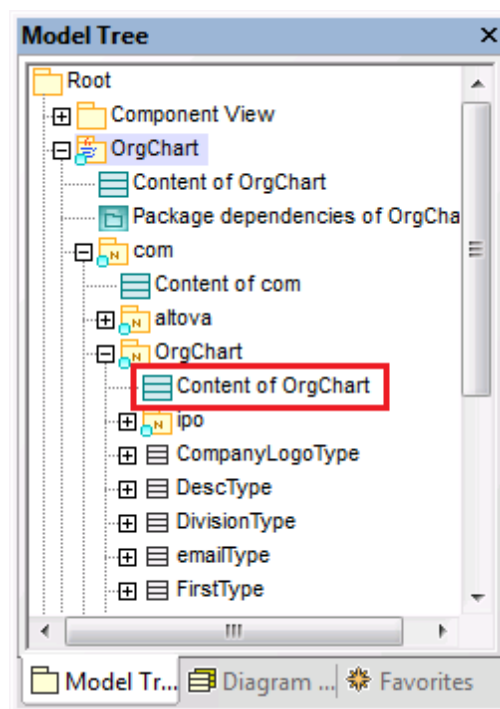
7. Click **Next** to continue. This dialog box allows you to define the package dependency generation settings.



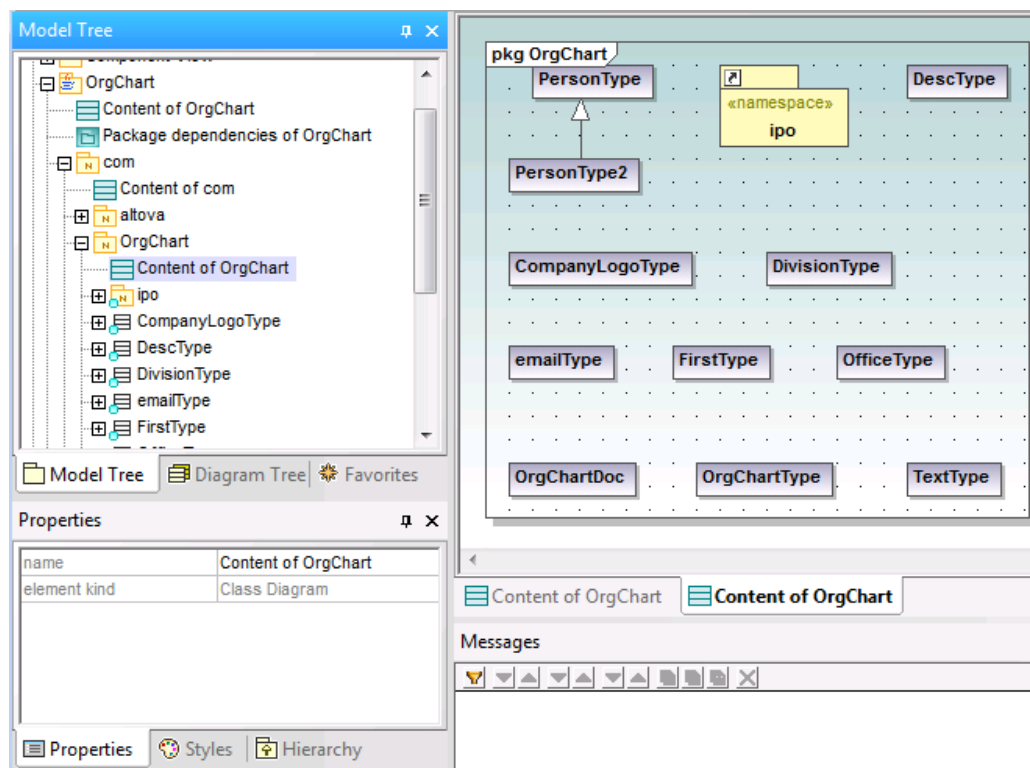
8. Click **Finish**. When prompted, save the new model to a directory on your system. The data is parsed, and a new package called "**OrgChart**" is created.



9. Expand the new package and keep expanding the sub packages until you get to the **OrgChart** package (**com | OrgChart**). Double-click the "**Content of OrgChart**" diagram icon:

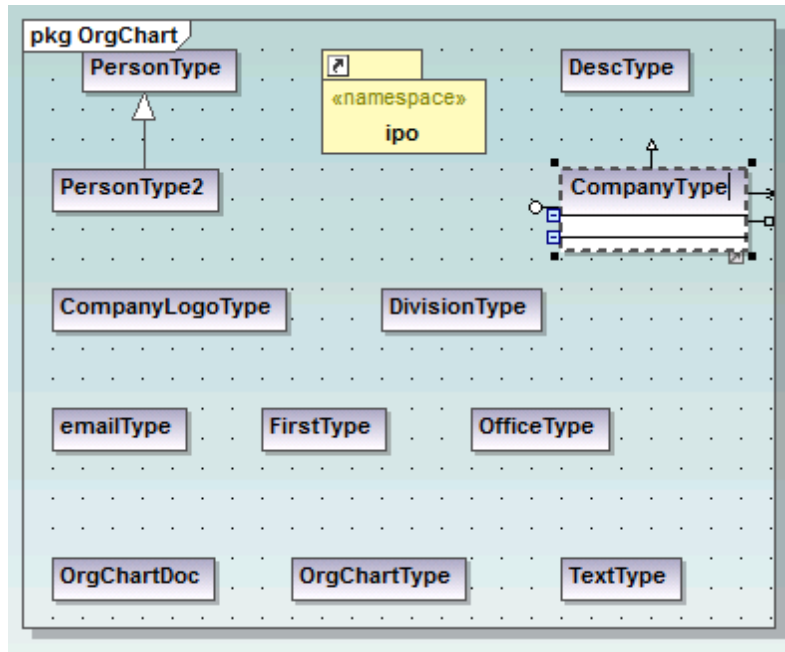


The collapsed classes that make up OrgChart are displayed in the main pane.

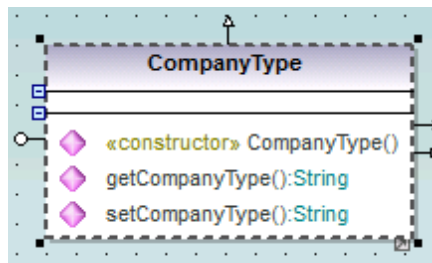


Adding a new class to the OrgChart diagram

1. Right-click inside the main pane, and then select **New | Class** from the context menu.
2. Click the header of the new class, and enter **CompanyType** as the name of the new class.

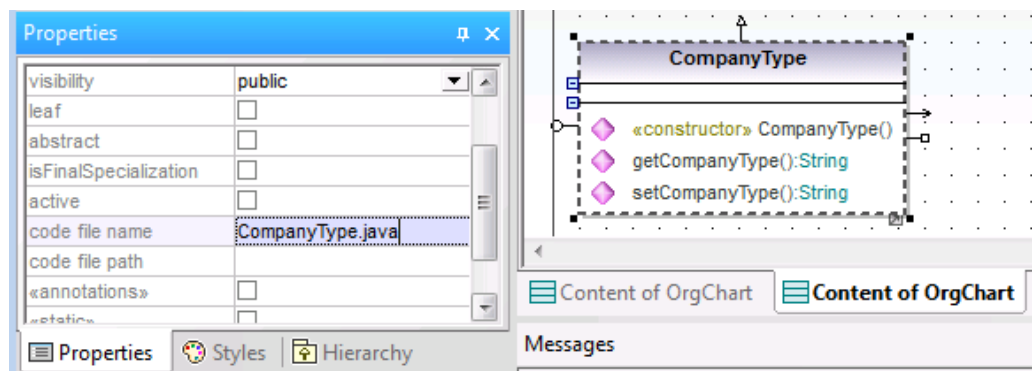


3. Add new operations to the class using the **F8** shortcut key. For the purpose of this example, add the following operations: **CompanyType()**, **getCompanyType():String**, **setCompanyType():String**.

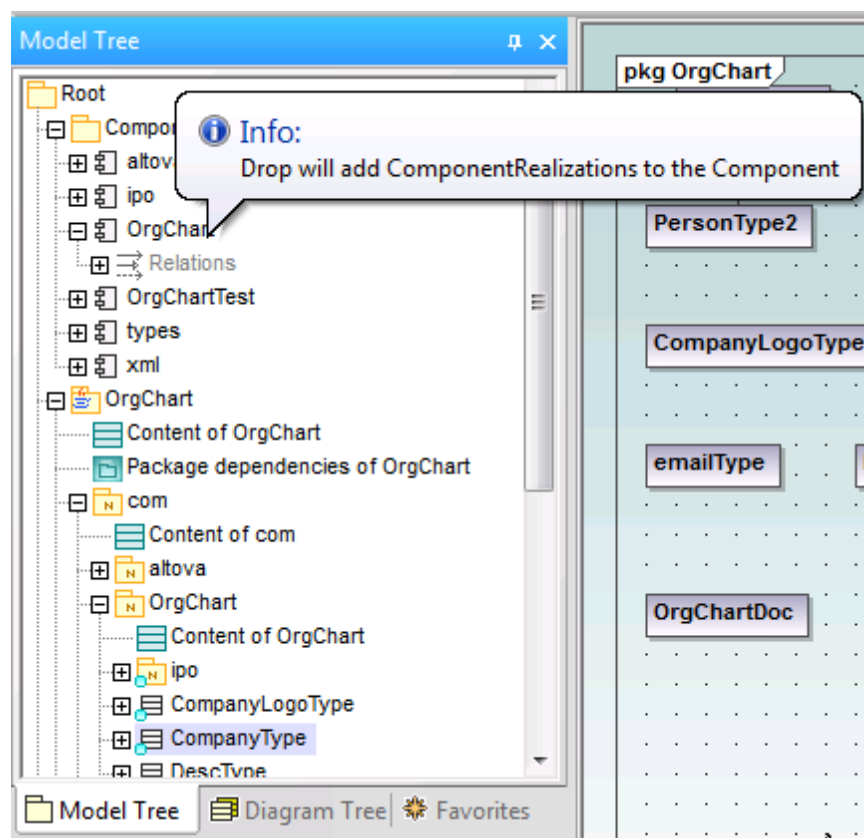


Making the new class available for code generation

1. While the **CompanyType** class is active, click into the "code file name" field and enter the Java file name of the new class **CompanyType.java**.



2. Click the new **CompanyType** class in the Model Tree, drag upwards and drop onto the **OrgChart** component below the Component View package. A notification appears when the mouse pointer is over a component.

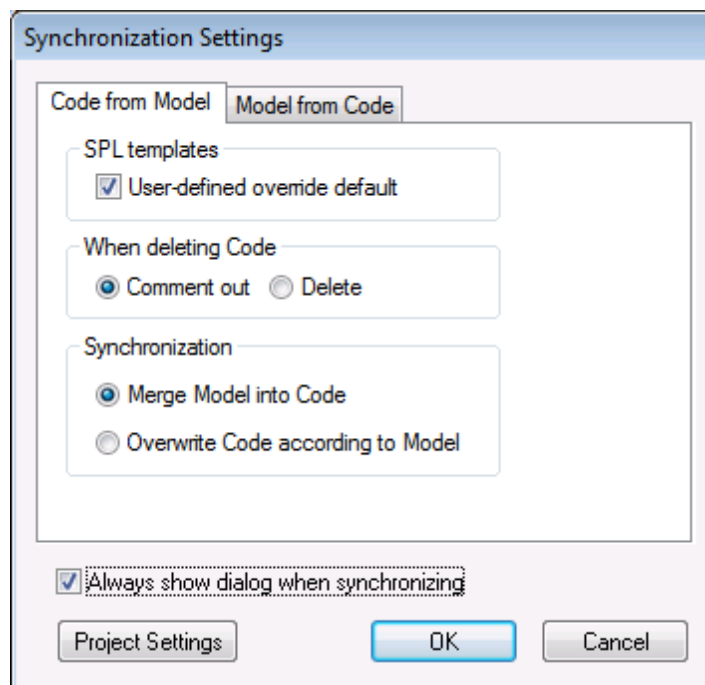


This method creates a Realization between a class and a component, so that you do not have to use component or deployment diagrams. Expand the **Relations** item below **OrgChart** to see the newly created Realization.

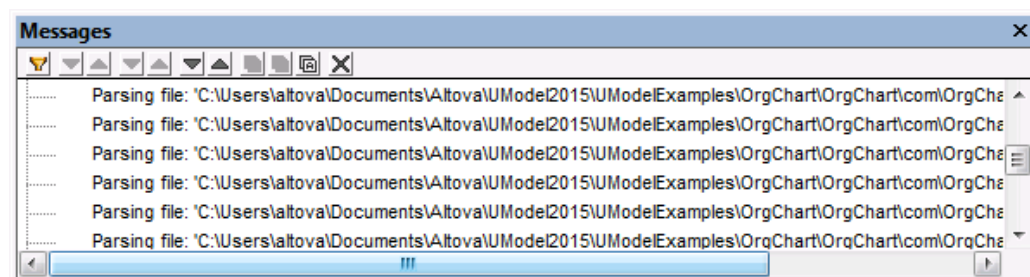


Merging program code from a package

1. Right-click the **OrgChart** package, select **Code Engineering | Merge Program code from UModel Component**, and then press Enter to confirm.



The messages window displays the syntax checks being performed and status of the synchronization process.



When the process completes, the new **CompanyType.java** class has been added to the folder **...\OrgChart\com\OrgChart**.

All method bodies and changes to the code will either be commented out or deleted

depending on the setting in the "When deleting code" group, in the Synchronization settings dialog box.

You have learned how to create a modeling project using the forward engineering process, and also completed a full round-trip code engineering cycle with UModel. The rest of this document describes how best to achieve modeling results with UModel.

Chapter 4

UModel User Interface

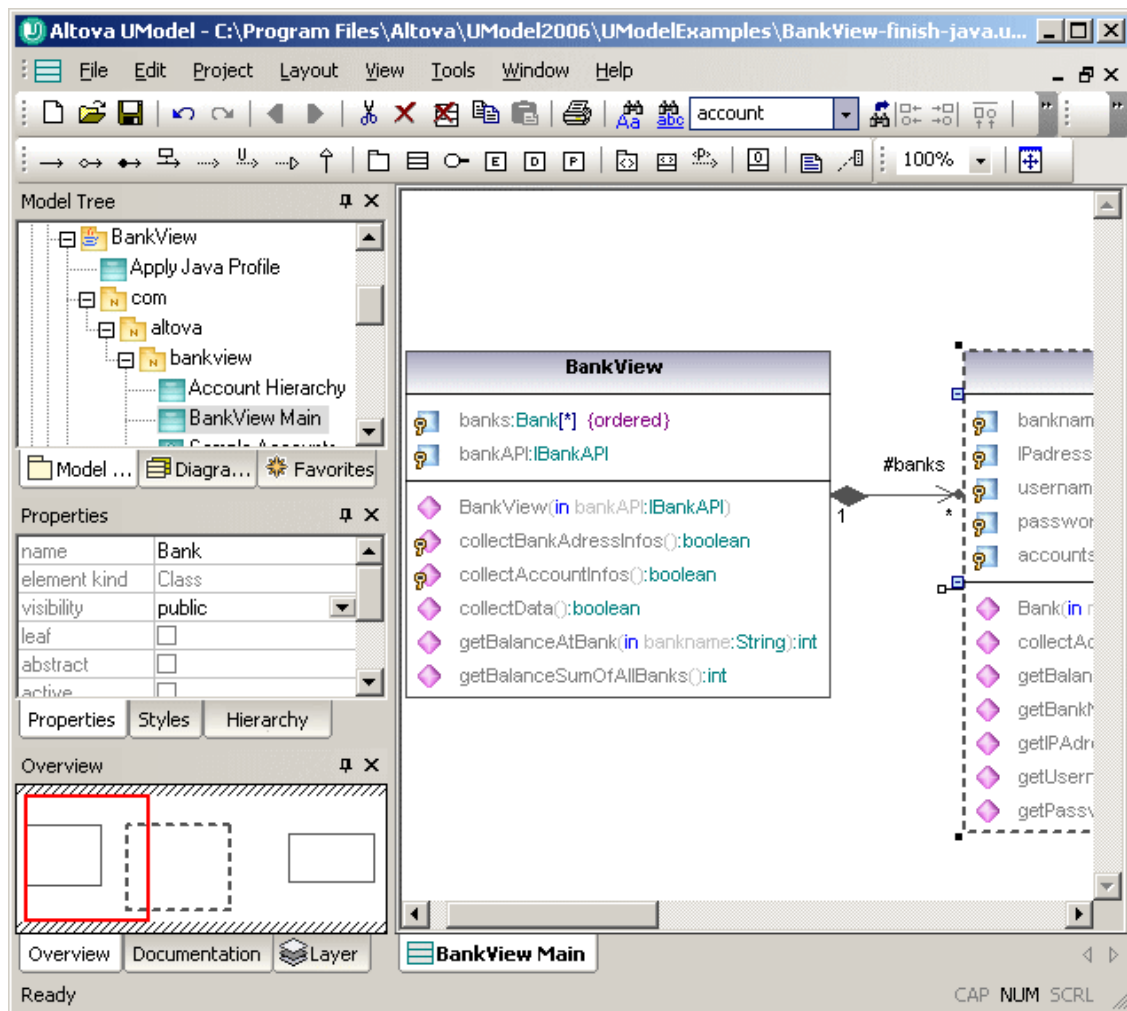
4 UModel User Interface

UModel consists of series of panes on the left and a larger diagram tab at right. The panes at left allow you to view and navigate your UModel project from differing viewpoints, and edit data directly.

The panes are Model Tree, Properties, and Overview. The working/viewing area at right is the UModel Diagram tab which currently shows the Class Diagram of the BankView Main package.

Please note:

All panes, as well as diagram tabs, can be searched using the **Find** combo box in the Main toolbar, which contains the text "**account**" in the screenshot below, or by pressing CTRL+F.



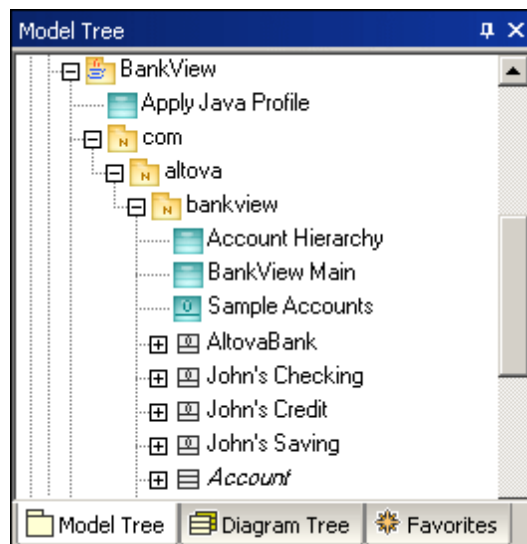
4.1 Model Tree

Model Tree tab

The Model Tree tab allows you to manipulate model items directly in the Model Tree, as well as navigate/view specific items in the Design tab. Right clicking an item opens the context menu, from which specific commands can be selected. The contents of the context menu depend on the item that you select.

Model elements in the Model Tree pane can be directly manipulated:

- Added / inserted
- Copied or moved
- Deleted
- Renamed
- Sorted according to several criteria
- Constrained



In the Model Tree tab, each folder symbol is a UML package.

Adding a new package (or any other modeling element):

1. Right click the **folder** that you want the new package/element to appear under.
2. Select **New | Package** (or respective model Element).

Copying or moving model elements:

1. Use the standard windows Cut, Copy or Paste commands or,
2. Drag model elements to different packages. Dragging an elements moves it. Holding down **Ctrl** and dragging an element creates a copy.

When dragging elements a message might appear stating that select "No sort" needs to be activated to allow you to complete the action. Please see ["Cut, copy and paste in UModel Diagrams"](#) for more information.

Sorting elements in the Model Tree (activating no sort):

1. Right click the empty background of the Model Tree tab.
 2. Select **Sort | No sort**.
- Elements can now be positioned anywhere in the Model Tree.

Please note:

The Sort pop-up menu also allows you to individually define the sort properties of **Properties** and **Operations**.

Renaming an element:

1. Double click the element **name** and edit it.
The Root and Component View packages are the only two elements that cannot be renamed.

Deleting an element:

1. Click the element you want to delete (use **Ctrl+Click** to mark multiple elements).
2. Press the **Del** key.

The modeling element is deleted from the Model Tree. This means that it is also deleted from the Diagram tab, if present there, as well as from the project. Elements can be deleted from a diagram without deleting them from the project, using **Ctrl+Del**. Please see [deleting elements](#).

To open a diagram in the Diagram tab:

1. Double click the diagram icon  of the diagram you want to view in the diagram tab.

Modeling element icon representation in the Model Tree

Package types:

-  UML Package
-  Java namespace root package
-  C# namespace root package
-  Visual Basic root package
-  XML Schema root package
-  Java, C#, or VB code package (package declarations are created when code is generated)

Diagram types:

 Activity diagram	 Profile diagram
 Class diagram	 Sequence diagram
 Communication diagram	 State Machine diagram
 Component diagram	 SysML diagrams (9 diagram types)
 Composite Structure diagram	 Timing diagram
 Deployment diagram	 Use Case diagram
 Interaction Overview diagram	 XML Schema diagram
 Object diagram	 Business Process Modeling Notation
 Package diagram	

Element types:



An element that is currently visible in the active diagram is displayed with a blue dot at its base. In this case a class element.

-  Class Instance/Object

■ Class instance slot

☰ Class

🔗 Property

◆ Operation

◆ Parameter

🧑 Actor (visible in active use case diagram)

○ Use Case

📦 Component

📁 Node

📄 Artifact

🔑 Interface

➡ Relations (/package)

{} Constraints

Opening / expanding packages in the Model Tree view:

There are two methods available to open packages in the tree view; one opens all packages and sub packages, the other opens the current package.

Click the package you want to open and:

- Press the * key to open the current package and all sub packages
- Press the + key to open the current package.

To **collapse** the packages, press the - keyboard key. To collapse all packages click the Root package and press -.

Note that you can use the standard keyboard keys, or the numeric keypad keys to achieve this.

To find modeling elements in Diagram tab(s):

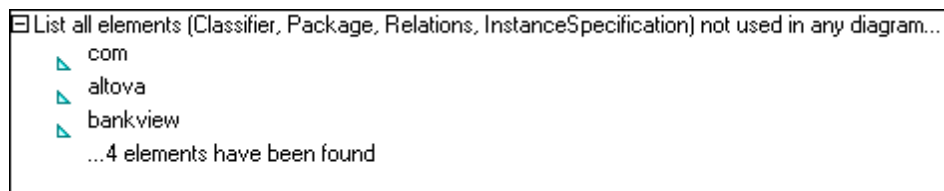
While navigating the elements in the Model Tree, you might want to see where, or if, the element is actually present in a model diagram. There are two methods to find elements:

1. Right click the element you want to see in the Model Tree tab, and select:
 - Show element in active diagram - to find it in the same type of diagram tab
 - Show element in all diagrams - if currently active diagram differs from selected model element.

To generate a list of elements not used in any diagram:

1. Right click the package you would like to inspect.
2. Select the menu option "List elements not used in any diagram."

A list of unused element appears in the Messages pane. The list in parenthesis, displays the specific elements which have been selected to appear in the unused list, please see the [View](#) tab in Reference section under, **Tools | Options** for more information.



To **locate** the missing elements in the Model Tree:

- Click the element name in the Messages pane.

Please note:

The unused elements are displayed for the **current** package and its sub packages.

Packages in the Model Tree tab:

Only the Root and Component packages are visible on startup, i.e. when no project is currently loaded.

- Packages can be **created**, or deleted at any position in the Model Tree
- Packages are the containers for all other UML modeling elements, use case diagrams etc.
- Packages/contents can be **moved**/copied to other packages in the Model Tree (as well as into valid model diagrams in the diagram tab)
- Packages and their contents can be **sorted** according to several criteria
- Packages can be placed within other packages
- Packages can be used as the **source**, or **target** elements, when generating or synchronizing code

Generating/merging code:

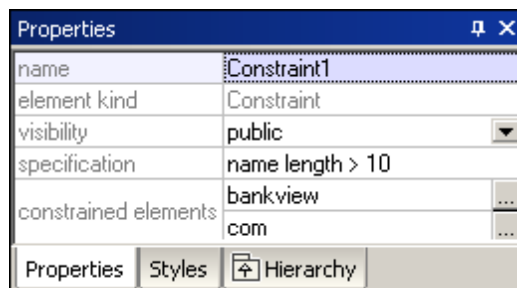
UModel allows you to generate, or merge program code directly from the Model Tree, please see: [Synchronizing Model and source code](#) for more information.

Constraining UML elements:

Constraints can be defined for most model elements in UModel. Please note that they are not checked by the syntax checker, as constraints are not part of the Java code generation process.

To constrain an element (Model Tree):

1. Right click the element you want to constrain, and select **New | Constraint**.
2. Enter the name of constraint and press Enter.
3. Click in the "specification" field of the Properties tab, and enter the constraint e.g. name length > 10.



To constrain an element in UML diagrams:

1. Double click the specific element to be able to edit it.
2. Add the constraint between curly braces e.g. interestRate:float #{interestRate >= 0}.



To assign constraints to multiple modeling elements:

1. Right click the "constrained elements" field in the Properties tab.
2. Select "Add element to constrained elements".

This opens the "Select Elements to be Constrained" dialog box.

3. Select the specific element you want to assign the current constraint to.

The "constrained element" field contains the names of the modeling elements it has been assigned to. The image above, shows that Constraint1 has been assigned to the **bankview** and **com** packages.

4.2 Diagram Tree

Diagram Tree tab

This tab displays the currently available UModel diagrams in two ways:

- Grouped by diagram type, sorted alphabetically
- As an alphabetical list of all project diagrams

Please note:

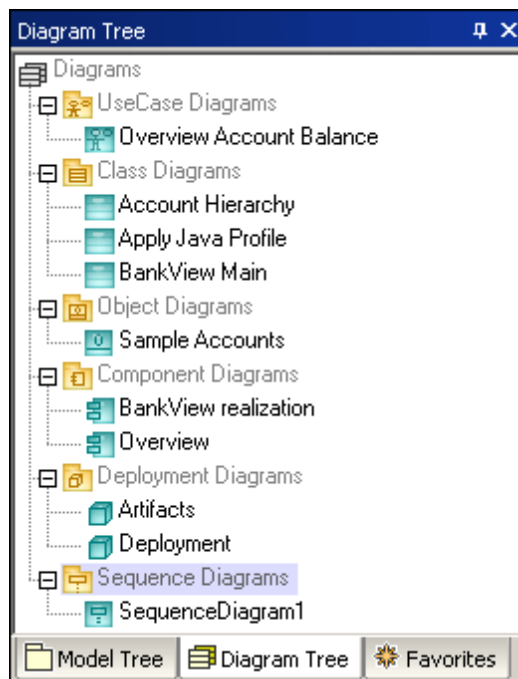
Diagrams can be added to, or deleted from, the Diagram Tree tab by right clicking and selecting the requisite command.

To open a diagram in the Diagram tab:

- Double click the diagram you want to view in the diagram tab.

To view all Diagrams within their respective model groups:

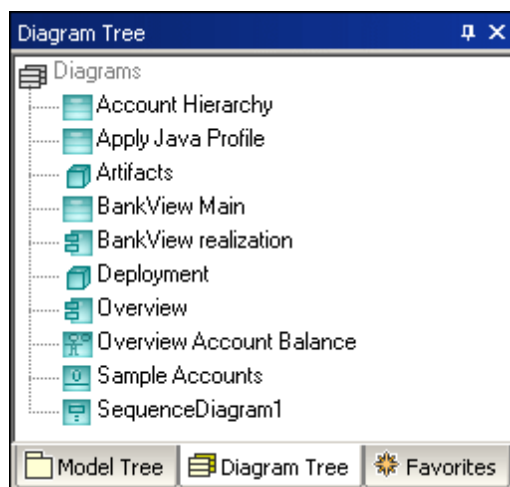
- Right click in the pane, and activate the "Group diagram by diagram type" option.



Diagrams are grouped alphabetically within their group.

To view all Diagram types in list form (alphabetically):

- Right click in the pane, and deactivate the "Group diagram by diagram type" option.



All Diagrams are shown in an alphabetically sorted list.

4.3 Favorites

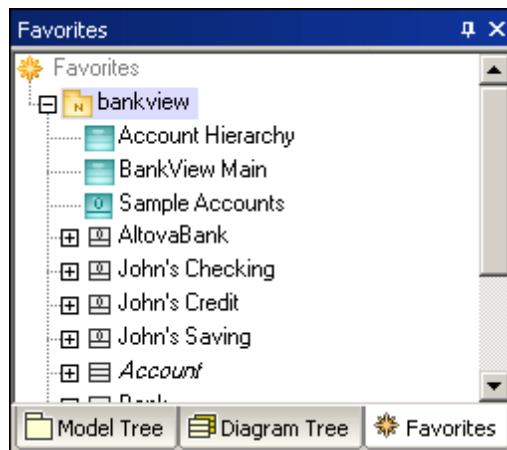
Favorites tab

Use this tab as a user-defined repository, or library, for all types of **named** UML elements i.e. classes, objects, associations etc. but not ProfileApplication or Generalization dependencies. This allows you to create your personal pick-list of modeling elements for quick access.

The contents of the Favorites tab are automatically saved with each project file. Select the menu option **Tools | Options, File** tab and click the "Load and save with project file" check box to change this setting.

To add an existing modeling element to the Favorites tab:

1. Right click an element in the Model Tree tab, or in the diagram working area.
2. Select the menu item "Add to Favorites".
3. Click the Favorites tab to see the element.



The element appears in the Favorites tab is a view of an existing element, i.e. it is not a copy or clone!

To add a NEW element to the Favorites tab:

1. Right click a previously added package, to which you want to add the element.
2. Select **New | "modeling element"** from the context menu, where "modeling element" is a class, component, or any other modeling element available in the context menu. New elements are added to the same element/package in the project, and are therefore also visible in the Model Tree tab.

To REMOVE an element from the Favorites tab:

1. Right click the same element/package that you added to Favorites.
2. Select **Remove from Favorites**.

Please note:

You can add and remove elements added to the Favorites tab, from the Favorites tab, as well as the Model Tree tab.

Deleting elements from the Favorites tab:

1. Right click the element you want to delete, and press the Del key. A message box appears, informing you that the element will be deleted from the project.
2. Click OK if you want to delete it from the project.
3. Click Cancel to retain it, and use the **Remove method** described above, to delete it from

the Favorites tab.

4.4 Properties

Properties tab

The Properties tab displays the UML properties of the currently active element.

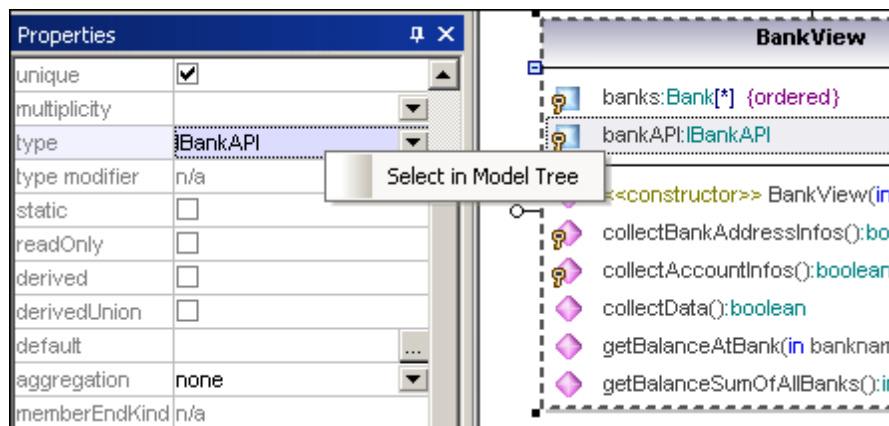


- Clicking **any** model element in any of the supplied views, or tabs, displays its properties.
- Once visible, model properties can be changed, or completed, by entering data, or selecting various options in the tab.
- Selected properties can also be located in the diagram tabs by selecting Show in Active Diagram from the context menu.

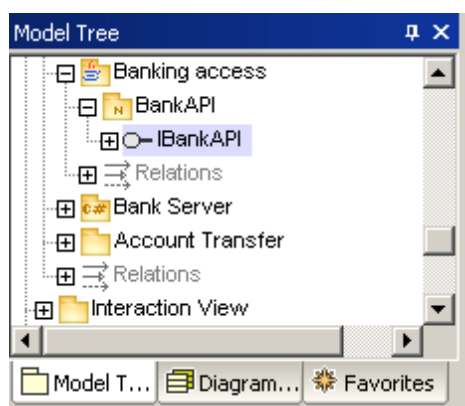
Select in Model Tree

Clicking an attribute in a class diagram displays its properties in the Properties tab. To navigate/ find it in the Model Tree:

1. Right click the **type** entry of the attribute in the Properties tab.
2. Click the "Select in Model Tree" context menu.



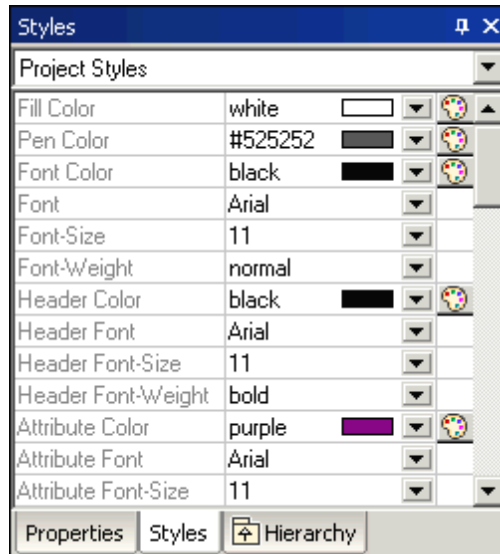
The IBankAPI interface is now visible in the Model Tree.



4.5 Styles

Styles tab

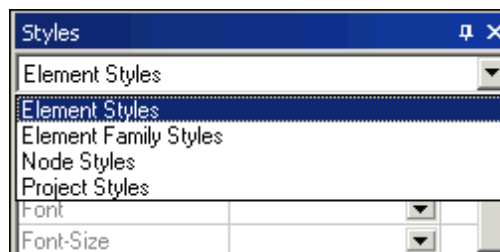
The Styles tab is used to view, or change attributes of diagrams, or elements that are displayed in the diagram view.



These style attributes fall into two general groups:

- Formatting settings; i.e. font size, weight, color etc.
- Display settings/options; show background color, grid, visibility settings etc.

The Styles tab is subdivided into several different categories/sections which can be selected by clicking the "Styles" combo box. The combo box contents depends on the currently selected model element.



Clicking an element in a diagram tab automatically selects the Element Style context, while clicking an element in the Model Tree tab selects the Project Style context.

Style **precedence** is bottom-up, i.e. changes made at the more **specific** level override the more general settings. E.g changes (to an object) made at the Element Style level override the current Element Family and Project Styles settings. However, selecting a different object and changing the Element Family Styles setting, updates all other objects except for the one just changed at the Element Style level.

Please note:

Style changes made to model elements can all be undone!

Element Styles:

Applies to the currently selected element in the currently active diagram. Multiple selections are possible.

Styles of Elements with this Stereotype:

Applies to the currently selected stereotype class in the diagram it has been defined, please see [User-defined stereotype styles](#).

Element Family Styles:

Applies to all elements of the same type i.e. of the selected Element Family. E.g. you want to have all Component elements colored in aqua. All components in the Component and Deployment diagrams are now in aqua.

Node / Line Styles:

"Node" applies to all rectangular objects.

"Lines" applies to all connectors: association, dependency, realization lines etc. for the whole project.

Project Styles:

Project Styles apply to the current UModel Project in its entirety (e.g. you want to change the default Arial font to Times New Roman for all text in all diagrams of the project).

Diagram Styles:

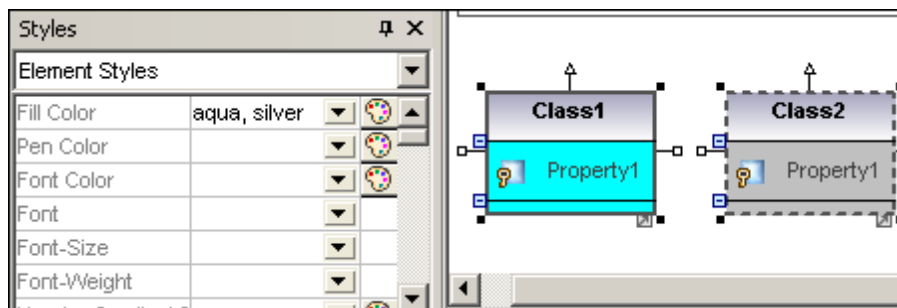
These styles only becomes available when you click/select a diagram background. Changing settings here, only affects the single UML diagram for which the settings are defined in the project.

To change settings for all diagrams of a project:

1. Click in the respective diagram,
 2. Select the **Project Styles** entry in the combo box, and scroll to the bottom of the tab.
 3. Select one of the **Diag.yyy** options e.g. Diag. Background color.
- This then changes the background color of all diagrams in the current project.

Styles display when multiple elements are selected:

If multiple elements are selected in the diagram pane, then all different style values are displayed in the respective field. In the screenshot below, Class1 and Class2 have been selected. The fill Color field displays the values for each of the elements, i.e. aqua and silver.

**Displaying cascading styles:**

If a style is overridden at a more specific level, a small red triangle appears in the respective field in the styles tab.

Placing the mouse pointer over the field displays a pop-up message which indicates the style precedence.



E.g.

The Enumeration, Package and Profile elements all have default fill color settings defined in the Element Family Styles settings. To change the fill colors at the project level, clear the value in the Element Family Styles i.e. select the empty entry in the drop-down list box, select Project styles from the Styles combo box, and change the fill color there.

4.6 Hierarchy

Hierarchy tab

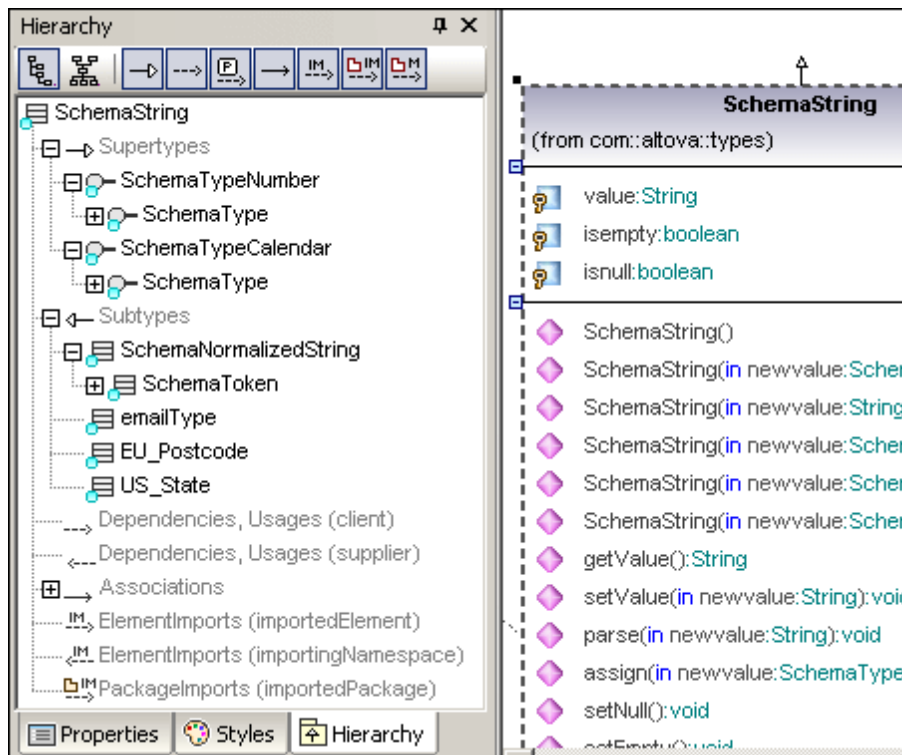
The hierarchy tab displays **all relations** of the currently selected modeling item, in two different views. The modeling element can be selected in a modeling diagram, the Model Tree, or in the Favorites tab.

Please note:

General Hierarchy tab settings are set using the menu option **Tools | Options | View**, in the Hierarchy group in the lower section of the dialog box.



Show Tree view

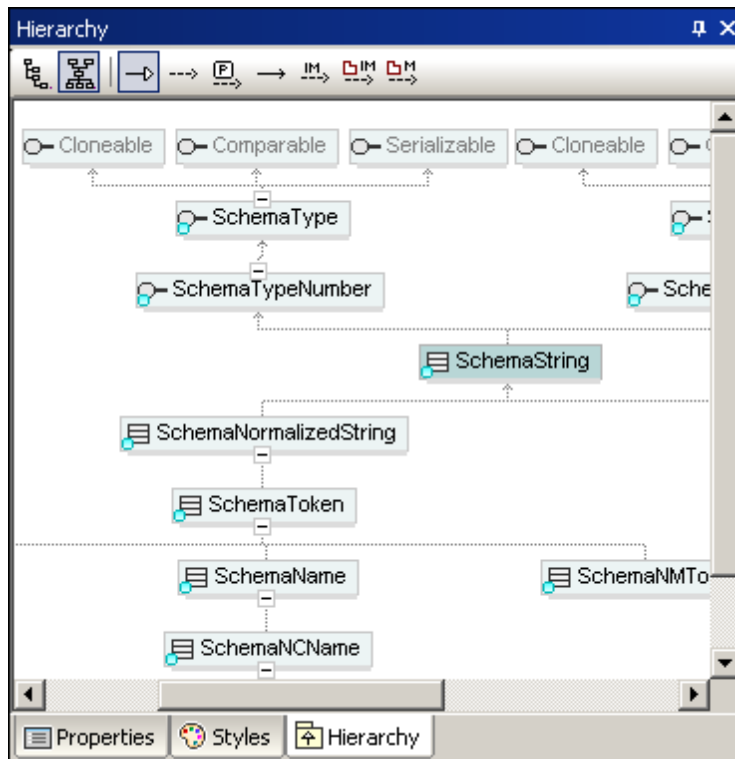


This view shows **multiple** relations of the currently selected element e.g. SchemaString. Clicking the various icons in the icon bar, allows you to show all types of relations, or narrow them down by clicking/activating the various icons. In the screenshot above, all icons are active and thus all relations are shown in a tree view.

Double clicking one of the element **icons**, in the tab, displays the relations of that element.

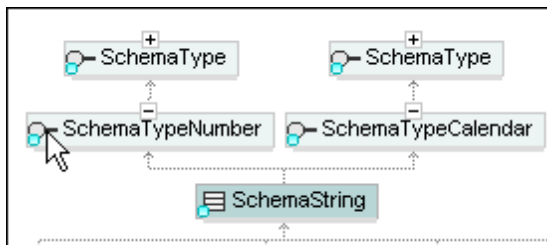


Show graph view

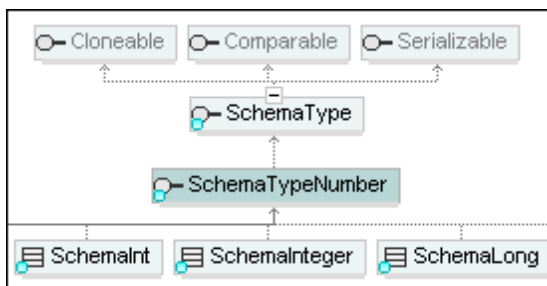


This view shows a **single** set of relations in an hierarchical overview. Only one of the relation icons can be active at any one time. The Show Generalizations icon is currently active.

Double clicking one of the element **icons** in the tab, e.g. SchemaTypeNumber, displays the relations of that element.



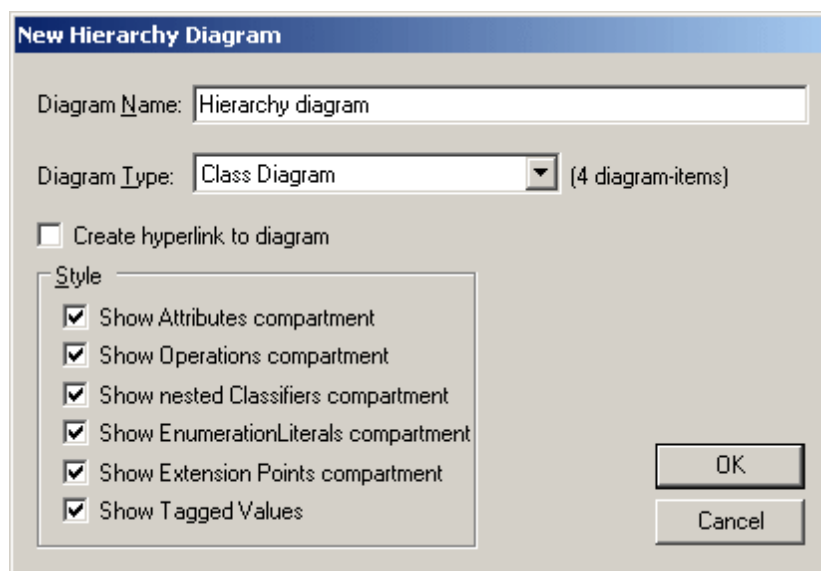
The currently selected element is now SchemaTypeNumber.



Creating a new diagram from the contents of the window:

The current contents of the graph view pane can be displayed in a new diagram.

1. Right click in the graph view pane and select **Create diagram as this graph**.

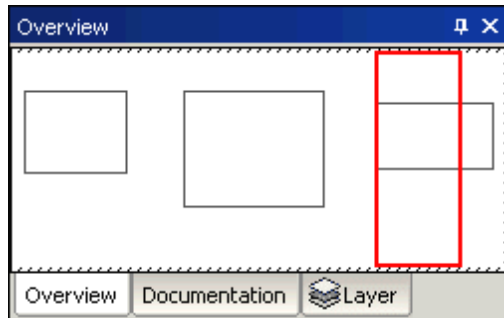


2. Edit the diagram name if necessary, select the style options and click OK. A new diagram is created.

4.7 Overview

Overview tab

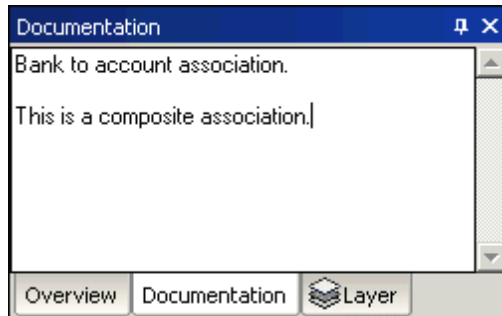
The Overview tab displays an outline view of the currently active diagram. Clicking and dragging the red rectangle, scrolls the diagram view in the diagram tab.



4.8 Documentation

Documentation tab

Allows you to document any of the UML elements available in the Model Tree tab. Click the element you want to document and enter the text in the Documentation tab. The standard editing shortcuts are supported i.e. cut, copy and paste.



Documentation and code engineering:

During code engineering, only class and interface documentation is input/output. This includes documentation defined for class/interface properties and operations.

1. Select **Project | Synchronization** settings.
2. Activate the "**Write Documentation as JavaDocs**" check box to enable documentation output.

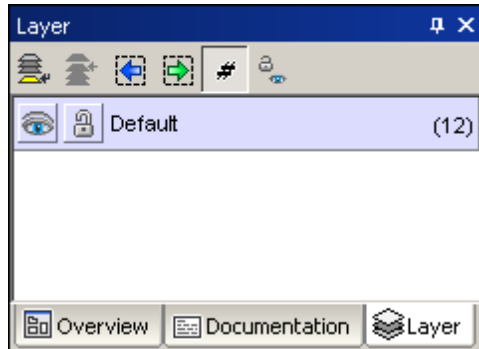
Please note:

When importing XML schemas, only the first annotation of a complex- or simpleType is displayed in the Documentation window.

4.9 Layer

Layer tab

The Layer tab allows you to define multiple layers for any UModel diagram. Single, as well as multiple, layers can be shown, locked and hidden. Layers allow you to make logical groupings of modeling elements on a diagram. You could for instance have internal info on classes, as notes, or unfinished classes on separate layers.



Please note:

Any UML elements can be assigned to any layer. You could have all associations on one layer, notes on another etc. **New** elements are always added to the **currently active** layer, which is highlighted in the Layer window.

The maximum number of layers per diagram is 20.

The following section describes all functions available through the [Layer context menu](#).

- Associations, dependencies etc. can also be created between elements on **different** layers, if those layers are currently visible.
- Multiple layers can be viewed and worked on simultaneously.
- Elements can be moved from one layer to another.
- Layers can be deleted where all elements are then merged to the layer you specify.
- When printing diagrams only those elements on the currently visible layer are printed.



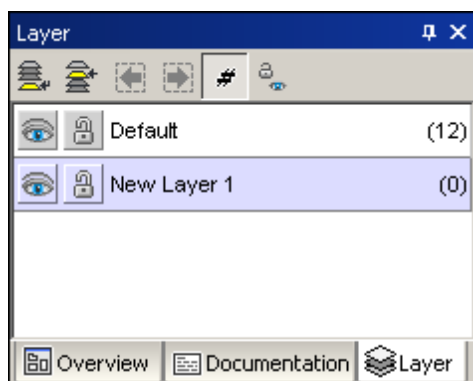
Append new layer

Appends a new layer to the current list of layers using a default name e.g. New Layer 1.



Insert new layer

Inserts a new layer above the currently active layer in the layer list. This is only possible if more than one layer exists.

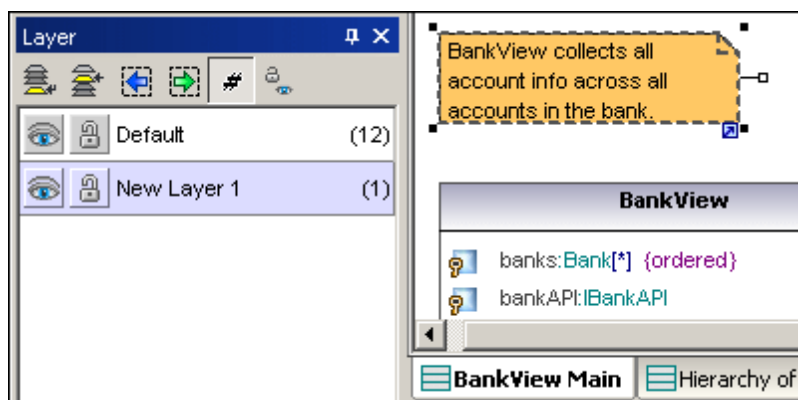


Having inserted a new layer, it then becomes the active layer. The new layer settings are:

- Visible, unlocked and (0) elements currently on this layer.
- Both the Default and New Layer are currently visible.

To add elements to a new layer:

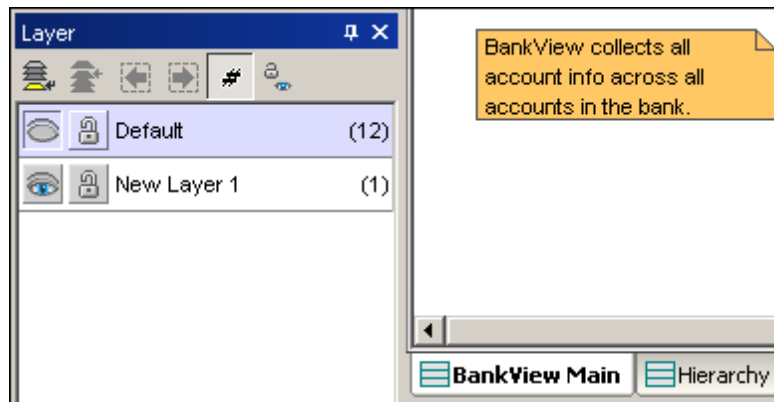
1. Click the layer you want to add the elements to in the layer list, e.g. New Layer 1.
2. Insert the new objects in the Diagram window, e.g. a note containing text describing a class.



Note that the element number on this layer has now changed to 1, and the elements of the Default layer are also still visible.

To view elements of a specific layer:

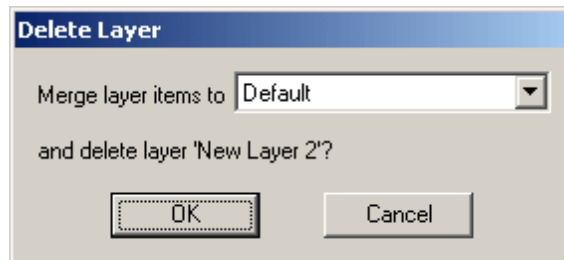
1. Click the hide/show icon  of the specific layer(s) you want to hide, so that only the layer you want to view is still active/visible.
2. E.g. Click the hide/show icon of the Default layer to only show the elements on New Layer 1.



Note that the show/hide icon changes to its hidden representation, and that only the note element is now visible in the diagram tab.

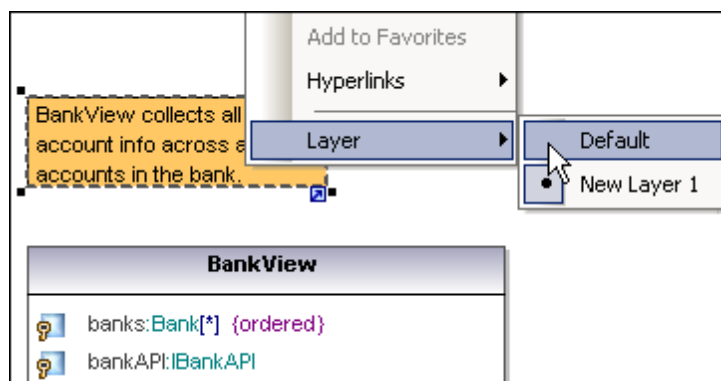
To delete a layer:

1. Right click the specific layer in the layer list and select **Delete** from the context menu. You are prompted if you want to Merge the layer items to one of the existing layers, and delete the current one.
OK merges and deletes, Cancel, cancels the action.



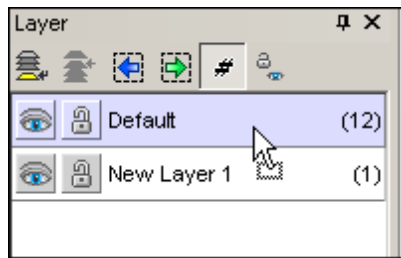
To move elements to a different layer:

1. Right click the element you want to move and select the option **Layer | "Name of Layer"** the element should be moved to.
2. E.g. right click the note element and select **Layer | Default**.



Alternatively:

1. Click the element and use drag & drop to place it onto the target layer name, e.g. Default, in the Layer list of the Layer tab.



To lock elements on a layer:

- Click the Lock icon  of the specific layer, to lock the **positions** of all the elements of the active layer.

Note that new elements can still be placed on the active layer, but cannot be selected or moved once they have been inserted.

To select elements of a specific layer:

The layer must be unlocked to be able to use this feature.

- Select the layer containing the elements you want to select in the layer list.
- Click the Previous  or Next  element icon in the Layer icon bar. Each element of the active layer is selected in sequence with every click on the icon, i.e. you cycle through all elements of the active layer.

To select all items of a layer:

- Right click the layer in the Layer window and select **Items | Select**. This selects all items of the **current layer**.

To select the items of all layers except the currently active layer:

- Right click the layer in the Layer window and select **Items | Select.Others**. This selects **all items** of all currently visible layers, **except** for those on the currently active layer!

To toggle the element counter of the layer list:

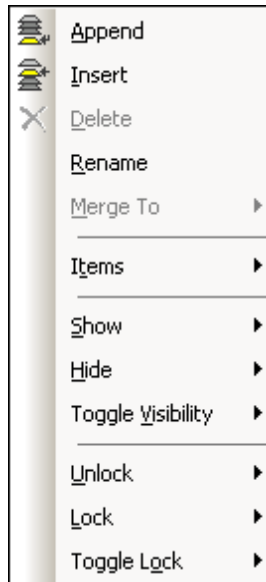
- Click the  icon of the icon bar.

To reset all layer states:

- Click the  icon of the icon bar, to set all layers to visible and unlocked.

4.9.1 Layer Context menu

Right click in the layer list window to open the context menu.



Append

Appends a new layer to the current layer list, and assigns a default name which you can change immediately or through the context menu option "Rename".

Insert

Inserts a new layer above the currently active layer in the layer list.

Delete

Deletes the currently active layer and moves the elements to an existing layer you select in the dialog box.

Rename

Allows you to rename the currently active layer in the layer list.

Merge to

Moves **all** the elements of the currently active layer to the layer you select in the flyout menu. The flyout menu contains a list of the currently available layers.

The following commands affect only those elements which are **currently visible** in the diagram tab, i.e. which are present on those layers which are currently set as active/visible.

The available sub-options are:

Selected Layer	affects only the current/active layer & elements on that layer
Unselected - Select Others	affects all other layers & elements on those layers

Items

Selected Layer, selects/marks all elements of the selected layer.

Unselected, selects/marks all elements of the unselected layer(s).

Show

Selected Layer, shows all elements of the selected layer.
Unselected, shows all elements of the unselected layer(s).

Hide

Selected Layer, hides all elements of the selected layer.
Unselected, hides all elements of the unselected layer(s).

Toggle Visibility

Selected Layer, toggles visibility of all elements of the selected layer.
Unselected, toggles visibility of all elements of the unselected layer(s).

Unlock

Selected Layer, unlocks all elements of the selected layer.
Unselected, unlocks all elements of the unselected layer(s).

Lock

Selected Layer, locks all elements of the selected layer.
Unselected, locks all elements of the unselected layer(s).

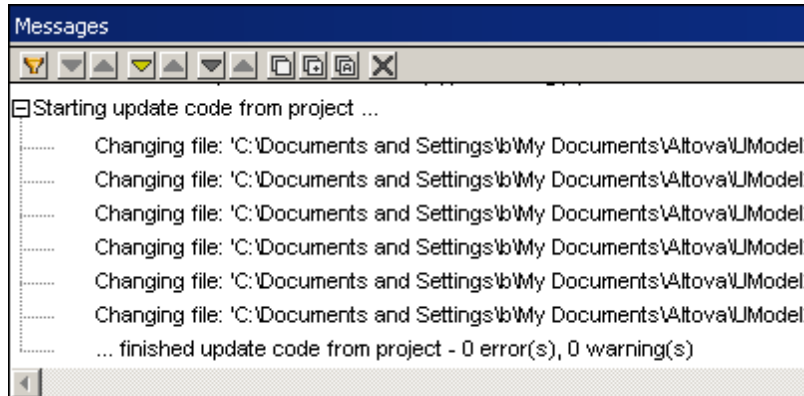
Toggle lock

Selected Layer, toggles lock of all elements of the selected layer.
Unselected, toggles lock of all elements of the unselected layer(s).

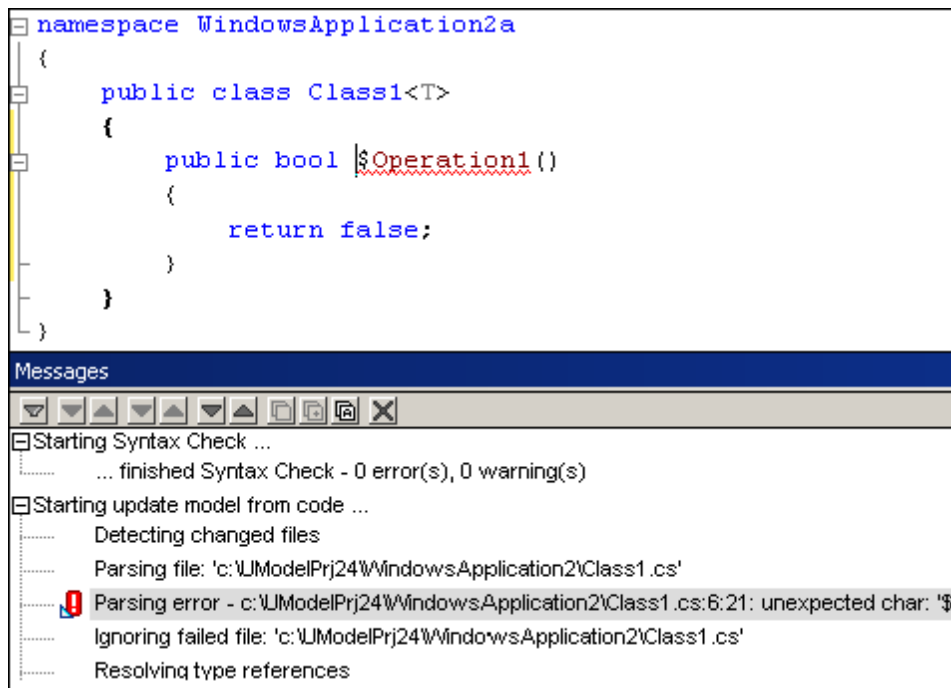
4.10 Messages

The Messages window displays warnings, hints and error messages when merging code, or checking the project syntax.

Clicking the error message highlights the corresponding element in the Model Tree and also in the owning diagram, if that diagram is currently active/open.

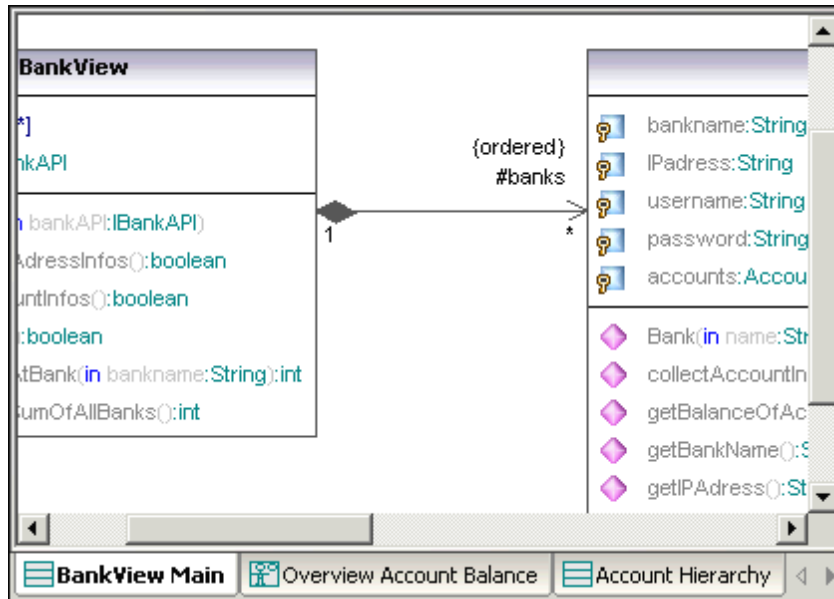


Clicking a parsing error in the Messages window of the Visual Studio or Eclipse Editions, opens the corresponding source code file with the cursor positioned in the line containing the error.



4.11 Diagram pane

The diagram pane displays all the currently opened UModel diagrams as individual tabs.



To create a new diagram:

1. Click a package in the Model Tree tab.
2. Select **New | YYY Diagram**.

To create a new diagram containing contents of an existing package:

1. Right click a package and select **Show in new Diagram | Content**.

To open / access a diagram:

- Double click the diagram **icon** in any of the Model Tree pane tabs (to open).
- Clicking any of the tabs in the Diagrams pane (to access).

To close all but the active diagram:

- Right click the diagram tab that is to remain open, select the option **Close All but active**.

Deleting a diagram:

- Click the diagram icon in the Model Tree and press Del. key.

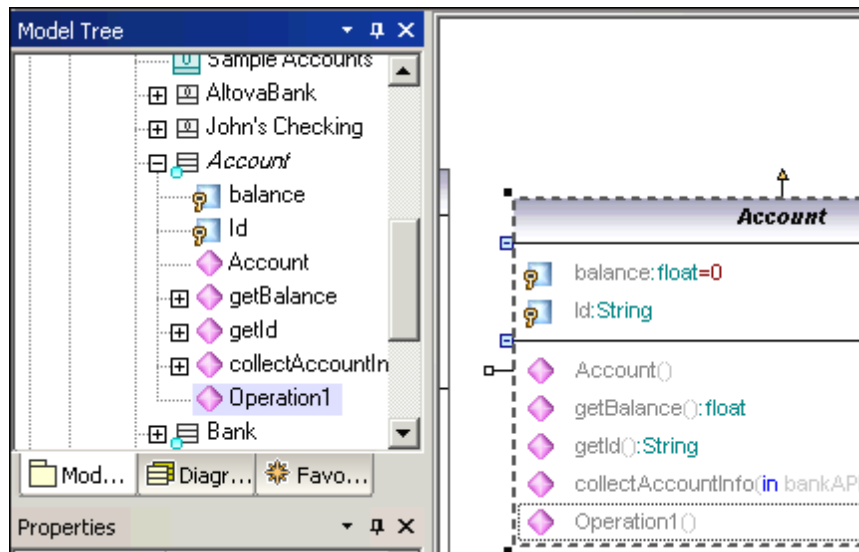
Moving diagrams in a project:

- Drag the diagram icon to any other package in the Model Tree Tab.
You might have to enable the "no sort" option to move it.

Finding (deleting) class properties and options from the Model Tree:

Properties and options can also be deleted directly from the Model Tree. To do this safely, it is important to first find the correct property. Assuming you have inserted "Operation1" in the Account class (press F8, then Enter to insert):

1. Right click Operation1 in the Account class.
2. Select the option **"Select in Model Tree"** or press F4.
The Operation1 item is now highlighted under *Account* in the Model Tree tab.



3. Press the **Del** key to delete the operation from the **class** and **project**!
Note that almost any modeling element can be found in the Model Tree when pressing F4.

Please note:

It is also possible to navigate from the Properties pane to the Model Tree when viewing an attributes properties, please see: the [Properties](#) in the User Interface section.

Deleting elements from a diagram:

Delete element from the **diagram** and **project**!

- Select the element you want to delete and press the Del. keyboard key.

Delete element **from diagram only** - not from the project!

1. Select the element you want to "delete"
2. Hold down the **Ctrl** key and press **Del**.

An [auto-layout](#) function allows you to define how you would like your diagram to be visually structured. Right click the diagram background and select either:

- Autolayout All | Force directed, or
- Autolayout All | Hierarchic
- Autolayout All | Block

Showing relationships between modeling elements:

1. Right click the specific element and select **Show**.
The pop-up menu shown below is context specific, meaning that only those options are available that are relevant to the specific element.

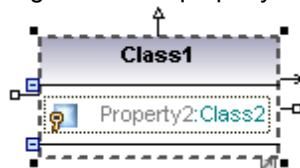
Generalizations (general)
Generalizations (specific)
Generalization Hierarchy (general)
Generalization Hierarchy (specific)
Full Generalization Hierarchy (general and specific)
InterfaceRealizations (contracts)
InterfaceRealizations (implementingClassifier)
Dependencies, Usages, Abstractions (client)
Dependencies, Usages, Abstractions (supplier)
ProfileApplications (appliedProfile)
ProfileApplications (applyingPackage)
Associations
All Properties as Associations
"Property1" as Association
ElementImports (importingNamespace)
ElementImports (importedElement)
PackageImports (importingNamespace)
PackageImports (importedPackage)
PackageMerges (receivingPackage)
PackageMerges (mergedPackage)
Templates
Bound elements
TypedElements

To show / hide text labels:

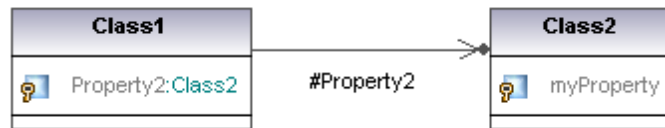
- Right click a class or association arrow and select **Text labels | Show (Hide) all Text labels**.

To show a class attribute/property as an association:

- Right click the property in the class.



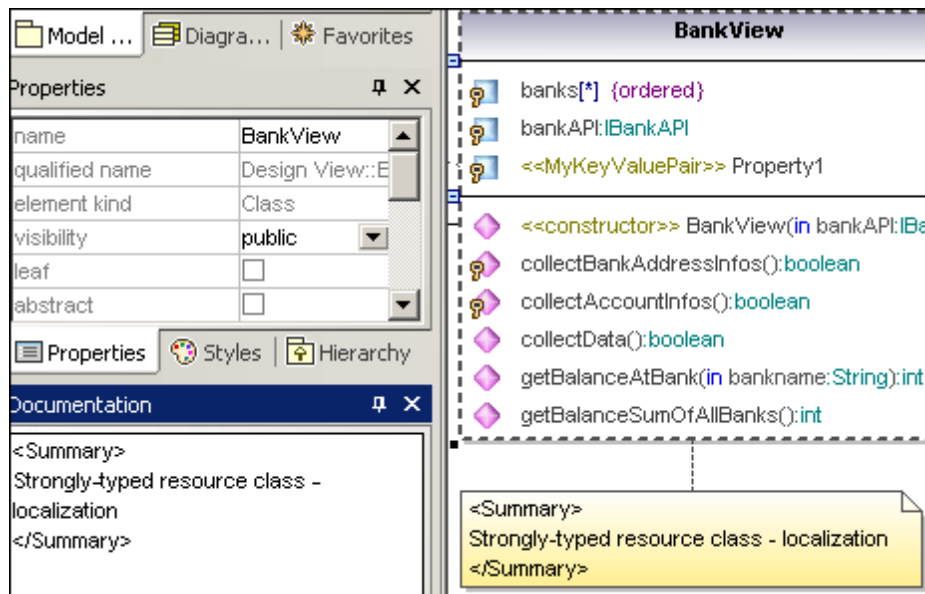
- Select the menu option **Show | "PropertyXX" as Association**. This inserts/opens the referenced class and shows the relevant association.



To show source code comments in the model diagram:

- Right click the class in the diagram pane and select **Show | Annotating comments**.

This allows you to directly edit source code DocComments, also visible in the Documentation window, in the modeling diagram.



To adjust the line thickness (weight) in a project:

- Click in any diagram, then click the Styles tab.
Making sure that the Project Styles entry in the combo box is active
- Scroll down to Line thickness and select the value that you want.
This affects the thickness of all association, aggregation, generalization etc. lines in all diagrams of the current UModel project.

4.11.1 Diagram properties

Configuring diagram properties

Click on the diagram background and then select one of the styles from the Styles combo box. Please see [Styles pane](#) for more information.

To enlarge the Diagram size:

The size of the diagram tab is defined by the elements and their placement.

- Drag an element to one of the diagram tab edges to automatically scroll the diagram tab and enlarge it.

Positioning modeling elements - the grid

Modeling elements can be positioned manually, or made to snap to a visible/invisible grid in a diagram.



toggles between showing / hiding the grid

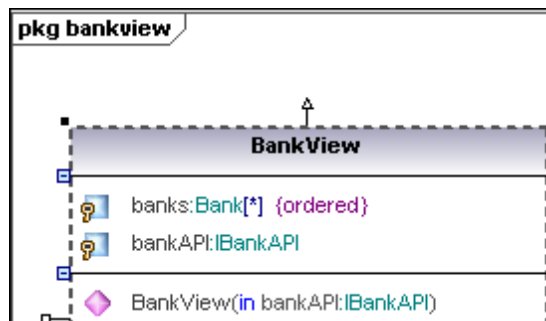


toggles between snapping elements to the visible / invisible grid

Displaying the UML diagram heading



toggles between displaying the UML diagram heading, i.e. the frame around a diagram with its name tag in the top left corner.



Aligning modeling elements

Modeling elements can be aligned, and resized, in relation to other elements e.g. all centered, left or right aligned etc.

Please note:

When marking several objects, the action that you apply uses the element that was selected **last** as the template you apply. E.g. if you mark three class elements and click the "Make same Width" icon, then all three will be made as wide as the last class you selected.

This also applies when marking several objects using the marquee, the element last selected by the marquee becomes the template element whose properties are applied to the other marked objects.



Icon functions for each icon group:

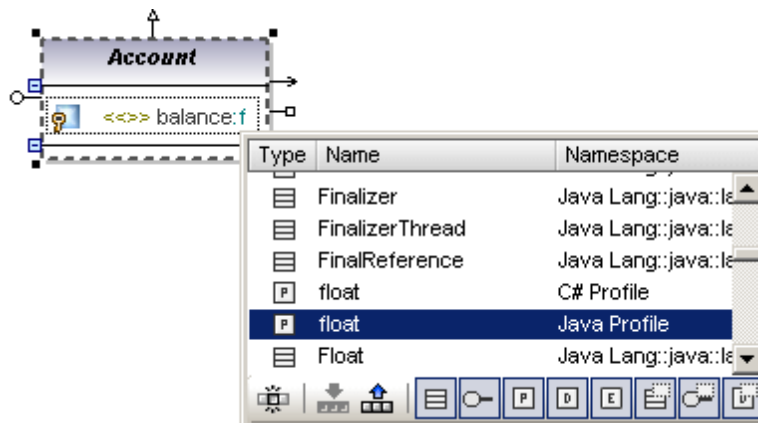
Align Left, align Right

Align Top, align Bottom
 Center vertically, center horizontally
 Space Across, Space down
 Line up horizontally, line up vertically.
 Make same width, make same height, make same size.

Enhanced autocompletion window - selecting datatypes

When inputting datatypes for operations or properties, an autocomplete window is automatically opened. The autocomplete window has the following features:

- Clicking a column name sorts the window by that attribute in ascending/descending order.
- The window can be resized by dragging the bottom right corner.
- The window contents can be filtered by clicking the respective category icons at the bottom of the window.
- The **Single mode** icon  lets you switch between enabling only one of the category icons at any one time (single mode), or enabling multiple category icons (multi mode). The image below shows the autocomplete window in "multi-mode", i.e. all category icons are enabled. The single mode icon is not enabled.
- The Set all / Clear all Categories icons, set or clear all categories icons.
- When autocompletion is disabled, press **Ctrl+Space** to display the autocomplete window on demand.



Filter categories:

Class, Interface, PrimitiveType, DataType, Enumeration, Class Template, Interface Template, DataType Template.

Please note:

To enable/disable the autocompletion select **Tools | Options | Diagram Editing** tab, then click the "Enable automatic entry helper" option. The on-demand availability is not affected by this setting however.

4.11.2 Cut, copy and paste in UModel Diagrams

Cut, Copy and Paste of diagram elements within the Diagram pane

All UModel diagram elements can be cut, copied and pasted within, across the same type, and even into other types of diagram tab. Mouse or keyboard shortcuts can be used to achieve this in two different ways:

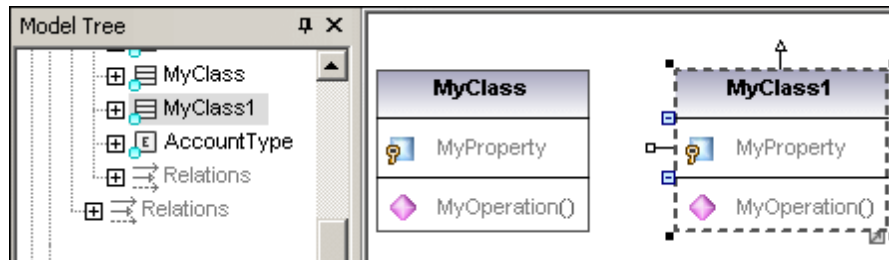
Having copied an element:

- **"Paste"**, using the keyboard shortcut CTRL+V, or "Paste" from the context menu, as well as Paste from the Edit menu, always adds a **new** modeling element to the diagram and to the Model Tree.
- **"Paste in diagram only"**, using the context menu, i.e. right clicking on the diagram background, only adds a "link/view" of the existing element, to the current diagram and not to the Model Tree.

Using the Class diagram as an example:

Paste (Ctrl+V) of a copied **class**:

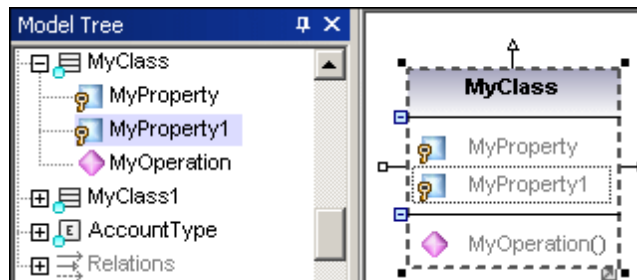
- **Pasting** a copied class in the same diagram (or package), inserts a **new class** with the source class name plus a sequential number. E.g source class name is **myClass**, pasted class name is **myClass1**. All operations and properties are also copied to the new class.



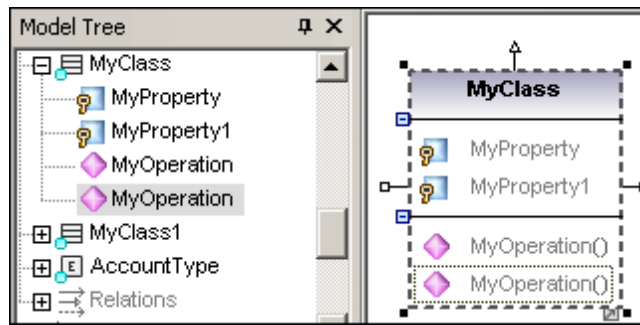
- **Pasting** a copied class into a different package, also inserts a **new** class, but keeps the original class name.
- In both cases the new class is also added to the Model Tree as well.

Paste (Ctrl+V) of copied **Properties or Operations**:

- **Pasting** a Property in the same class, inserts a **new** property with the source property name plus a sequential number e.g. MyProperty1.



- **Pasting** an Operation in the same class, inserts a **new** operation of the same name as the source operation.

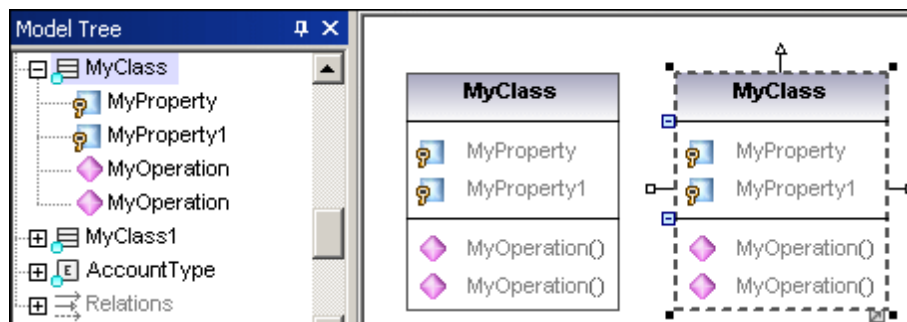


- In both cases a new property/operation is added to the Model Tree.

"Paste in Diagram only":

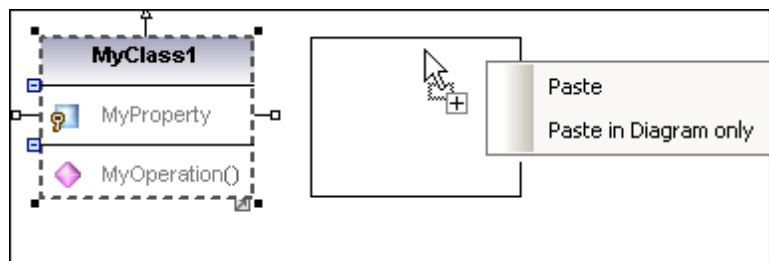
Whenever you use the context menu and select this option, a "link", or "view" to the element is created in the **diagram** you paste it into. Using the Class diagram as an example:

- "Paste in diagram only", creates a "view" to the original class
- The class is inserted into the diagram and displayed exactly as the source class
- A **new** class has **not** been added to the Model Tree!
- No class name or other Operation/Property changes are made
- Changing element properties in one of the "views", changes it in the other one automatically



Copy and pasting of elements using the mouse:

1. Click on the modeling element you want to copy.
2. Move the mouse pointer to the position you want to place the new element.
3. Hold down the **Ctrl** key. A small plus appears below the mouse pointer to signify that this is a copy procedure.
4. Release the mouse button.



A pop-up menu appears at this point allowing you to select between Paste, and Paste in Diagram only.

5. Select the option that you would like to perform.

Please note:

Using the mouse and **Ctrl** key allows you to copy, or move properties and operations directly within a class.

4.12 Adding/Inserting model elements

Model elements can be created and inserted into diagrams using several methods:

- By adding the elements to specific packages, in the Model Tree view
- By dragging existing elements from the Model Tree tab into the diagram tab
- By clicking a specific UML element icon, and inserting it into the diagram
- By using the context menu to add elements to the diagram (and automatically to the Model Tree view).

Please note that multiple elements can be selected in the Model Tree using either **Shift+Click**, or **Ctrl+Click**.

Adding elements in the Model Tree/Favorites tab:

- Right click a package, select New, and then select the specific element from the submenu.
This adds the new element to the Model Tree tab in the current project.

Inserting elements from the Model Tree view into a diagram:

Model elements can be inserted individually, or as a group. To mark multiple elements use the **Ctrl** key and click each item. There are two different methods of inserting the elements into the diagram: drag left, and drag right.

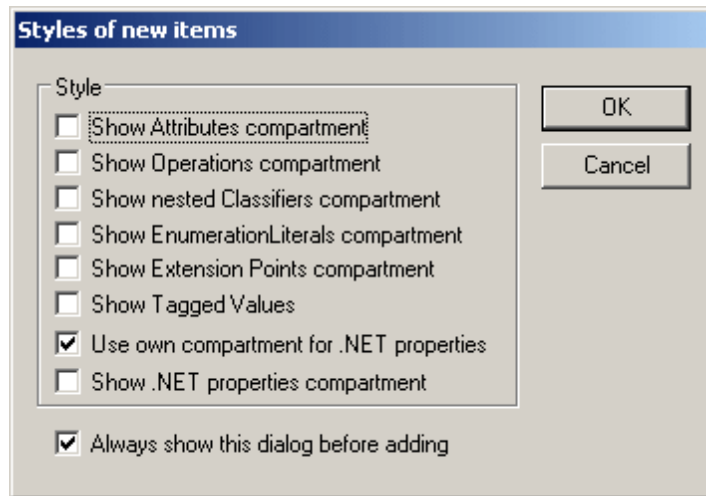
- **Drag left** (normal drag and drop) inserts elements immediately at the cursor position (any associations, dependencies etc. that exist between the currently inserted elements and the new one, are automatically displayed).
- **Drag right** (holding down the right mouse button and releasing it in the diagram tab) opens a pop-up menu from which you can select the specific associations, generalizations you want to display.



Example:

You want to replicate the Account Hierarchy diagram in a new class diagram.

1. Right click the **bankview** package and select **New | Class Diagram**.
2. Locate the abstract *Account* class in the model tree, and use **drag right** to place it in the new diagram.
The context menu shown above, is opened.
3. Select the **Insert with Generalization Hierarchy (specific)** item.



4. Deselect the check boxes for specific items you want to appear in the elements (Properties and Operations in this case).
5. Click OK.
The Account class **and its three subclasses**, are all inserted into the diagram tab. The Generalization arrows are automatically displayed.

Adding elements to a diagram using the icons in the icon bar:

1. Select the specific element you want to insert by clicking the associated icon in the icon bar.
2. Click in the diagram tab to insert the element.

Please note:

Holding down the **Ctrl** key before clicking in the diagram tab, allows you to insert multiple elements of the same type with each individual click in the diagram.

Adding elements to a diagram using the context menu:

- Right click the diagram background and select **New | element name**.

Please note:

Adding new elements directly to the diagram tab, automatically adds the same element to the Model Tree tab. The element is added to the package containing the UML diagram in the Model Tree view.

- Right click an element and select **Show | xx**
E.g. Right clicking the Account class and selecting **Show | Generalization hierarchy**. This then inserts the derived classes into the diagram as well.

4.13 Hyperlinking modeling elements

UModel now supports automatic and manual hyperlinking of modeling elements. Automatic hyperlinking occurs when selecting the specific setting when importing source code, or binary files, into a model.

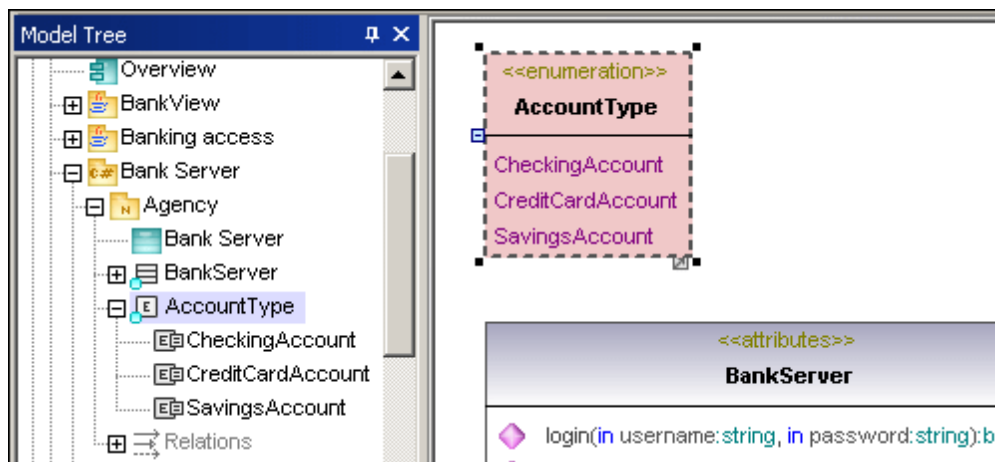
Manual hyperlinks are created between most modeling elements (except for lines) and:

- any [diagram](#) in the current *.UMP project
- any [element](#) on a diagram
- any element in the [Model Tree](#)
- external documents, e.g. PDF, Excel or Word documents
- web pages

Note:

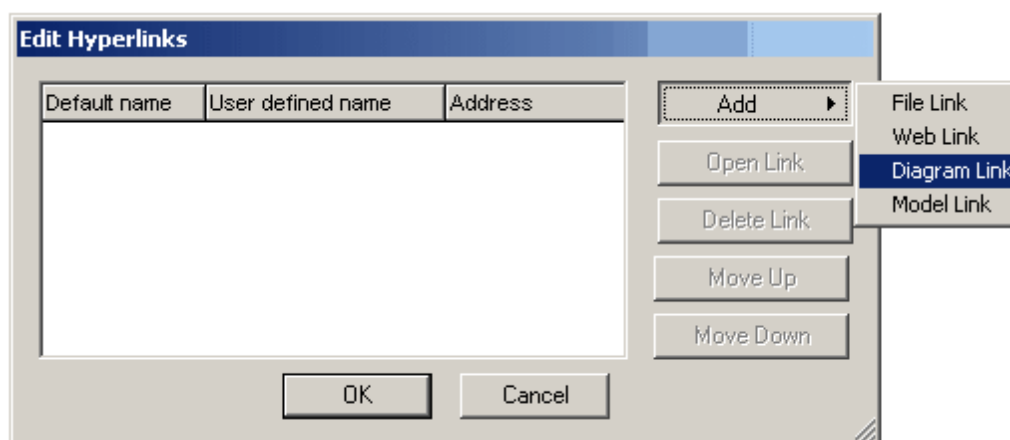
All manually created hyperlinks are also available in the HTML project documentation when generating [UML documentation](#).

Opening the Bank Server diagram under the Bank Server package displays the IBankAPI interface as well as the BankServer class. An enumeration element containing the names of the EnumerationLiterals is also visible. What we want to do is create a hyperlink from the Enumeration to the Account Hierarchy class diagram.



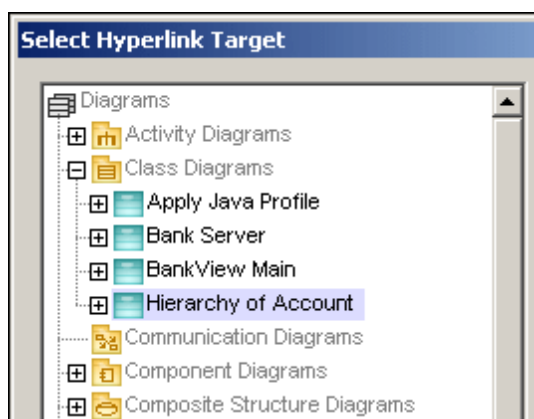
To create a diagram hyperlink:

1. Right click the element and select **Hyperlinks | Insert/Edit hyperlinks**.



This opens the Edit Hyperlinks dialog box in which you manage the hyperlinks.

2. Click the **Add** button and select **Diagram Link** to define a link to an existing diagram.



3. Select the hyperlink target that you want to be able to navigate to, e.g. Hierarchy of Account diagram, and click OK.

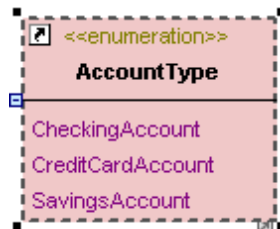


Double clicking in the User defined name column allows you to define your own link name.

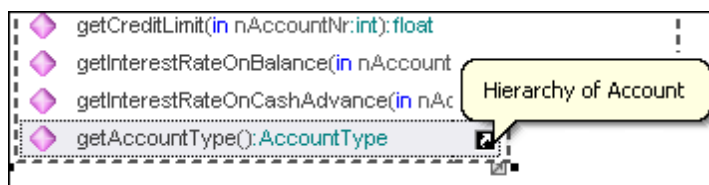
Note that you can add multiple, as well as different kinds of links from a single modeling element e.g. a web link to http://altova.com/support_help.html using the **Add | Web Link**.

Edit Hyperlinks		
Default name	User defined name	Address
Hierarchy of Account	Account Hierarchy	Hierarchy of Account ...
http://altova.com/suppc		http://altova.com/support_help.html

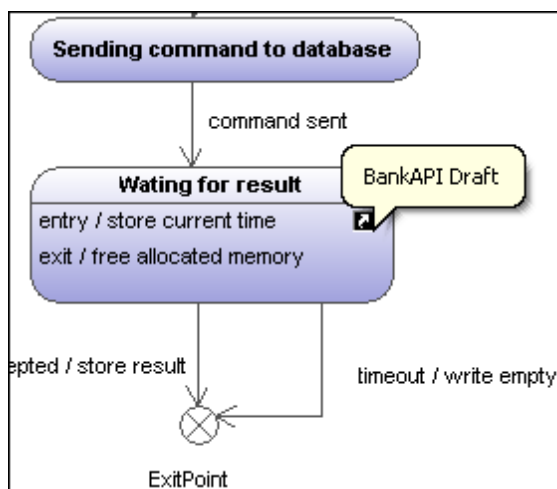
- Click OK when you have finished defining your hyperlinks.
A link icon has now been added to the top left of the enumeration element. Placing the mouse cursor over the link icon, displays a pop-up label with the name of the target element.



Hyperlinks on operations

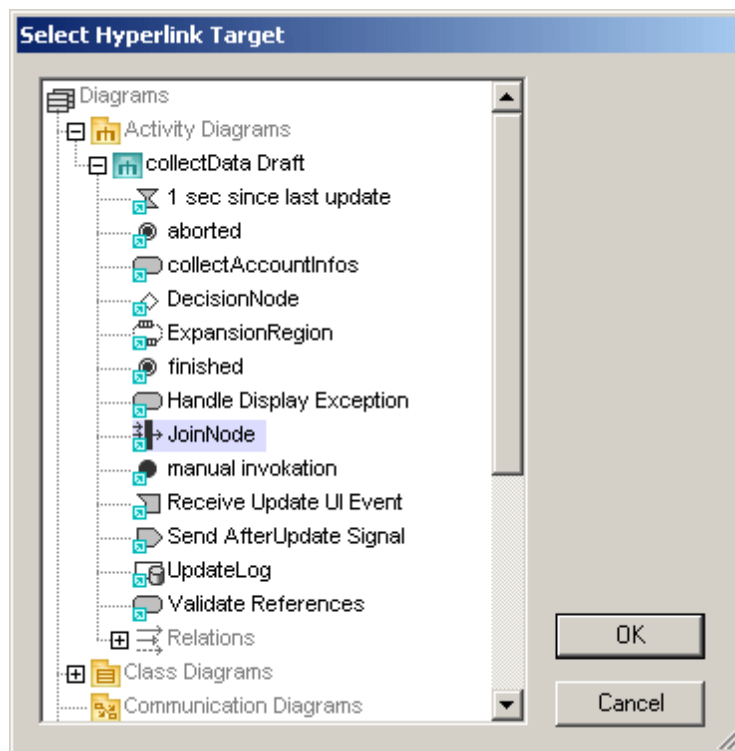


Hyperlinks from a state machine entry action



To create a link to a specific Diagram element (Diagram Link):

- Create the hyperlink as before but click the + sign to expand the diagram contents.



2. Select the specific modeling element you want to link to and click OK to confirm.
Clicking the link icon opens the designated diagram with the element visible and selected.

To create a link to an element in the Model Tree (Model Link):

- Use the same method as described above, but click **Add | Model Link** in the dialog box. Clicking the link once it has been inserted, navigates to the specific element you selected from the dialog box in the **Model Tree**.

Note: when generating project documentation, the hyperlinks to the model elements in the Model Tree are adjusted to point to the correct definitions in the generated documentation.

To create a link to a document:

1. Click the **Add | File Link** in the Edit Hyperlinks dialog box.
2. Select the document that you want to link e.g. *.doc, *.xls, *.pdf, etc.

To create a hyperlink from a note:

1. Select the text in the note by dragging, or double clicking a word.
2. Right click the selected text and select the menu object **Insert/Edit Hyperlinks**.
3. Use the Edit Hyperlinks dialog box to create a link to a diagram.

Click [here](#) to go to BankView Main

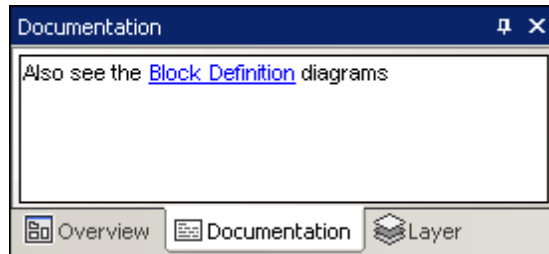
Please note:

Use the same method to create hyperlinks from **Comment** elements.

Also see the [Block Definition](#) diagrams

To create a link from the Documentation tab:

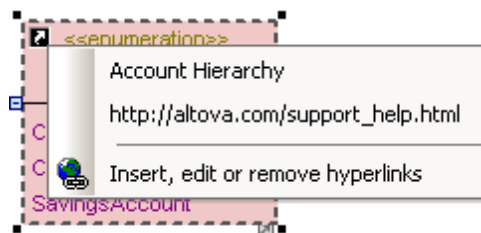
1. Enter the explanatory text into the Documentation tab.
2. Select the text you want to use as a link
3. Right click and select the type of link you want to create.



To navigate to a hyperlink target:

1. Click the hyperlink icon in the modeling element.
If only one target is defined then the target diagram, website etc., will appear immediately.

If multiple targets were defined, a pop-up dialog appears allowing you to select one of the available targets.



Clicking the first item opens the Hierarchy of Account diagram.

Navigating hyperlinks:

- Click the Previous  and Next  icons, in the main icon bar, to navigate the source and destination links.

To edit/change a hyperlink target:

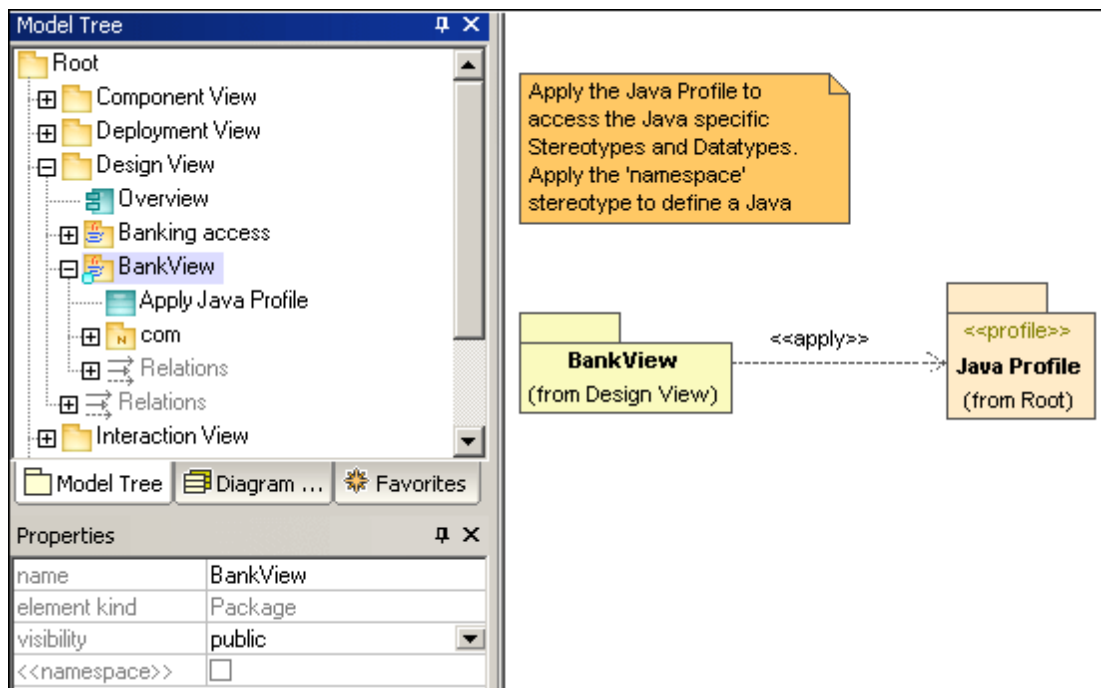
1. Right click the link icon and select **Insert, edit or remove hyperlinks** item.
2. Use the Edit Hyperlinks dialog box in to manage your hyperlinks.

4.14 Bank samples

The ...\\UModelExamples folder contains sample files which show different aspects of UML modeling in UModel. They are designed to show language specific models for Java, C# and a combination of both languages in one modeling project.

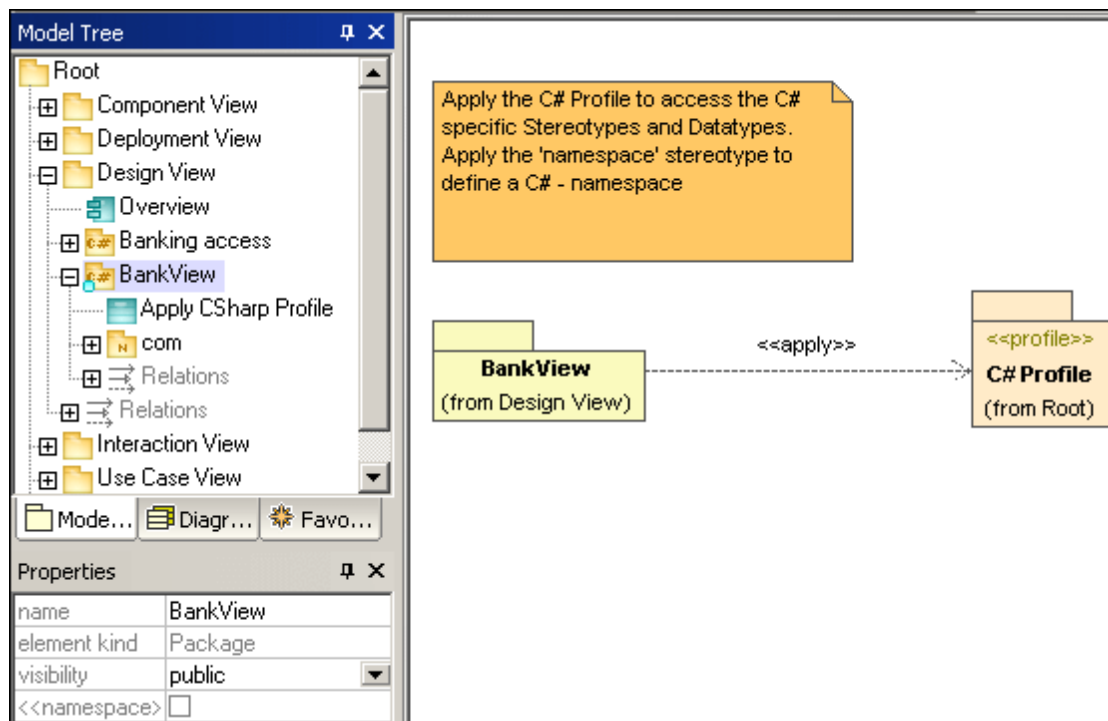
The **Bank_Java.ump** sample file is shown below:

- the Java profile has been assigned to the Bankview package
- the Java namespace root has been assigned to the Banking access and BankView packages.
- the Interaction View package contains two interaction elements which each contain a sequence diagram.



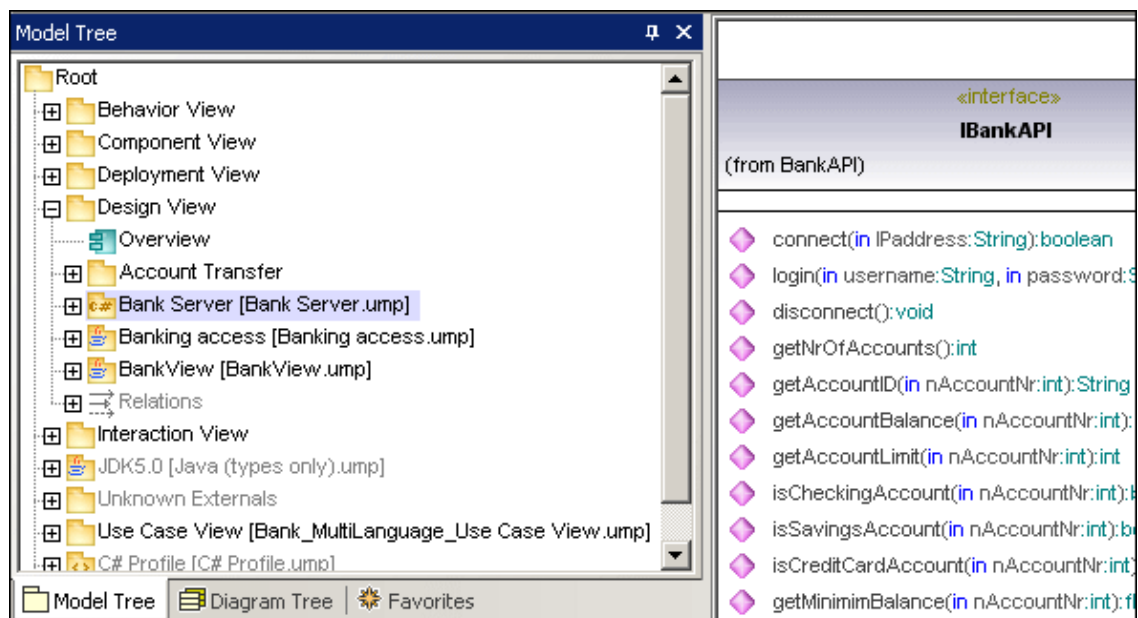
The **Bank_CSharp.ump** sample file is shown below:

- the C# profile has been assigned to the BankView package
- the C# namespace root has been assigned to the Banking access and BankView packages.
- the Interaction View package contains two interaction elements which each contain a sequence diagram.



The **Bank_MultiLanguage.ump** sample file is shown below:

- the Java profile has been assigned to the BankView package
- the C# namespace root has been assigned to the Bank Server package
- the Java namespace root has been assigned to the BankView package.
- the Interaction View package contains two interaction elements which each contain a sequence diagram.
- the project has been split up into 4 editable subprojects, Bank Server.ump, Banking access.ump, BankView.ump and Bank_Multilanguage_Use Case View.ump.



Chapter 5

UModel Command Line Interface

5 UModel Command Line Interface

In addition to the graphical user interface, UModel also has a command line interface. To open the command line interface, run the **UModelBatch.exe** file available in the **C:\Program Files\Altova\UModel2016** directory. If you run UModel 32-bit on a 64-bit operating system, the path is **C:\Program Files (x86)\Altova\UModel2016**.

The command line parameter syntax is shown below, and can be displayed in the command prompt window by entering: **umodelbatch /?**

Note: If the path or file name contains spaces, enclose it in quotes, for example: "c:\Program Files\...\MyProject.ump".

```
usage: umodelbatch [project] [options]

/? or /help ... display this help information

project    ... project file (*.ump) see also File: New/Load/Save options
/new=[file] ... create/save/save as new project
/set       ... set options permanent
/gui       ... display UModel user interface

commands (executed in given order):
/chk       ... check project syntax
/isd=path  ... import source directory
/isp=file  ... import source project file
            (*.project,*.xml,*.jspx,*.csproj,*.csdproj,*.vbproj,*.vbdproj,*.sln,*.bdsproj)
/ibt=list  ... import binary types (specify binary[typename] list)
            (';'=separator, '*'=all types, '#' before assembly names)
/ixd=path  ... import XML schema directory
/ixs=file  ... import XML schema file (*.xsd)
/m2c      ... update program code from model (export/forward engineer)
/c2m      ... update model from program code (import/reverse engineer)
/ixf=file  ... import XMI file
/exf=file  ... export to XMI file
/inc=file  ... include file
/mrg=file  ... merge file
/doc=file  ... write documentation to specified file
/lue=[cpri] ... list all elements not used on any diagram (i.e. unused)
/ldg       ... list all diagrams
/lcl       ... list all classes
/lsp       ... list all shared packages
/lip       ... list all included packages

options for save as new project:
/npad=opt  ... adjust relative file paths (Yes | No | MakeAbsolute)

options for import commands:
/iclg=lang ... code language (Java1.4 | Java5.0 | Java6.0 | Java7.0 |
Java8.0 | C#1.2 | C#2.0 | C#3.0 | C#4.0 | C#5.0 | C#6.0 | VB7.1 | VB8.0 |
VB9.0)
/ipsd=[0|1] ... process sub directories (recursive)
/irpf=[0|1] ... import relative to UModel project file
```

```
/ijdc[=0|1] ... JavaDocs as Java comments
/icdc[=0|1] ... DocComments as C# comments
/icds[=lst] ... C# defined symbols
/ivdc[=0|1] ... DocComments as VB comments
/ivds[=lst] ... VB defined symbols (custom constants)
/imrg[=0|1] ... synchronize merged
/iudf[=0|1] ... use directory filter
/iflt[=lst] ... directory filter (presets /iudf)

options for import binary types (after /iclg):
/ibrv=vers ... runtime version
/ibpv=path ... override of PATH variable for searching native code
libraries
/ibro[=0|1] ... use reflection context only
/ibua[=0|1] ... use add referenced types with package filter
/ibar[=flt] ... add referenced types package filter (presets /ibua)
/ibot[=0|1] ... import only types
/ibuv[=0|1] ... use minimum visibility filter
/ibmv[=key] ... keyword of required minimum visibility (presets /ibuv)
/ibsa[=0|1] ... suppress attribute sections / annotation modifiers
/iboa[=0|1] ... create only one attribute per attribute section
/ibss[=0|1] ... suppress 'Attribute' suffix on attribute type names

options for diagram generation:
/dgen[=0|1] ... generate diagrams
/dopn[=0|1] ... open generated diagrams
/dsac[=0|1] ... show attributes compartment
/dsoc[=0|1] ... show operations compartment
/dscc[=0|1] ... show nested classifiers compartment
/dstv[=0|1] ... show tagged values
/dudp[=0|1] ... use .NET property compartment
/dspd[=0|1] ... show .NET property compartment

options for export commands:
/ejdc[=0|1] ... Java comments as JavaDocs
/ecdc[=0|1] ... C# comments as DocComments
/evdc[=0|1] ... VB comments as DocComments
/espl[=0|1] ... use user defined SPL templates
/ecod[=0|1] ... comment out deleted
/emrg[=0|1] ... synchronize merged
/egfn[=0|1] ... generate missing file names
/eusc[=0|1] ... use syntax check

options for XMI export:
/exid[=0|1] ... export UUIDs
/exex[=0|1] ... export UModel specific extensions
/exdg[=0|1] ... export diagrams (presets /exex)
/exuv[=ver] ... UML version (UML2.0 | UML2.1.2 | UML2.2 | UML2.3 | UML2.4)

options for merge file:
/mcan=file ... common ancestor file

options for documentation generation:
/doof=fmt ... output format (HTML | RTF | MSWORD | PDF)
/dsps=file ... SPS design file
```

In the projects section:

- The **/new** parameter defines the path and file name of the new project file (*.ump). It can also be used to save an existing project under a different name e.g. **UModelBatch.exe MyFile.ump /new=MyBackupFile.ump** (see also [File: New/Load/Save options](#)).
- The **/set** parameter overwrites current default settings in the registry, with the options/settings defined here.
- The **/gui** parameter displays the UModel interface during the batch process.

Example 1

The following command imports source code and creates a new project file. Notice that the project path contains spaces and is enclosed in quotes.

```
"C:\Program Files\Altova\UModel2016\UModelBatch.exe" /new="C:\Program Files
\Altova\UModel2016\UModelBatchOut\Fred.ump" /isd="X\TestCases\UModel\Fred" /
set /gui /iclg=Java5.0 /ipsd=1 /ijdc=1 /dgen=1 /dopn=1 /dmax=5 /chk
```

/new:	Specifies that the newly-created project file should be called "Fred.ump" in C:\Program Files\Altova\UModelBatchOut\
/isd=	Specifies that the root directory to import into should be "X\TestCases\UModel\Fred"
/set:	Specifies that any options used in the command line tool will be saved in the registry (When subsequently starting UModel, these settings become the default settings).
/gui:	Display the UModel GUI during batch processing.
/iclg:	UModel will import the code as Java5.0.
/ipsd=1:	Recursively process all subdirectories of the root directory specified in the /isd parameter.
/pfd=1:	Create packages in the UModel project for each imported directory.
/ijdc=1:	Create JavaDoc from comments where appropriate.
/dgen=1:	Generate diagrams.
/dopn=1:	Open generated diagrams.
/dmax=5:	Open a maximum of five diagrams.
/chk:	Perform a syntax check.

Example 2

The following command imports source code from **X:\TestCases\UModel**, and saves the resulting project file to **C:\Program Files\Altova\UModel2016\UModelBatchOut\finalclass.ump**.


```
"C:\Program Files\Altova\UModel2016\UModelBatch.exe" /new="C:\Program Files
\Altova\UModel2016\UModelBatchOut\finalclass.ump" /isd="X:\TestCases\UModel
\" /iclg=Java5.0 /ipsd=1 /ijdc=1 /dgen=1 /dopn=1 /dmax=5 /dsat=1 /dsnc=1 /chk
```

/dsat=1	Suppress attributes in the generated diagrams.
/dsnc=1	Suppress nested classifiers in the generated diagrams.

Example 3

The following command synchronizes code using an existing project file ("C:\Program Files\Altova\UModel\QMS\UModelBatchOut\Fred.ump").

```
"C:\Program Files\Altova\UModel2016\UModelBatch.exe" "C:\Program Files\Altova
\UModel2016\UModelBatchOut\Fred.ump" /m2c /ejdc=1 /ecod=1 /emrg=1 /egfn=1 /
eusc=1
```

/m2c	Update the code from the model.
/ejdc:	Comments in the project model should be generated as JavaDoc.
/ecod=1:	Comment out any deleted code.
/emrg=1	Synchronize the merged code.
/egfn=1:	Generate any missing file names in the project.
/eusc=1	Use the syntax check.

5.1 File: New / Load / Save options

Full batch mode i.e. `/gui` parameter not used.

new

UModelBatch /new=xxx.ump (options)

creates a new project, executes options, xxx.ump is always saved (regardless of options)

auto save

UModelBatch xxx.ump (options)

loads project xxx.ump, executes options, xxx.ump is saved **only** if document has changed (like `/ibt`)

save

UModelBatch xxx.ump (options) /new

loads project xxx.ump, executes options, xxx.ump is **always** saved (regardless of options)

save as

UModelBatch xxx.ump (options) /new=yyy.ump

loads project xxx.ump, executes options, always saves xxx.ump as yyy.ump (regardless of options)

Batch mode with **UModel user interface** visible i.e. `/gui` parameter used.

new

UModelBatch /gui /new (options)

creates a new project, executes options, nothing saved, the GUI is left open

save new

UModelBatch /gui /new=xxx.ump (options)

creates a new project, executes options, xxx.ump saved, the GUI is left open

user mode

UModelBatch /gui xxx.ump (options)

loads project xxx.ump, executes options, nothing saved, the GUI is left open

save

UModelBatch /gui xxx.ump (options) /new

loads project xxx.ump, executes options, xxx.ump is saved, the GUI is left open

save as

UModelBatch /gui xxx.ump (options) /new=yyy.ump

loads project xxx.ump, executes options, xxx.ump is saved as yyy.ump, the GUI is left open

The project will be saved successfully provided that no critical errors occur while executing the options.

Chapter 6

Projects and code engineering

6 Projects and code engineering

UModel now supports all Java specific constructs, among them:

- Java annotations
- Attributes, operations and nested qualifiers for EnumerationLiterals
- Enumerations can realize interfaces
- NetBeans project files

Reverse engineering now supports:

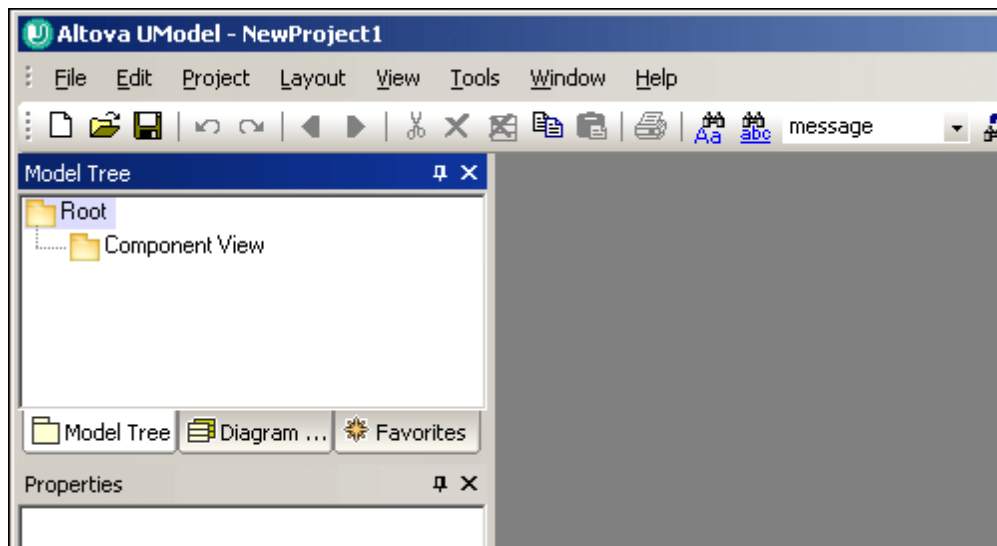
- The ability to generate a single diagram for all reverse engineered elements
- Possibility to show/hide anonymous bound elements on diagrams
- Ability to automatically create hyperlinks from packages to their corresponding package content diagrams during the import process.
- Ability to resolve [aliases](#).
- Writing of Unicode characters into new source code files
- Ability to create associations from .NET properties

To create a new project:

1. Click the **New** icon in the icon bar, (or select the menu item **File | New**).

The Root and Component packages are automatically inserted when a new project is created, and are visible in the Model Tree tab. A new project with the default name NewProject1 is created.

Note that starting UModel opens a new project automatically.



A newly created UModel project consists of the following packages:

- Root package, and
 - Component View package
- These two packages are the only ones that cannot be renamed, or deleted.

All project relevant data is stored in the UModel project file, which has an *.ump extension. Each folder symbol in the Model Tree tab represents a UML package.

UModel Project workflow:

UModel does not force you to follow any predetermined modeling sequence.

You can add any type of model element: UML diagram, package, actor etc., to the project in any sequence (and in any position) that you want; Note that all model elements can be inserted, renamed, and deleted in the Model Tree tab itself, you are not even forced to create them as part of a diagram.

To insert a new package:

1. Right click the package you want the new package to appear under, either Root, or Component View in a new project.
2. Select **New | Package**.

A new package is created under an existing one. The name field is automatically highlighted allowing you to enter the package name immediately.

- Packages are the containers for all other UML modeling elements, use case diagrams, classes, instances etc.
- Packages can be **created**, at any position in the Model Tree.
- Packages/contents can be **moved**/copied to other packages in the Model Tree (as well as into valid model diagrams in the diagram tab).
- Packages and their contents can be **sorted** (in the Model Tree tab) according to several criteria.
- Packages can be placed within other packages.
- Packages can be used as the **source**, or **target** elements, when merging, or synchronizing code.

To have elements appear in a UML diagram, you have to:

1. Insert a new UML diagram, by right clicking and selecting **New | (Class) Diagram**.
2. Drag and drop an existing model element from the Model Tree into the newly created Diagram, or
3. Use the context menu within the diagram view, to add new elements directly.

To save a project:

Select the menu option **File | Save as...** (or File | Save). Note: Using the Tools | Option | File tab, you can define if you want the *.ump file to be saved in a "pretty-print" format on saving.

To open a project:

Select the menu option **File | Open**, or select one of the files in the file list.

Please note:

Changes made externally to the project file, or included file(s), are automatically registered and cause a prompt to appear. You can then choose if you want to reload the project or not.

To move a project:

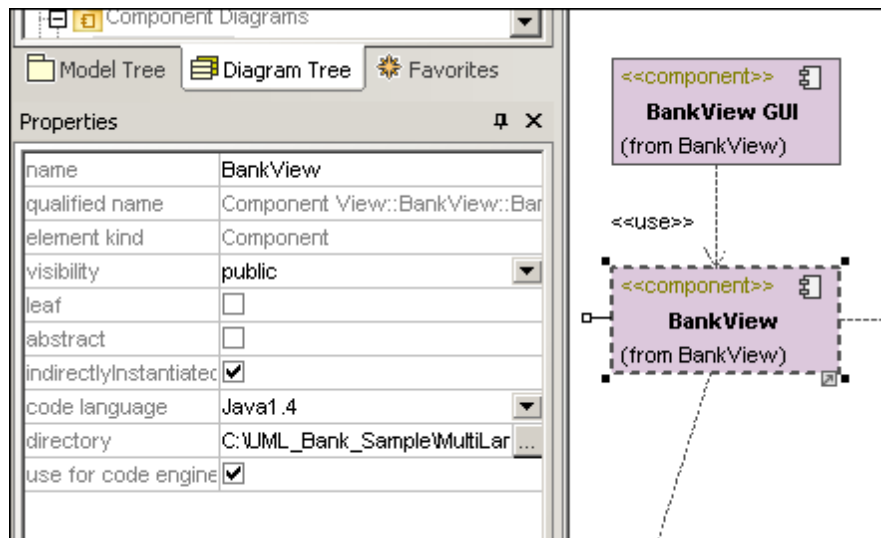
UModel projects and generated code can be easily moved to a different directory (or a different computer) and be resynchronized there.

There are two ways to do this:

- Select the menu option **File | Save As...**, and answer **Yes** at the prompt if you want to adjust the file paths to the new project location.
- Copy the UModel project (*.ump) to a new location, adjust the paths of the code generation package(s) in the Component Diagram view to the new directory location.

E.g. using BankMultilanguage.ump:

1. Open the Component Diagram "**Overview**" and click the BankView component.
2. Update the entry in the **directory** field to the new project location.
3. Re-synchronize the model and code.



6.1 Minimalist UModel project - starting from scratch

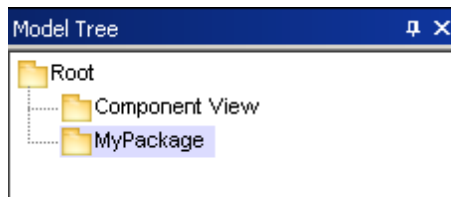
This section describes the steps necessary to create a project from scratch and successfully generate code for a single class. Although it is very minimalist, it uses various diagrams to show how to add methods etc., it is not the absolute minimum needed to produce code.

All of the actions could be achieved using only the Model Tree tab, there is no need to use modeling diagrams to create classes or methods.

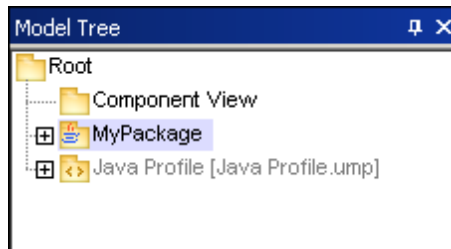
Creating a new project and defining the Namespace Root

In the Model Tree pane of UModel,

1. Select **File | New** to create a new modeling project.
2. Right click the **Root** package and select **New Element | Package**, name the package e.g. MyPackage.



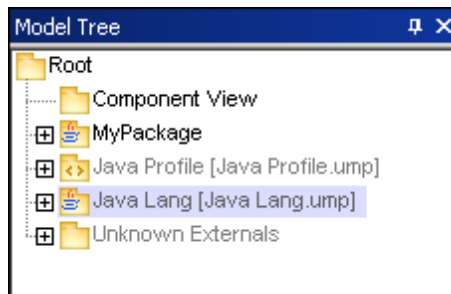
3. Right click MyPackage and select **Code Engineering | Set as Java Namespace Root**, click OK to apply the UModel Java profile to the package. This adds a Java Profile package to the Model Tree.



Including Java Lang to supply JDK datatypes

1. Click the Root package and select the menu option **Project | Include Subproject**.
2. Click the Java 1.4 tab and select the **Java Lang.ump** package. Click OK to use the default option "Include by reference".

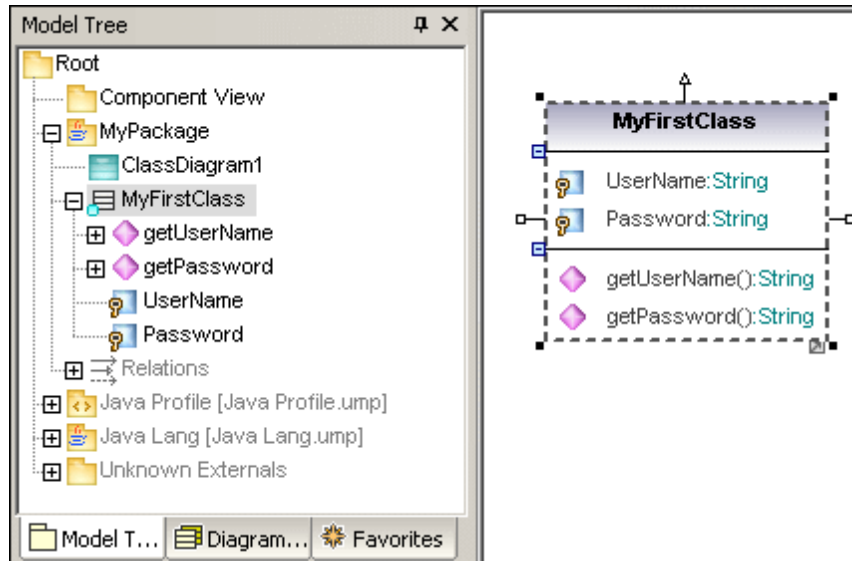
This adds a Java Lang, and an Unknown Externals package to the Model Tree.



Creating the class properties and methods

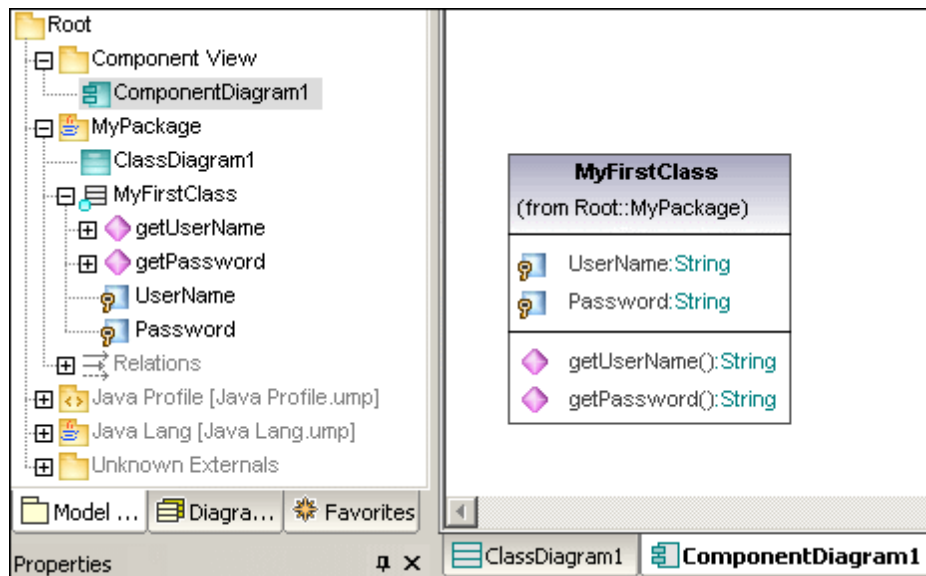
1. Right click **MyPackage** and select **New Diagram | Class Diagram**.
2. Right click in the class diagram and select **New | Class** to create a new class in the diagram e.g. **MyClass**.

3. Press F7 and add some attributes e.g. `UserName:String` and `Password:String`.
4. Press F8 and add some operations e.g. `GetUserName():String` and `GetPassword():String`.

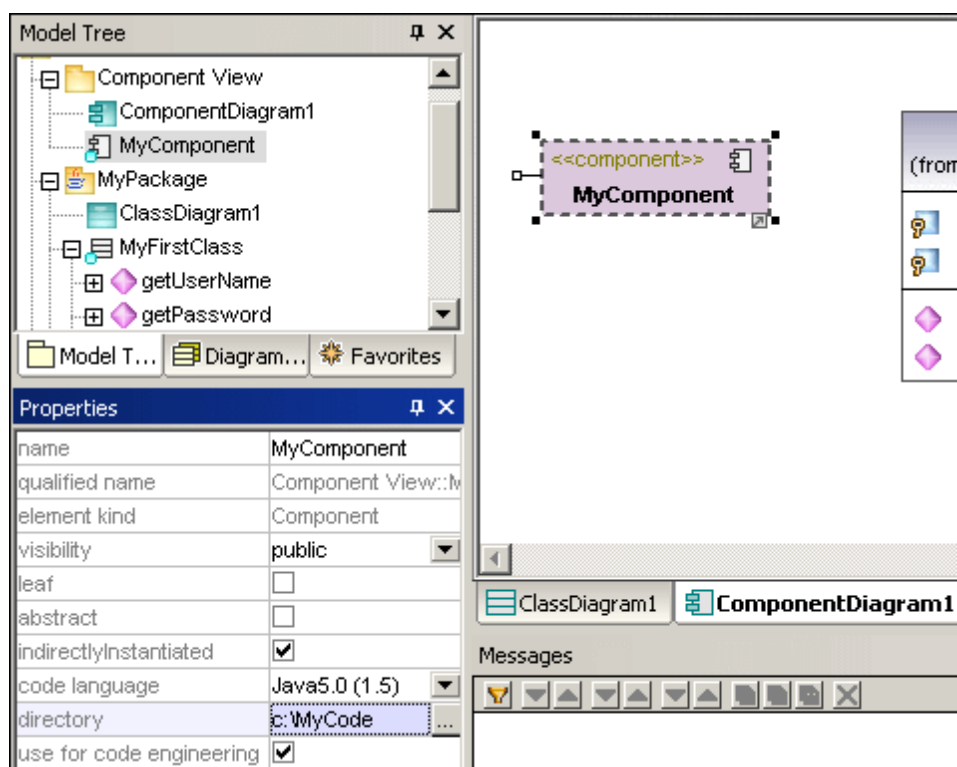


Creating a Component and defining the code directory

1. Right click the **Component View** package and add a new Component diagram.
2. Drag the **MyFirstClass** class element from the Model Tree into the Component diagram.

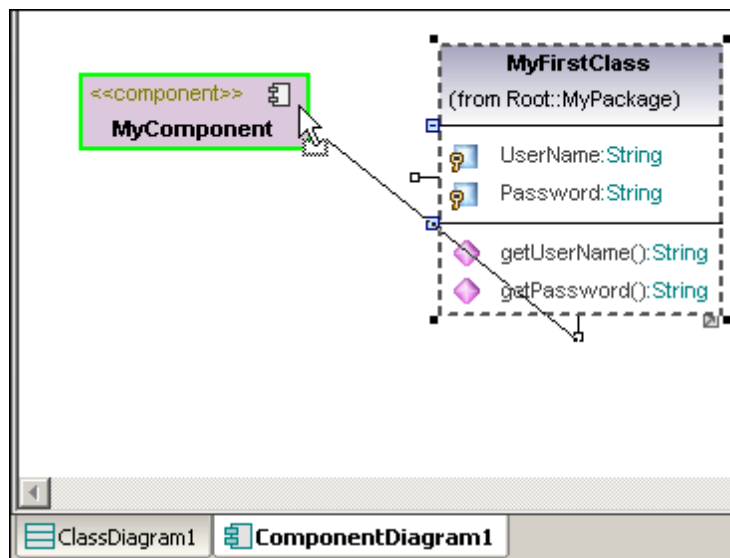


3. Add a new component to the diagram e.g. **MyComponent**.
4. Click the component in the diagram to select it, then click in the **directory** field of the Properties window and enter the directory you want the code to be placed in e.g. `C:\MyCode`.



Realizing the class

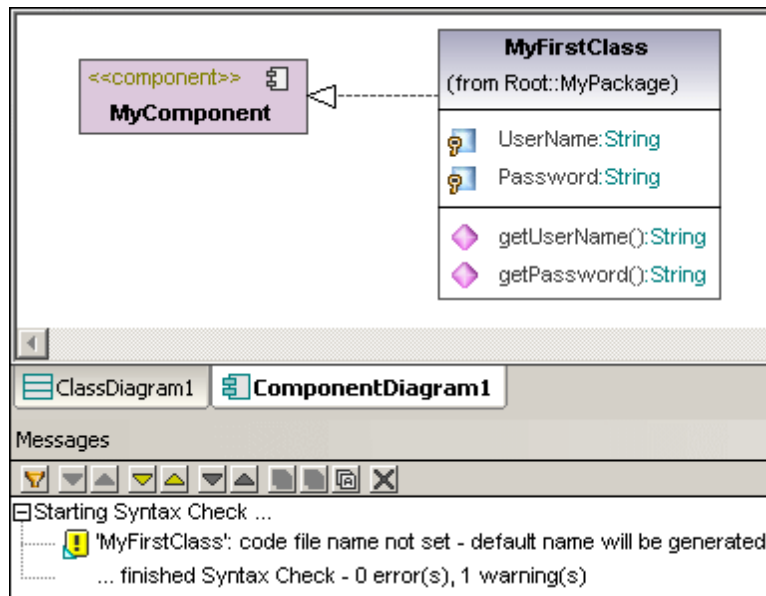
1. Click **MyClass** and drag the "**ComponentRealization**" handle at the bottom of the element and drop it on the new component, **MyComponent**.



A class has to be "realized" before code can be generated. Note that you could also drag the class and drop it on the component directly in the Model Tree.

Syntax check and generating code

1. Select **Project | Check Project Syntax** to check to see if everything is OK.



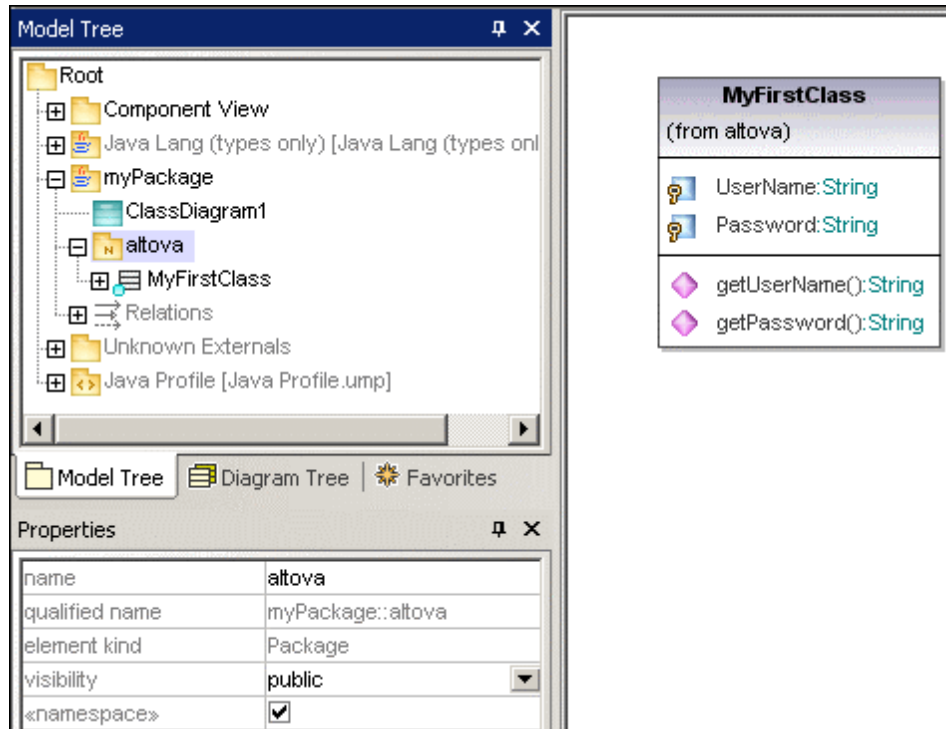
Zero errors and one warning are generated. Although the code name was not previously set, a default name will be generated automatically.

2. Select **Project | Merge Project code from UModel Project...** to output/generate the Java code.

Creating a namespace

If you want to generate the **class** inside a specific namespace:

1. Add a new package below myPackage e.g. altova.
2. Click the package and activate the <<namespace>> check box in the Properties window.



3. In the Model Tree window, move the MyFirstClass class into the altova package.
When code is generated, the class will now be in the namespace **altova**.

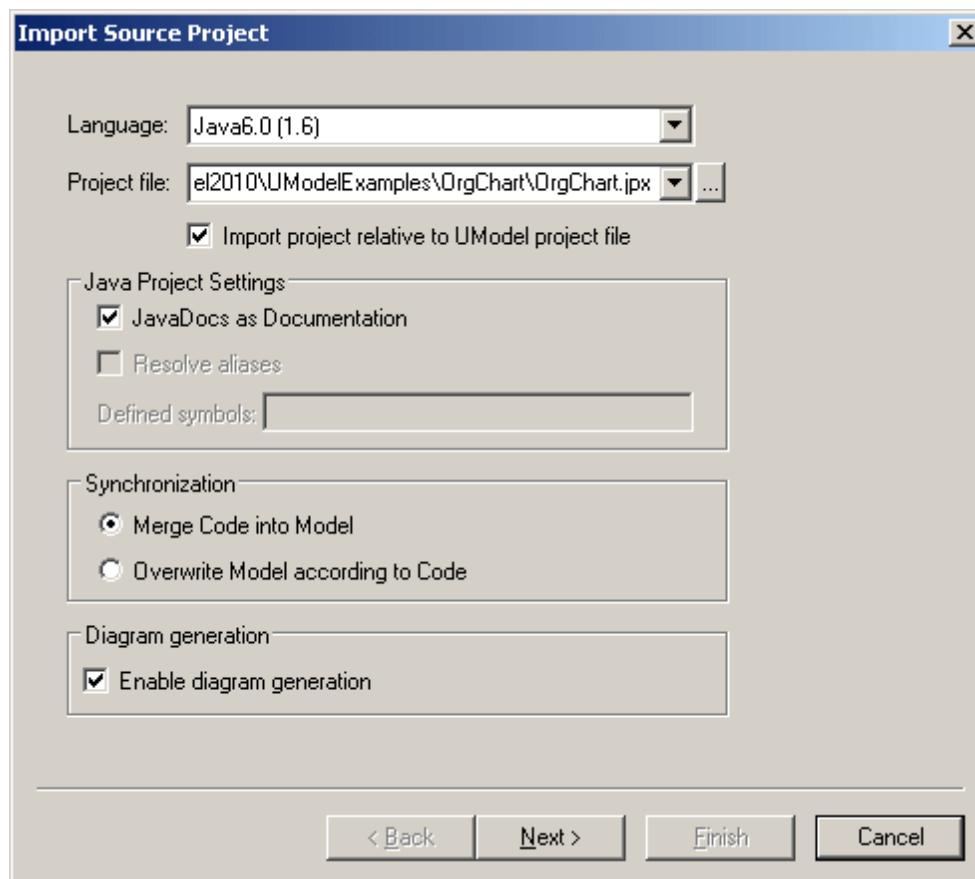
6.2 Importing source code into projects

Source code can be imported as a source project or as a source directory. For an example of importing a **source directory** please see [Round-trip engineering \(code - model - code\)](#) in the tutorial.

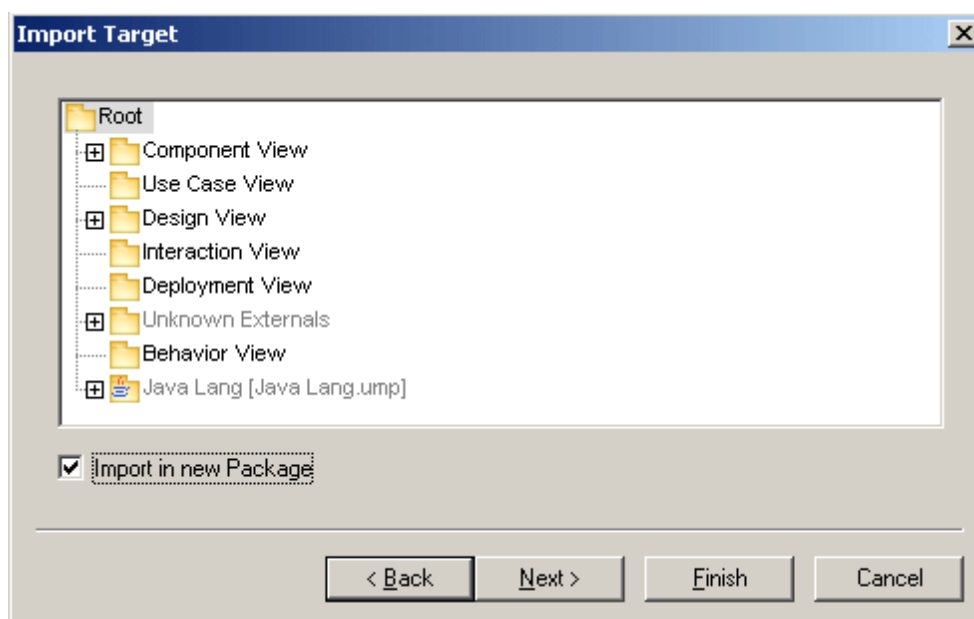
- JBuilder **.jpx**, Eclipse **.project** project files, as well as NetBeans (**project.xml**) are currently supported.
- C# projects / Visual Basic projects (Visual Studio **sln**, **cspjproj**, **csdprj**..., **vbproj**, **vbp** as well as Borland **.bdsproj** project files)

To import an existing project into UModel:

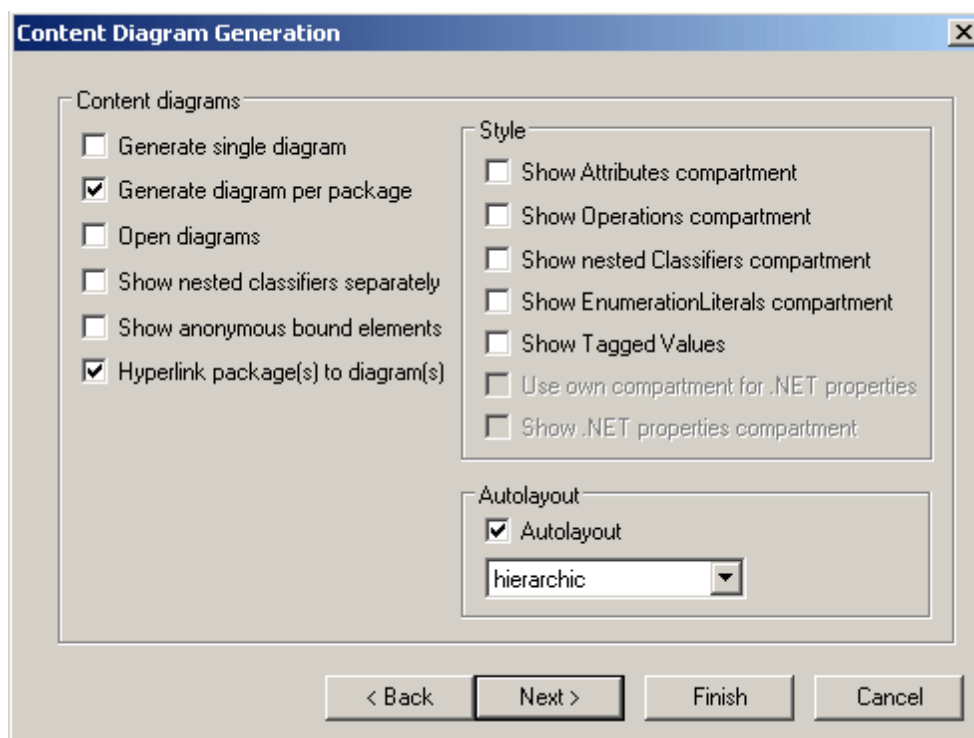
1. Select **Project | Import source project**.
2. Click the browse button  in the "Import Source Project" dialog box.



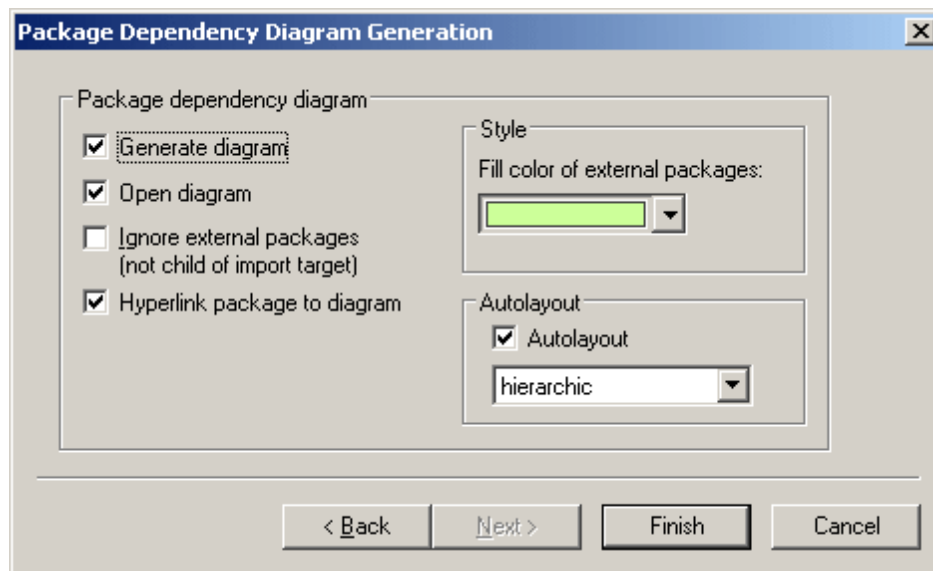
3. Select the project file type (in this example, **.jpx**), and click Open. This JBuilder project file is available in the OrgChart.zip file in the ...**UModelExamples** folder. Note that the option to import the project relative to the UModel project file is active. To instruct UModel to generate [Class Diagrams](#) and [Package Diagrams](#) from the source code, select the **"Enable diagram generation"** check box, and click Next.
4. Activate the "Import in new package" check box (or click the respective package in the Import Target window).



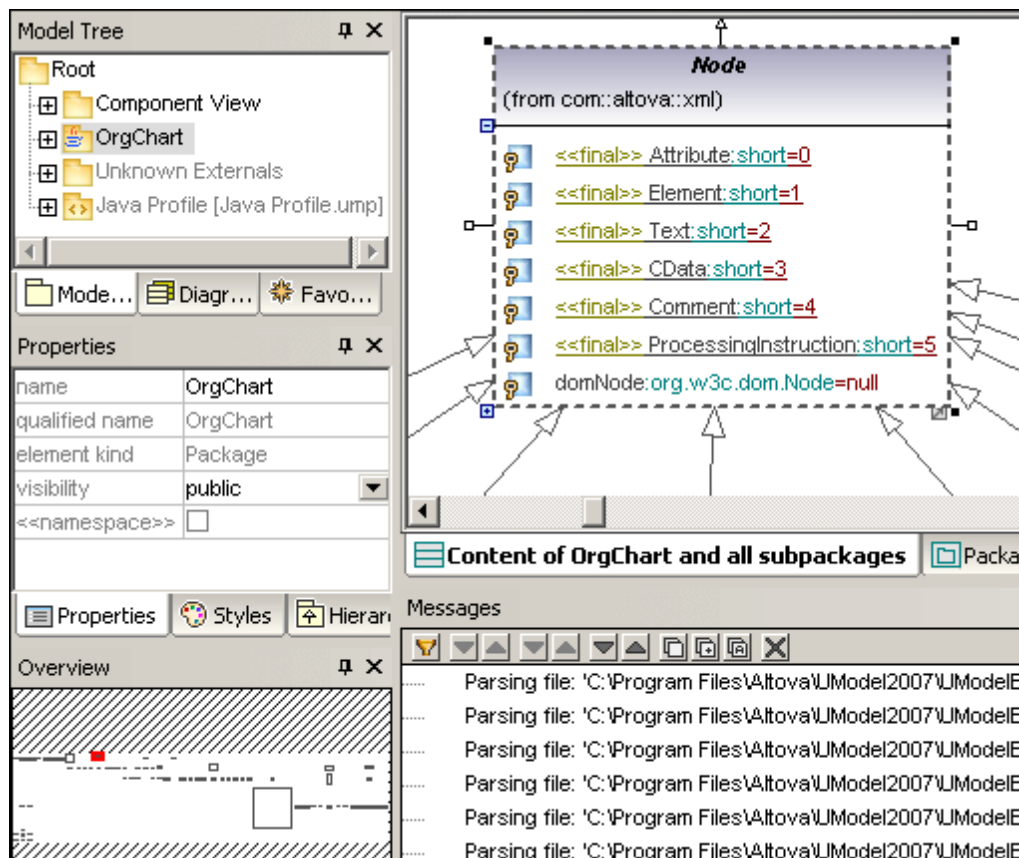
5. When prompted, select the **Generate single diagram** and/or **Generate diagram per package** options. The diagram styling options can be changed later if necessary.



6. Click Next to continue. This dialog box allows you to define the package dependency generation settings.



- Click Finish to use the default settings. The project is parsed and the UModel model is generated.



Please note:

If you are importing into an existing project, you will be prompted for the package it should be imported into. If you are using a new project, an OrgChart folder is

automatically created.

Resolve aliases:

When reverse engineering code which contains namespace or class aliases, you can now choose if the alias is to be resolved. (Check box "Resolve aliases" in Import Source Project/Directory dialog box).

When updating code, i.e. model to code, alias definitions are retained in the code as is. The scope of the alias definitions are the files in which they appear.

e.g.

```
using Q = System.Collections.Generic.Queue<String>;  
Q myQueue;
```

Any potentially conflicting aliases are added to the "Unknown externals" if their use is unclear.

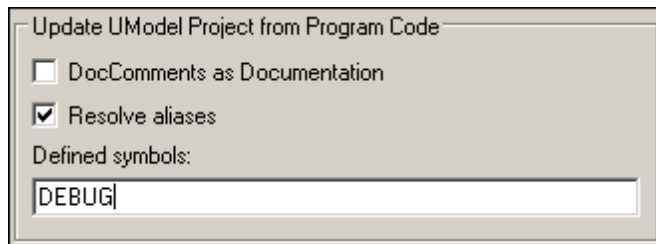
Note:

The Resolve alias settings can be changed at any time in the Project Settings dialog box.

Defined symbols

C# or Visual Basic code allow you to enter a list of "Defined symbols" in the field of that name. These directives are used to conditionally compile sections of code. Enter these directives as a semicolon delimited list. UModel takes the defined symbols into account during the code engineering process.

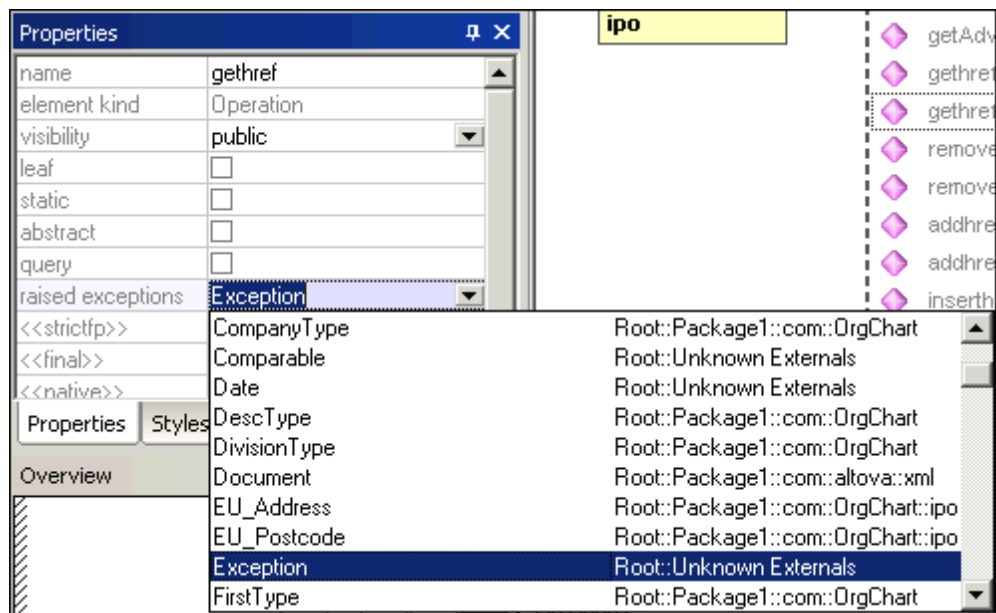
UModel outputs all symbols used in the source code in the message window, when the reverse-engineering process is completed.



E.g. #If DEBUG Then
Dim i = 10
#Else
dim a = 20
#End If

Raised exceptions

Clicking an operation in one of the classes, then clicking the **Exception** combo box, displays the exception information that an operation can throw.



6.3 Importing Java, C# and VB binaries

UModel supports the import of C# , Java and VB binaries. This is extremely useful when working with binaries from a third party, or the original source code has become unavailable.

If you intend to import Java binary files, the Java Runtime Environment (JRE), or Development Kit (JDK) version 1.4, 1.5, 1.6, 1.7, 1.8 must be installed. Type import is supported for class archives targeting these environments, i.e. adhering to the Java Virtual Machine Specification.

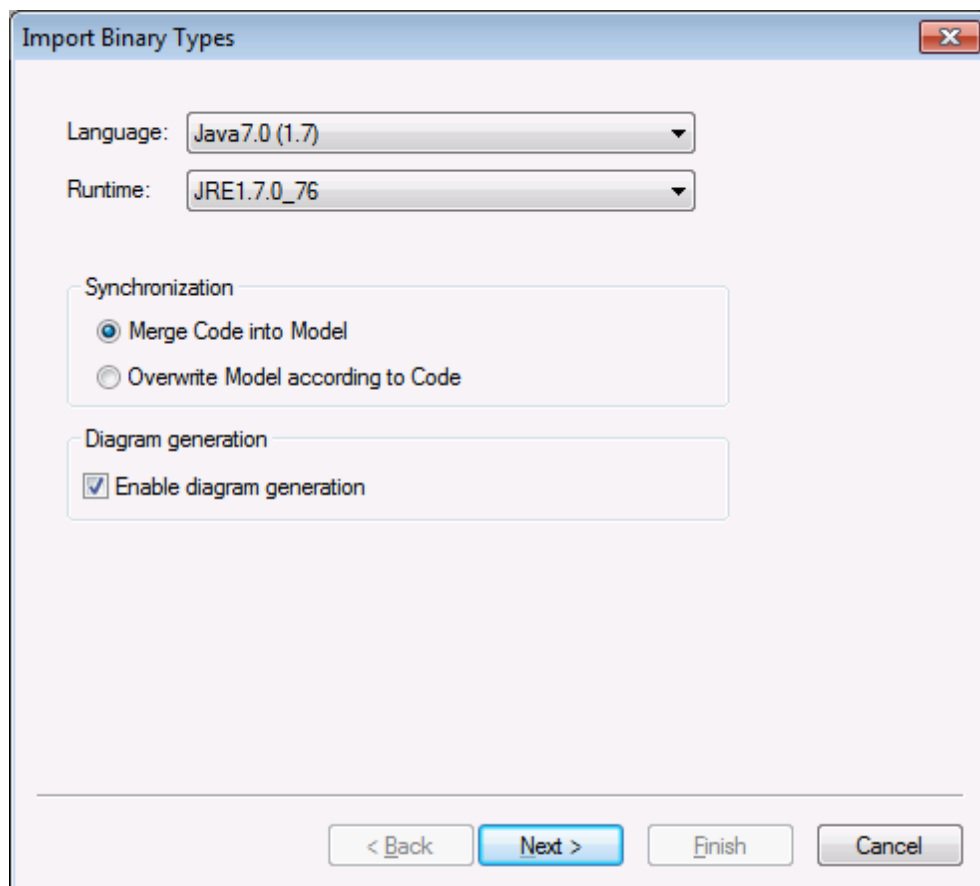
If you intend to import C# or VB binary files, the .NET Framework must be installed. Type import is supported for assemblies targeting .NET Framework 1.1, 2.0, 3.0, 3.5, 4.0, 4.5, 4.6, and .NET Compact Framework v1.0, v2.0, v3.5 (for PocketPC, Smartphone, WindowsCE).

These requirements only apply if you intend to import Java or C# or VB.NET binaries; if you do not, there is no need for the Java Runtime Environment, or the .NET Framework to be installed.

The import of obfuscated binaries is not supported.

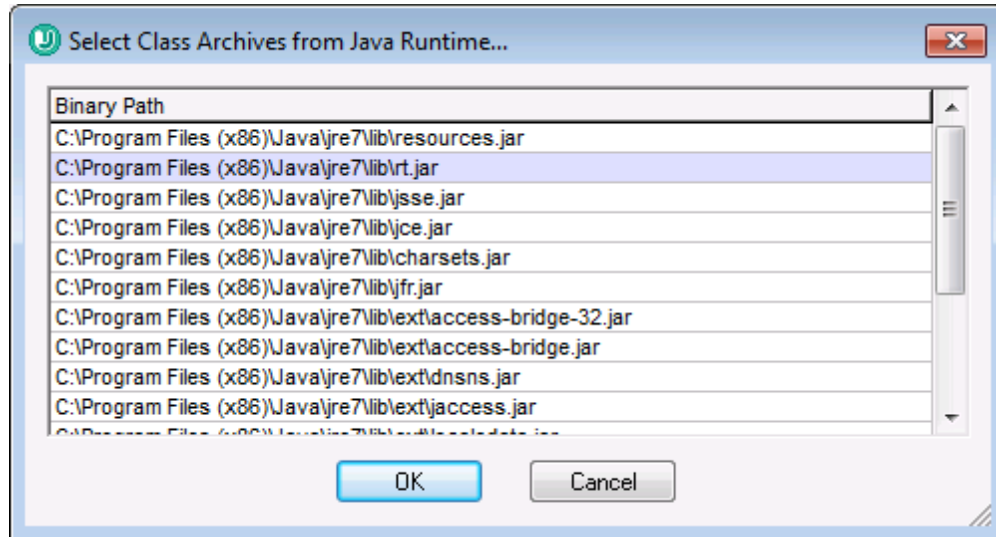
To import binary files:

1. Select the menu option **Project | Import Binary Types**.

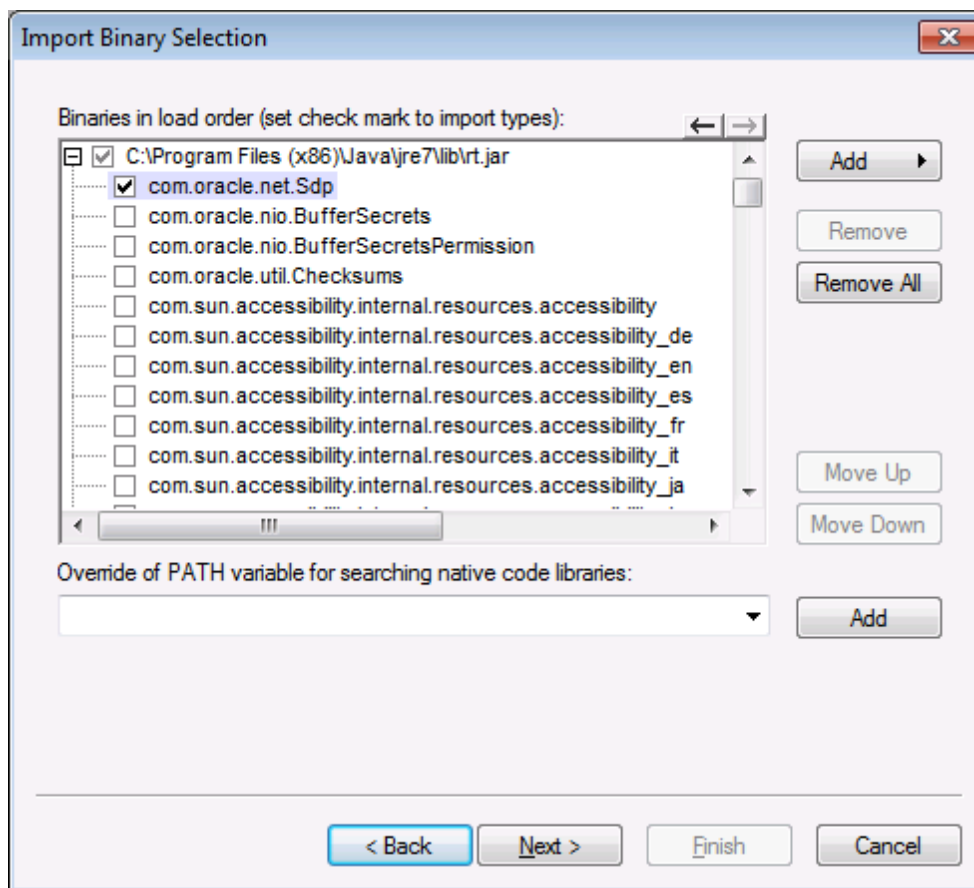


2. Select the language and runtime edition.
3. To instruct UModel to generate [Class Diagrams](#) and [Package Diagrams](#) from the source code, select the **Enable diagram generation** check box.

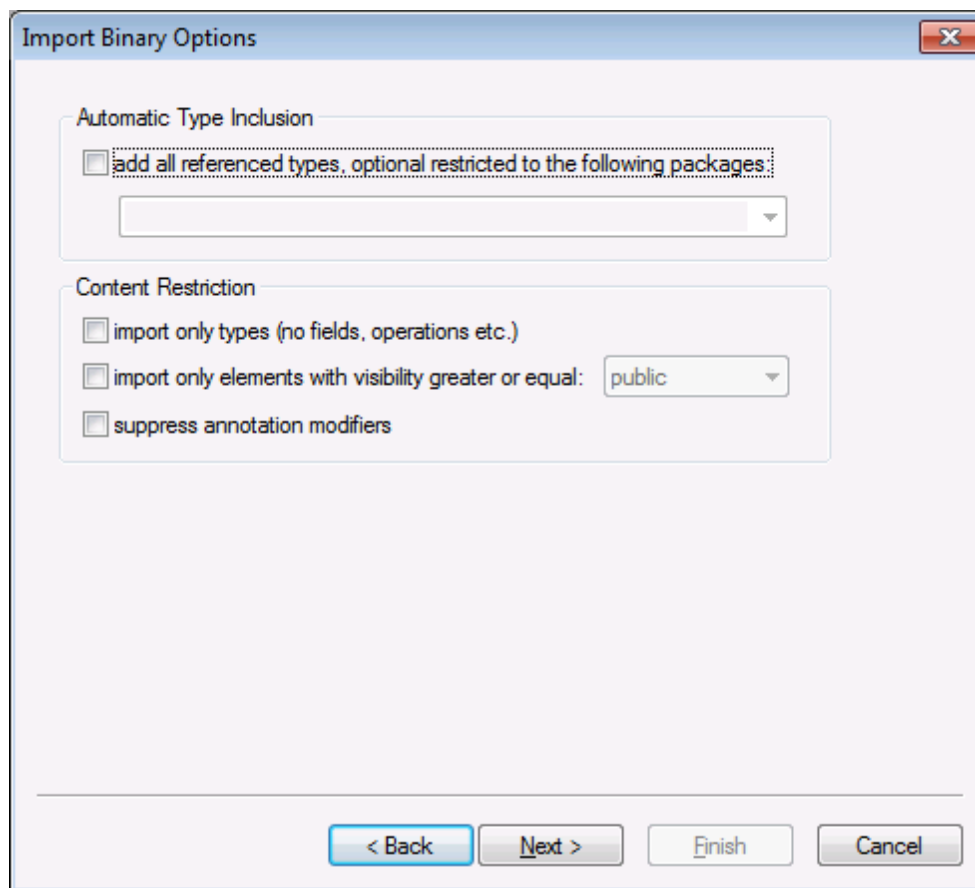
4. Click Next. This opens the Import Binary Selection dialog box.
5. Click **Add** and select the Class Archive from the context menu (in this example, **Class Archives from Java Runtime**).
6. Select a binary (in this example, **rt.jar**).



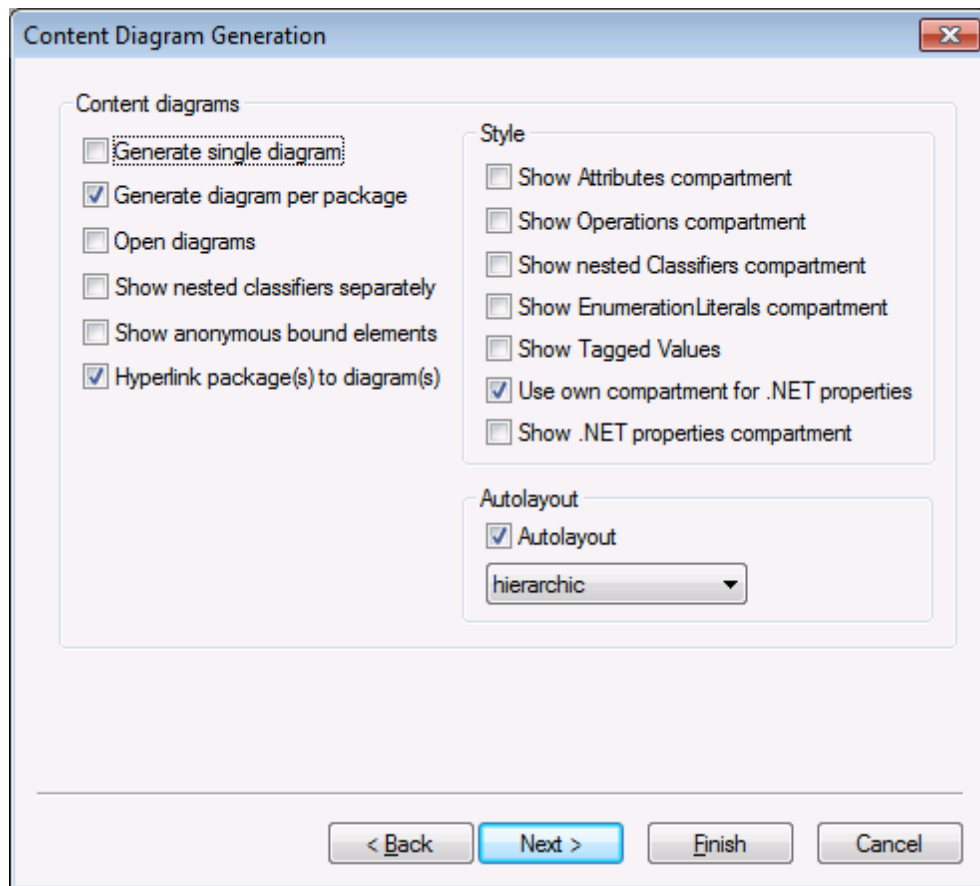
7. Click the "+" expand button to expand the list of binaries, and activate the check box(es) of those that you want to import.



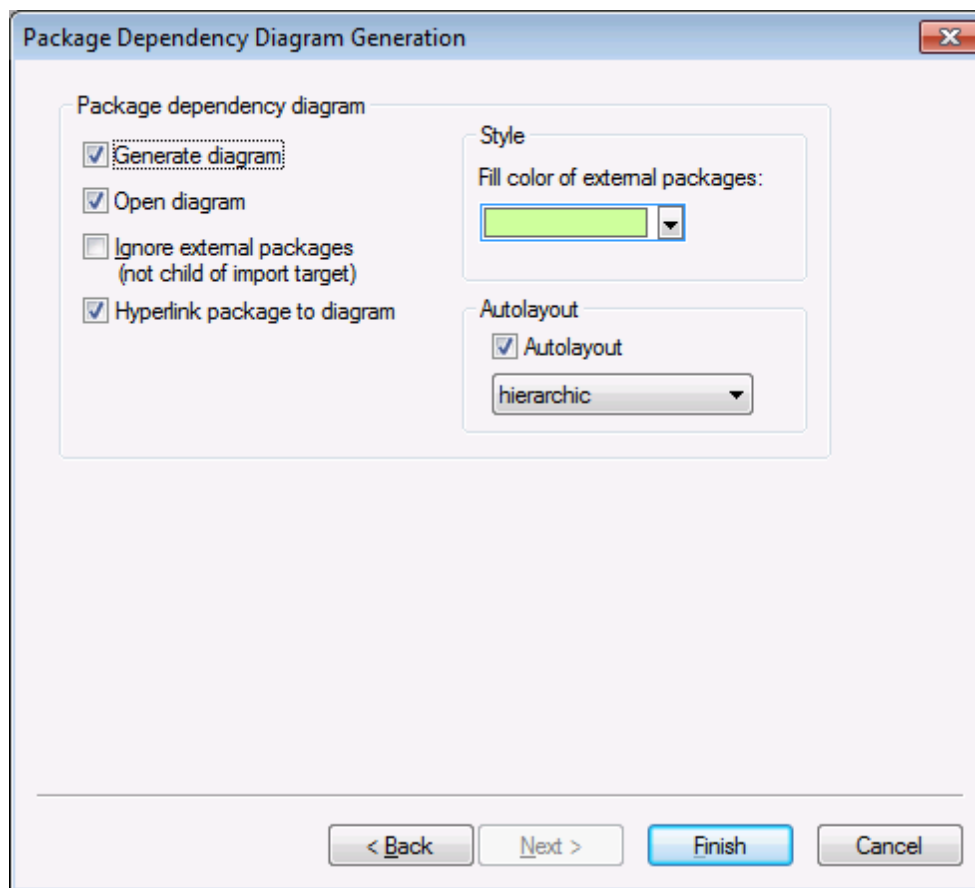
8. Click Next. This opens the **Import Binary Options** dialog box.



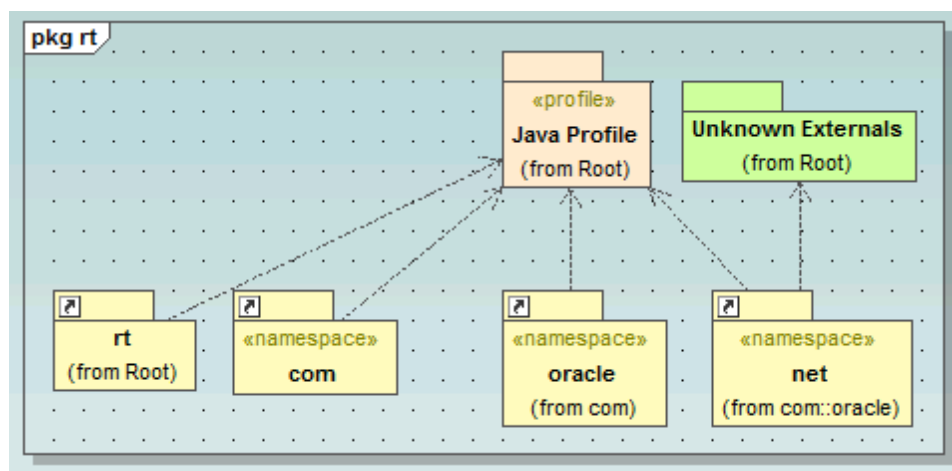
9. Click **Next**.
10. If prompted, define the Import Target, or select the **Import in new Package** check box, then click **Next**.



11. Select the Content Diagram Generation properties from the dialog box and click Next to continue. You can generate a diagram for each package, as well as a single overview diagram.



12. Select the Package Dependency options that you would like to include and click Finish to complete the import procedure. The screenshot below shows the diagram containing the package dependencies of the Java binaries.



Note: Clicking the link icon of a folder automatically opens the referenced diagram.

6.4 Synchronizing the Model and Source Code

You can synchronize the model and code in either direction, and at different levels (for example, project, package or class).

When UModel (Enterprise or Professional) runs as an Eclipse or Visual Studio plug-in, synchronization between model and code takes place automatically. Manual synchronization is possible at the project level; the option to update individual classes or packages is not available. For more information, see [UModel Plug-in for Visual Studio](#) and [UModel Plug-in for Eclipse](#).

When you right-click an element in the Model Tree (for example, a class), the context menu displays the code synchronization or merging commands under the **Code Engineering** menu item:

- **Merge Program Code from UModel *****
- **Merge UModel *** from Program Code**

*** is a Project, Package, Component, Class, and so on, depending on your current selection.

Depending on the settings you have defined from **Project | Synchronization Settings**, the alternative name of these two commands may be:

- **Overwrite Program Code from UModel *****
- **Overwrite UModel *** from Program Code**

To update the entire project (but not classes, packages, or other local elements), you can also use the following commands on the **Project** menu of UModel:

- **Merge (or Overwrite) Program Code from UModel Project**
- **Merge (or Overwrite) UModel Project from Program Code**

For convenience, any of the commands listed above will be generically referred to as "code synchronization commands" further in this topic.

To synchronize at the project or Root package level, do one of the following:

- Right-click the Root package in the Model Tree, and select the required code synchronization command.
- On the **Project** menu, click the required code synchronization command.

To synchronize at package level:

1. Use **Shift**, or **Ctrl + Click** to select the package(s) you want to merge.
2. Right-click the selection, and select the required code synchronization command.

To synchronize at class level:

1. Use **Shift**, or **Ctrl + Click** to select the classes(s) you want to merge.
2. Right-click the selection, and click the required code synchronization command.

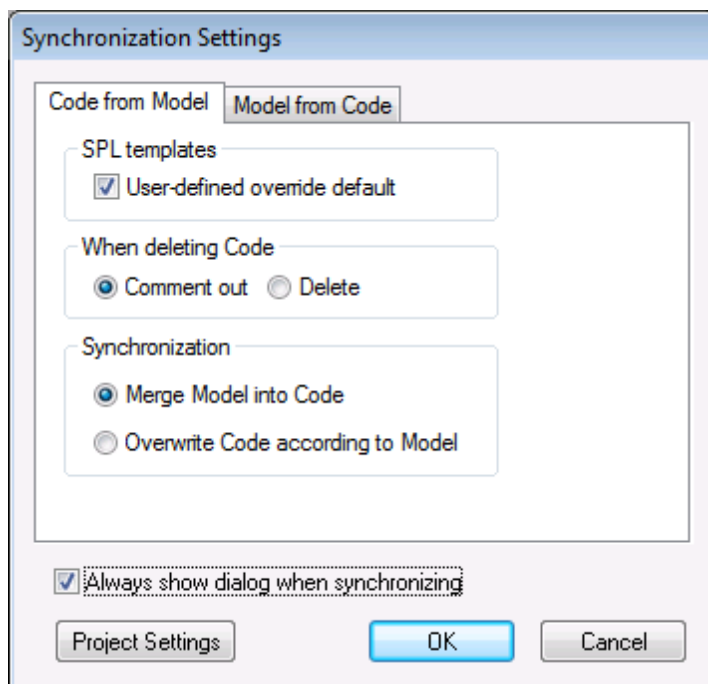
To avoid undesired results when synchronizing the model and code, consider the following scenarios:

On the Project menu, click Overwrite UModel Project from Program Code .	• This checks all directories (project files) of all different code languages you have defined in your project. • New files are identified and added to the project. • An entry "Collecting source files in (...)" appears in the Messages window.
Right-click a class or interface in the Model Tree and select Code Engineering Overwrite UModel Class from Program Code .	• This updates only the selected class (interface) of your project. • However, if the source code contains classes that are new or modified classes since the last synchronization, those changes will not be added to the model.
Right-click a Component in the Model Tree (within the Component View package) and select Code Engineering Overwrite UModel Component from Program Code .	• This updates the corresponding directory (or project file) only. • New files in the directory (project file) are identified and added to the project. • An entry "Collecting source files in (...)" appears in the Message window.

Note: When synchronizing code, you might be prompted to update your UModel project before synchronization. This occurs when you open UModel projects created before the latest release. Click **Yes** to update your project to the latest release format, and save your project file. The notification message will not occur once this has been done.

Synchronization settings

You can change the synchronization options using the menu command **Project | Synchronization options**.



SPL Templates

SPL templates are used during the generation of Java, C# and VB.NET code. SPL templates are only used/accessed when **new** code is generated, i.e. new classes, operations etc. have been added to the model. Existing code does not access/use the SPL templates.

To modify the provided SPL templates:

1. Locate the provided SPL templates in the UModel installation directory ("Program Files"): ...\\UModel2016\\UModelSPL\\Java\\Default (or ...\\C#\\Default, ...\\VB\\Default.)
2. Copy the SPL files you want to edit/modify into the **parent** directory, i.e. ...\\UModel2016\\UModelSPL\\Java\\.
3. Make your changes and save them there.

To use the user-defined SPL templates:

1. Select the menu option **Project | Synchronization settings**.
2. Select the **User-defined override default** check box in the SPL templates group.

6.4.1 Synchronization Tips

Renaming of classifiers and reverse engineering

The process described below applies to the standalone application as well as to the plug-in versions (Visual Studio or Eclipse) when reverse engineering or automatic synchronization takes place.

Renaming a classifier in the code window of your programming application causes it to be deleted and re-inserted as new classifier in the **Model Tree**.

The new classifier is only re-inserted in those modeling diagrams that are automatically created during the reverse-engineering process, or when generating a diagram using the **Show in new Diagram | Content** option. The new classifier is inserted at a default position on the diagram, that will likely differ from the previous location.

See also [Refactoring code and synchronization](#).

Automatic generation of ComponentRealizations

UModel is capable of automatically generating ComponentRealizations during the code engineering process. ComponentRealizations are only generated where it is absolutely clear to which component a class should be assigned:

- Only one Visual Studio project file exists in the .ump project.
- Multiple Visual Studio projects exist but their classes are completely separate in the model.

To enable automatic generation of ComponentRealizations:

1. Open the menu item **Tool | Options**.
2. Click the **Code Engineering** tab and activate the **Generate missing ComponentRealizations** option.

Automatic ComponentRealizations are created for a **Classifier** that can be assigned one (and only one) Component

- without any ComponentRealizations, or
- contained in a code language namespace.

The way the Component is found differs for the two cases.

Component representing a code project file (property "**projectfile**" set)

- if there is ONE Component having/realizing classifiers in the containing package
- if there is ONE Component having/realizing classifiers in a subpackage of the containing package (top down)
- if there is ONE Component having/realizing classifiers in one of the parent packages (bottom up)
- if there is ONE Component having/realizing classifiers in a subpackage of one of the parent packages (top down)

Component representing a directory (property "**directory**" set)

- if there is ONE Component having/realizing classifiers in the containing package
- if there is ONE Component having/realizing classifiers in one of the parent packages (bottom up)

Notes:

- The option "**Code Engineering | Generate missing ComponentRealizations**" has to be set.
- As soon as ONE viable Component is found during one of the above steps, this Component is used and the remaining steps are ignored.

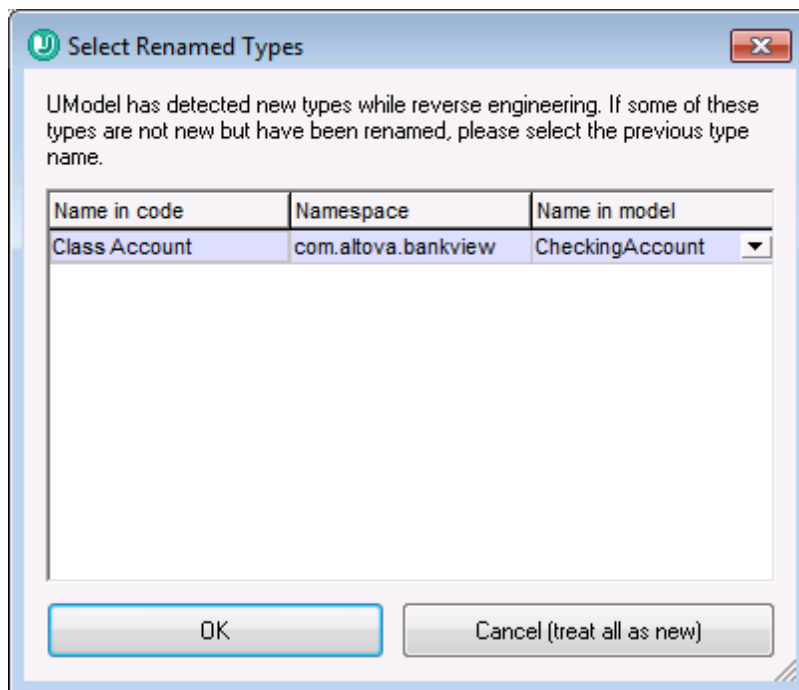
Error/Warnings:

- If no viable Component was found, a warning is generated (message log)
- If more than one viable Component was found, an error is generated (message log)

6.4.2 Refactoring Code and Synchronization

When refactoring code, it is often the case that class names are changed/updated. Versions of UModel prior to version 2009 deleted the "old" classes and inserted new ones, during the code/model synchronization process.

UModel 2009 or later displays a dialog box if it detects that new types have been added/renamed during reverse engineering. The new types are listed in the "Name in code" column while the assumed original type name is listed in the "Name in model" column. UModel attempts to determine the original name by relying on namespace, class content, base classes and other data.



If a class was renamed, select the previous class name using the combo box in the "Name in model" column, e.g. C1. This ensures that all related data are retained and the code engineering process remains accurate.

Changing class names in the model and regenerating code

Having created a model and generated code from it, it is possible that you might want to make changes to the model again before going through the synchronization process.

E.g. You decide that you want to change the class names before generating code the second time round. As you previously assigned a file name to each class, in the "code file name" field of the Properties window, the new class and file name would now be out of sync.

UModel prompts if you want the code file name to agree with the new class name, when you start the synchronization process. Note that you also have the option to change the class constructors as well.

Round-trip engineering and relationships between modeling elements

When updating model from code, associations between modeling elements are automatically displayed, if the option **Diagram Editing | Automatically create Associations** has been activated in the **Tools | Options** dialog box. Associations are displayed for those elements where the attributes type is set, and the referenced "type" modeling element is in the same diagram.

InterfaceRealizations as well as Generalizations are all automatically shown in the diagram when updating model from code.

6.5 Forward engineering prerequisites

Minimum conditions needed to produce code for forward engineering:

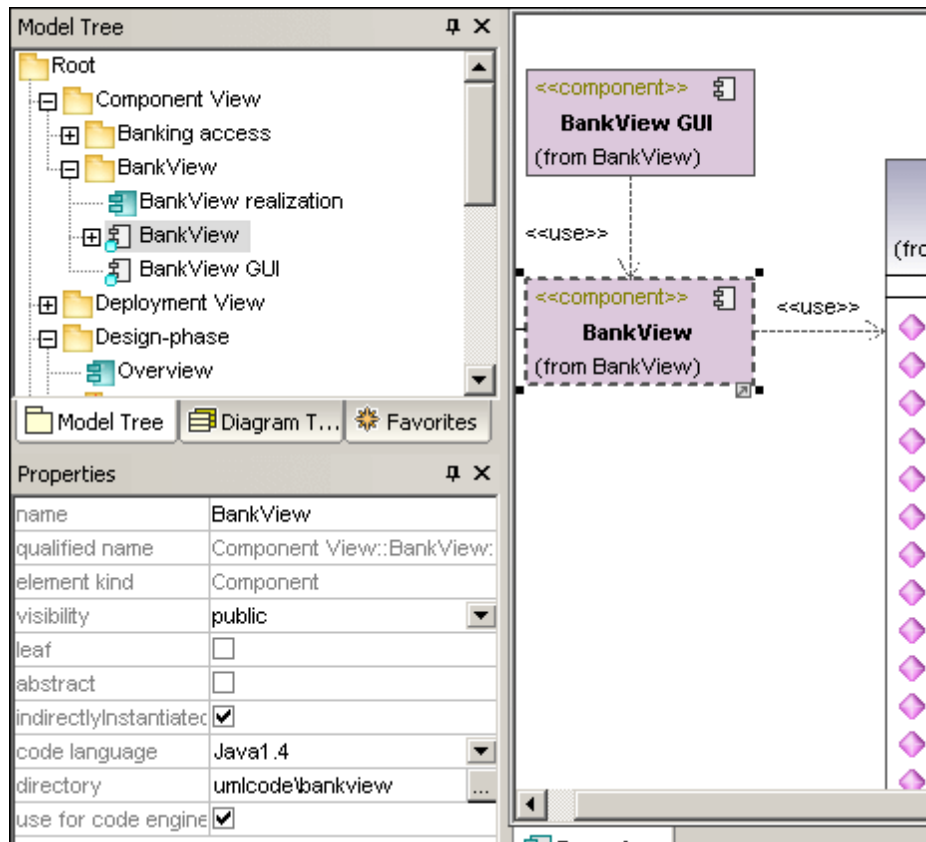
- A component must be **realized** by one or more **classes**, or **interfaces**.
- The component must have a **physical location**, i.e. directory, assigned to it. The generated code is then placed in this directory.
- Components must be individually set to be **included** in the code engineering process.
- The Java, C#, or VB namespace root package must be defined.

To create a component realization:

1. Drag the class, or interface onto the respective component in the Model Tree view.
You can also create a realization in a component diagram using the Realization icon.

To assign a physical location:

1. Select the component in the Model Tree, or in the diagram.
2. Click the Browse button  of the **directory** property and select a directory (or enter it directly).



To include components in the code engineering process:

1. Select the component in the Model Tree, or in the diagram.
2. Activate the "use for code engineering" check box.

To define the Java namespace root:

1. Right clicking a package and selecting "Set as Java namespace root" sets the Java namespace root.

This means that this package and all sub packages, are enabled during the code engineering process. The Java namespace root is denoted with a  icon in the Model Tree pane.

- Selecting the command again **removes** the Java namespace for this package.

6.6 Java code to/from UModel elements

The table below shows the one-to-one correspondence between:

- UModel elements and Java code elements, when outputting model to code
- Java code elements and UModel model elements, when inputting code into model

The screenshot shows a small section of the table. Please click [HERE](#) to open the HTML version of the table in your browser.

Java <=> UModel				
Java			UModel	
Project	projectfile		projectfile	Component
	directory		directory	
Package	name		name	Package <<namespace>>
	name		name	
	modifiers	package		
		public		
		protected	visibility	
		private		
		abstract		
		final	abstract	
	filename	strictfp	isFinalSpecification	
			<<strictfp>>	
			code file name	
		associated projectfile/directory	ComponentRealization	
		extends clause	Generalization	
		implements clause	InterfaceRealization(s)	
	java docs		Comment(->Documentation)	

6.7 C# code to/from UModel elements

The table below shows the one-to-one correspondence between:

- UModel elements and C# code elements, when outputting model to code
- C# code elements and UModel model elements, when inputting code into model

The screenshot shows a small section of the table. Please click [HERE](#) to open the HTML version of the table in your browser.

C# < > UModel			
C#		UModel	
Project	projectfile	projectfile	Component
	directory	directory	
Namespace	name	name	Package <<namespace>>
	name	name	
	modifiers	internal	package protected <<internal>> public protected private sealed isFinalSpecification abstract <<static>> <<unsafe>> <<partial>> <<new>>
		protected internal	
		public	
		protected	
		private	
		sealed	
		abstract	
		static	
		unsafe	
		partial	
		new	
	filename	code file name	
	associated projectfile/directory	ComponentRealization	
	base types	Generalization, InterfaceRealization(s)	
	attribute sections	<<attributes>>	
	doc comments	Comment(->Documentation)	

6.8 XML Schema to/from UModel elements

The table below shows the one-to-one correspondence between:

- UModel elements and XML Schema elements, when outputting model to code
- XML Schema elements and UModel model elements, when inputting code into model

 XSD/UML Element

 Stereotype property (=tagged value)

The screenshot shows a small section of the table. Please click [HERE](#) to open the HTML version of the table in your browser.

XSD <=> UModel					
XSD			UModel		
file path			project file	Component	
	target namespace		name	Package <<namespace>>	
	attributeFormDefault		attributeFormDefault		
	blockDefault		blockDefault		
	elementFormDefault		elementFormDefault		
	finalDefault		finalDefault		
	version		version		
	xml:lang		xml:lang		
	xmlns		xmlns		
	annotation	source			source
		appinfo			Comment <<appinfo>>
		documentation	xml:lang	xml:lang	Comment <<documentation>>

6.9 VB.NET code to/from UModel elements

The table below shows the one-to-one correspondence between:

- UModel elements and VB.NET code elements, when outputting model to code
- VB.NET code elements and UModel model elements, when inputting code into model

The screenshot shows a small section of the table. Please click [HERE](#) to open the HTML version of the table in your browser.

VB.NET <=> UModel				
VB.NET			UModel	
Project	projectfile		projectfile	Component
	directory		directory	
Namespace	name		name	Package <<namespace>>
	name		name	
	modifiers	Friend	visibility	package
		Protected Friend		protected <<Friend>>
		Public		public
		Protected		protected
		Private		private
		NotInheritable	isFinalSpecification	
		MustInherit	abstract	
		Partial	<<Partial>>	
		Shadows	<<Shadows>>	
	filename		code file name	
	associated projectfile/directory		ComponentRealization	
	base types		Generalization, InterfaceRealization(s)	
	attribute sections		<<Attributes>>	
	doc comments		Comment(->Documentation)	

6.10 Database to/from UModel elements

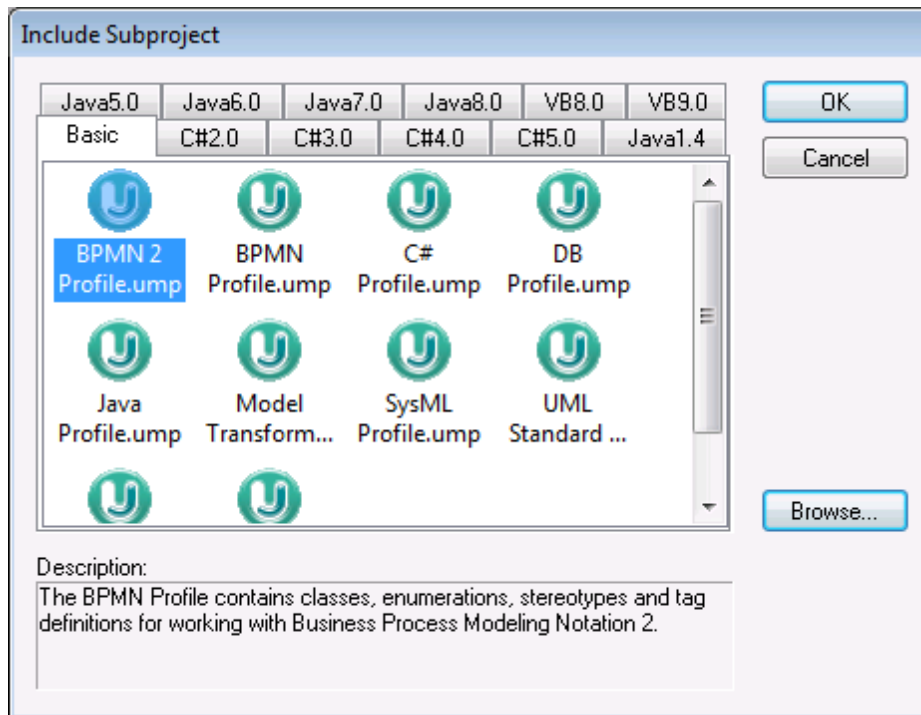
The table below shows the one-to-one correspondence between:

- UModel elements and database elements, when outputting model to code
- Database elements and UModel model elements, when inputting code into model

		Database		<>	Umodel						
Database		connection		UModel					Component		
Database	Schema	name		name					Package <<namespace>> <<Schema>>	Package <<namespace>> <<Database>>	
		Table	Column	name		name		Property			Class <<Table>>
				Data Type		type					
				Not Null		<<not_null>>					
				Null		<<nullable>>					
				Length		Multiplicity					
				Precision							
				Scale							
				Default							
				Autoincrement		<<autoincrement>>					
				Part of Primary Key		<<PK>>					
			Part of Foreign Key		<<FK>>						
			Part of Unique Key		<<unique>>						
			Primary Key	name		name		Property			Class <<PrimaryKey>>
				Column	name	name					
			Foreign Key	name		name		Property			Class <<ForeignKey>>
				Column	name	name					
			Foreign Column	name		name		Property			Class <<UniqueKey>>
				foreign table	type		name				
			Unique Key	name		name		Property			Class <<Index>>
				Column	name	name					
			Index	name		name		Property			Class <<CheckConstraint>>
				Column	name	order: ascending order: descending					
		CheckConstraint	name		name						
			definition		definition						
		View	name		name		Property	Class <<View>>			
			definition		definition						
			Column	name		name					
				Data Type		type					
				Not Null		<<not_null>>					
				Null		<<nullable>>					
				Length		Multiplicity					
				Precision							
				Scale							
				Default							
		Autoincrement		<<autoincrement>>							
		Stored Procedure	name		name		Operation <<StoredProcedure>>	Class <<StoredProcedures>>			
			definition		definition						
			Parameter	name		name					
				direction mode		direction					
		Function	name		name		Operation <<Function>>	Class <<Functions>>			
			definition		definition						
			Parameter	name		name					
				direction mode		direction					
		Trigger	name		name			Class <<Trigger>>			
			definition		definition						

6.11 Including other UModel projects

To include a UModel project as a subproject of another UModel project, select the menu command **Project | Include Subproject**.



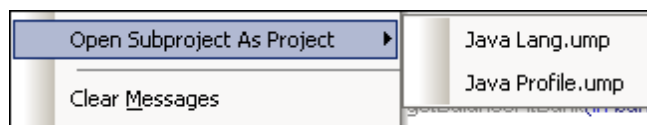
Include Subproject dialog box

The tabs and UModel projects (.ump files) available on the "Include Subproject" dialog box are configurable. Namely, UModel reads this information from the following path relative to the "Program Files" folder on your operating system: **AltovaUModel2016UModelInclude**. Note that the project files available on the **Basic** tab exist directly under the **UModelInclude** folder, while projects in each of the Java, VB, and C# tabs exist as subfolders of the **UModelInclude** folder.

By default, UModel is supplied with several project profiles (.ump files) that can be useful in your UModel project. For example, Java lang classes, interfaces and packages are available on the Java tabs, organized by Java version. Likewise, .ump files for the Microsoft C# and VB types are available on separate tabs organized by version.

To view all currently imported projects:

- Select the menu option **Project | Open Subproject Individually**. The context menu displays the currently included subprojects.



To create a custom tab on the "Include Subproject" dialog box:

- Navigate to the **\Altova\UModel2016\UModelInclude** folder (relative to your "Program Files"), and create your custom folder in it, for example **\UModelInclude\myfolder**. The name you give to the folder determines the name of the tab on the "Include Subproject" dialog box.
- Copy to your custom folder any .ump files that you want to make available on the corresponding tab.

To create descriptive text for each UModel project file:

- Create a text file using the same name as the *.ump file and place in the same folder. For example, the **MyModel.ump** file requires a descriptive file called **MyModel.txt**. Please make sure that the encoding of this text file is UTF-8.

To remove an included project:

1. Click the included package in the Model Tree view and press the **Delete** key.
2. You are prompted if you want to continue the deletion process.
3. Click OK to delete the included file from the project.

To delete or remove a project from the "Include Subproject" dialog box:

- Delete or remove the (MyModel).ump file from the respective folder.

6.12 Merging UModel projects

It is now possible to perform a 2-way, or 3-way, project merge in UModel. Both merge different UModel project files into a common UModel *.ump model.

This option is useful if multiple persons are working on the same project at the same time, or you just want to consolidate your work into one model.

Please see:

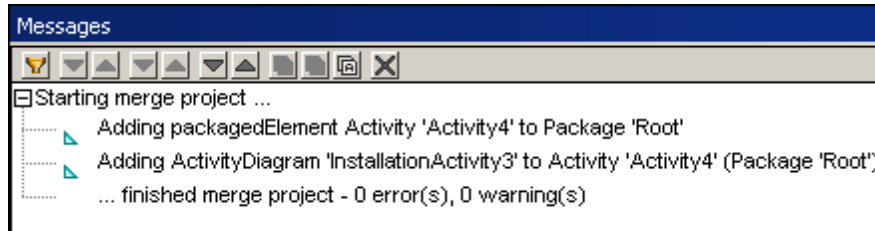
[2-way Project merge](#)

[3-way Project merge](#)

6.12.1 2-way Project merge

To merge two UML projects:

1. Open the UML file that is to be the target of the merge process, i.e. the file into which the second model will be merged - the merged project file.
2. Select the menu option **Project | Merge Project...**
3. Select the **second** UML project that is to be merged into the first one.
The message window reports on the merge process, and logs the relevant details.



Please note:

Clicking on one of the entries in the message window displays that modeling element in the Model Tree.

Merging results:

- **New** modeling elements i.e. those that do not exist in the source, are added to the merged model.
- **Differences** in the same modeling elements; the elements from the **second** model take precedence, e.g. there can only be one default value of an attribute, the default value of the second file is used.
- **Diagram** differences: UModel first checks to see if there are differences between diagrams of the two models. If there are, then the new/different diagram is added to the merged model (with a running number suffix, activity1 etc.) and the original diagram is retained. If there are no differences, then identical diagrams(s) are ignored, and nothing is changed. You can then decide which of the diagrams you want to keep or delete, you can of course keep both of them if you want.
- The whole merge process can be **undone** step-by-step by clicking the Undo icon in the icon bar, or pressing **Ctrl+Z**.
- Clicking an entry in the message window displays that element in the Model Tree.
- The **file name** of the merged file (the first file you opened) is retained.

6.12.2 3-way Project merge

UModel now supports the merging of multiple UModel projects that have been simultaneously edited by different developers, in a 3-way project merge.

Project merging

The 3-way project merge works with **top-level** UModel projects, i.e. main projects that may contain subprojects, it does not support individual **file** merging, when these files have unresolved references to other files.

When merging **main** projects, any editable subprojects are automatically merged as well. There is no need for a separate subproject merging process.

Please see [Manual 3-way project merge example](#) for a simple example.

- The whole merge process can be **undone** step-by-step by clicking the **Undo** toolbar button, or pressing **Ctrl+Z**.
- Clicking an entry in the message window displays that element in the Model Tree.
- The **file name** of the merged file, the first file you opened, is retained!

Merging results:

Note: **source** in the following text, means the initial/first project file you open before starting the merge process.

- **New** modeling elements in the second file i.e. that do not exist in the source, are added to the merged model.
- New modeling elements in the source file i.e. that do not exist in the second file, remain in the merged model.
- **Deleted** modeling elements from the second file i.e. those that still exist in the source, are removed from the merged model.
- Deleted modeling elements from the source file i.e. that still exist in the second file, remain deleted from the merged model.

Differences to the **same** modeling elements:

- If a property (e.g. the visibility of a class) is changed in either the source, **or** second file, the updated value is used in the merged model
- If a property (e.g. the visibility of a class) is changed in both source **and** second file, the value of the second file is used (and a warning is shown in the messages window)

Moved elements:

- If an element is moved in the source, **or** second file, then the element is moved in the merged model
- If an element is moved (to different parents) in **both** the source and second file, a prompt appears, and you have to manually select the parent element in the merged model

Diagram differences:

UModel first checks to see if there are differences between diagrams of the two models.

If there are, then the new/different diagram is added to the merged model (with a running number suffix, activity1 etc.) and the original diagram is retained.

If there are no differences, then identical diagrams(s) are ignored, and nothing is changed.

You can then decide which of the diagrams you want to keep or delete, you can of course keep both of them if you want.

Source control systems support for 3-way merging

When checking in/out project files, UModel automatically generates "Common ancestor" (or snapshot) files which are then used for the 3-way merge process. This enables a much finer merge result than the normal 2-way merge.

The specific source control system you use, determines if the **automatic** snapshot 3-way merge process is supported by UModel. A manual 3-way merge is however, always possible.

- Source control systems that perform automatic file merging without user intervention, will probably not support an automatic 3-way merge.
- Source control systems that prompt you to choose between Replace or Merge, when a project file has been changed, will generally support a 3-way merge. After the source control plug-in has replaced the file, selecting the Replace command activates the UModel file alert which then allows you to do a 3-way merge. UModel must be used for the check in/out process.
- Main projects as well as subprojects can be placed under source control. Changing data in a subproject automatically prompts you if the subproject(s) should be checked out.
- Each check in/out action, creates a Common ancestor, or a snapshot, file which is then used during the 3-way project merge process.

Please note:

Snapshot files are only automatically created and used with the standalone versions of UModel, i.e. these functions are not available in the Eclipse or Visual Studio plug-in versions.

E.g.

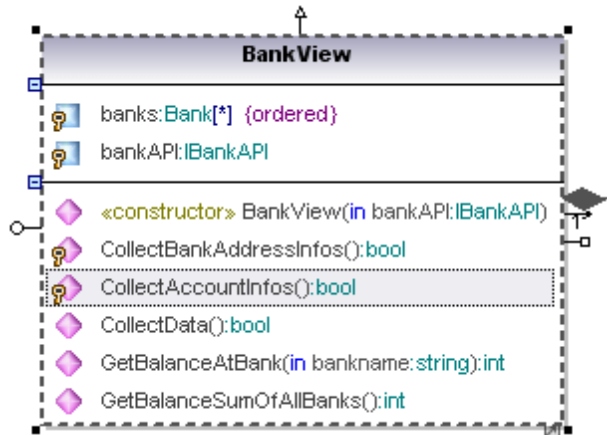
User A edits a UModel project file and changes the name of a class in the BankView Main diagram. User B opens the same project file and changes the visibility of the same class.

As snapshot files are created for each user, the snapshot editing history allows the individual changes to be merged into the project. Both the name and visibility changes are merged into the project file during the 3-way merge process.

6.12.3 Manual 3-way project merge example

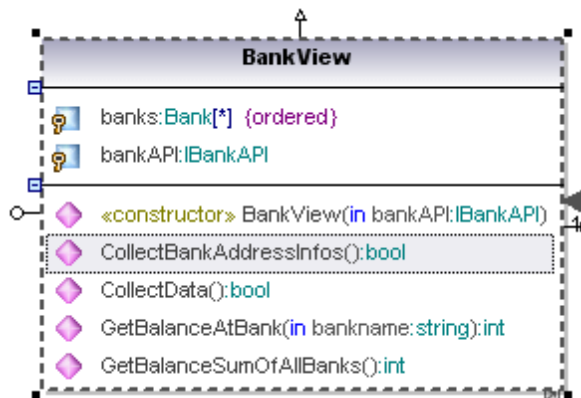
This simple example uses the **Bank_CSharp.ump** file available in the ...\\UModelExamples folder. Two other instances of the same project are copied to child folders below the UModelExamples folder, i.e. \\C#_1 and \\C#_2.

User1 opens the **Bank_CSharp.ump** project file in folder **C#_1** and makes changes to the **BankView** class.



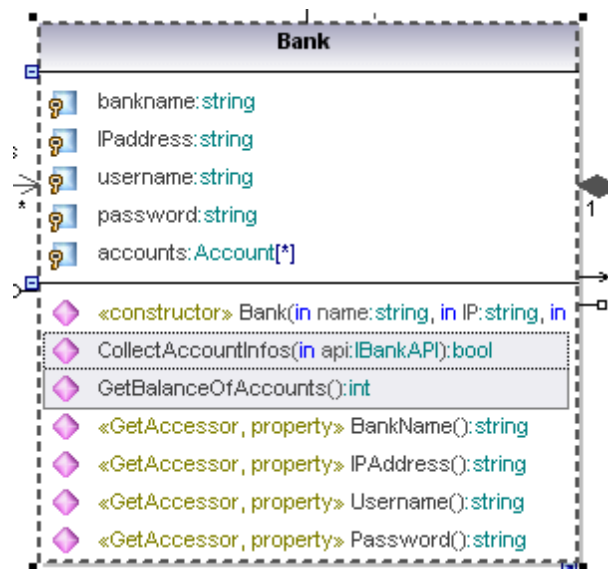
Changes made to the BankView class:

1. Operation **CollectAccountInfos():bool** is **deleted** from the BankView class.
2. The **visibility** of the **CollectBankAddressInfos():bool** operation is changed from "protected" to "public".



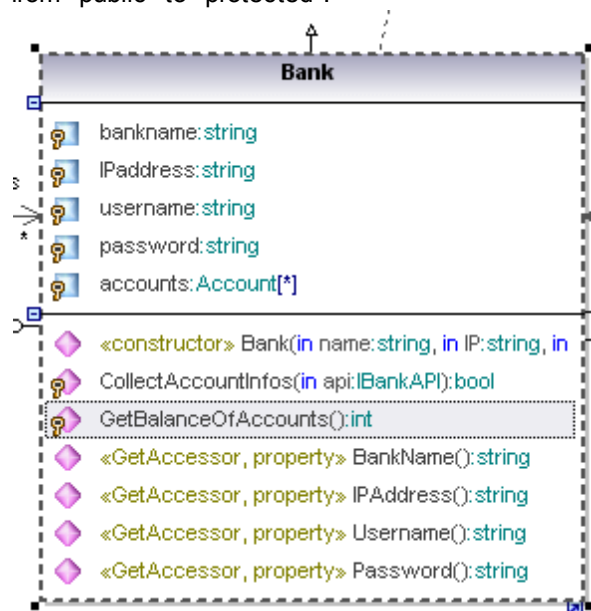
3. The project is then saved.

User2 opens the **Bank_CSharp.ump** project file in folder **C#_2** and makes changes to the **Bank** class.



Changes made to the Bank class:

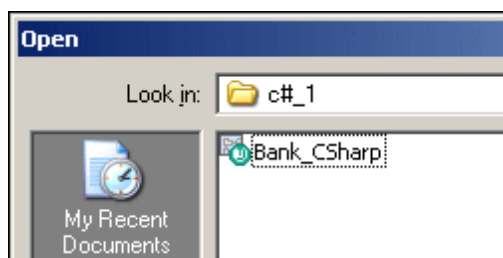
1. The operations **CollectAccountInfos...** and **GetBalanceOfAccounts...** are both changed from "public" to "protected".



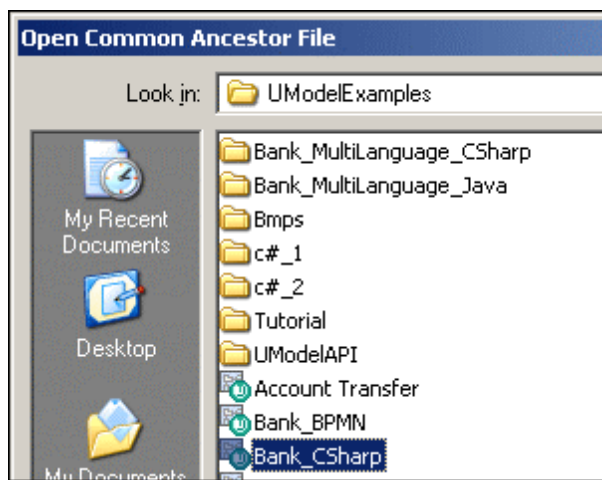
2. The project is then saved.

Umr2 now starts a 3-way project merge:

1. Select **Project | Merge Project (3-way)**.
The Open file dialog box appears.
2. Select the project file changed by **User1** in the ...\\C#_1 folder.

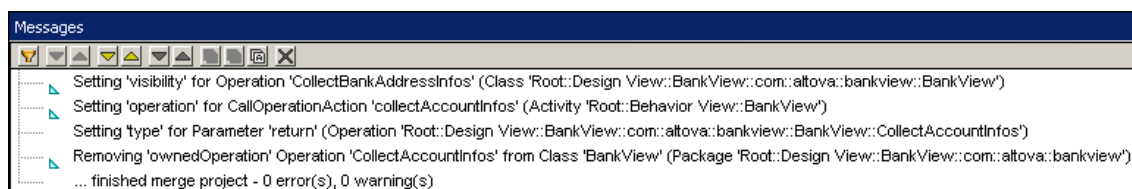


You are now prompted to open the **Common Ancestor file**, which is the original project file in the ...\\UModelExamples folder.



The 3-way merge process is started and you return to the project file from which you started the 3-way merge process, i.e. from the project file in the C#_2 folder.

The Messages window shows you the merge process in detail.



- The changes made to the project in the C#_1 folder, are replicated in the C#_2 project file.
- The changes made to the project in the C#_2 folder, are retained in the project file.
- The project file in the C#_2 folder should now be used as the Common Ancestor file for future 3-way merges between the project files in folders C#_1 and C#_2.

6.13 Sharing Packages and Diagrams

UModel allows you to share packages and UML diagrams they might contain, between different projects. Packages can be included in other UModel projects by reference, or as a copy.

Also note that subproject files can be split off a main, or subproject, file at any time. The subproject files can be included as editable or read-only from the main project; each package is shared and saved as a subproject file. Subprojects can be added to a source control system. Please see [Teamwork support for UModel projects](#) for more information.

Shared package prerequisites:

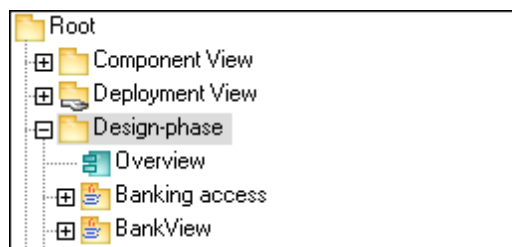
- Links to other packages outside of the shared scope are not permissible.

Note:

When creating UModel project files, do not use one project file as a "template/copy" for another project file into which you intend to share a package. This will cause conflicts due to the fact that every element should be globally unique (see [uuid](#)) and this will not be the case, as two projects will have elements that have identical uuids.

To share a package between projects:

1. Right click a package in the Model Tree tab and select **Subproject | Share package**.



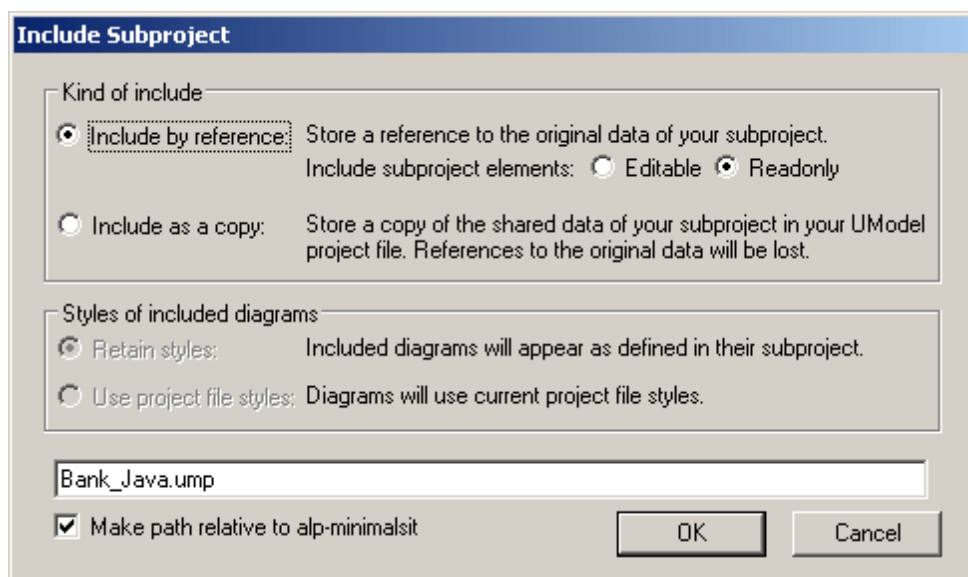
A "shared" icon appears below the shared package in the Model Tree. This package can now be included in any other UModel project.

To include/import a shared folder in a project:

1. Open the project which should contain the shared package (an empty project in this example).

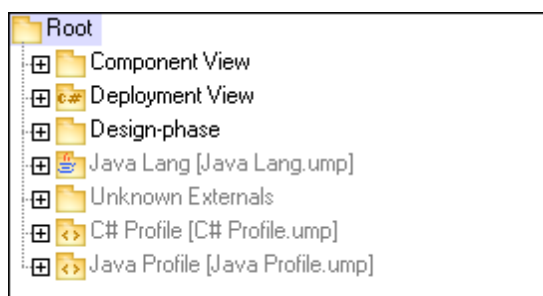


2. Select the menu item **Project | Include Subproject...**
3. Click the Browse button, select the project that contains the shared package and click Open.



The "Include" dialog box allows you to choose between including the package/project by reference, or as a copy.

4. Select the specific option (Include by reference) and click OK.



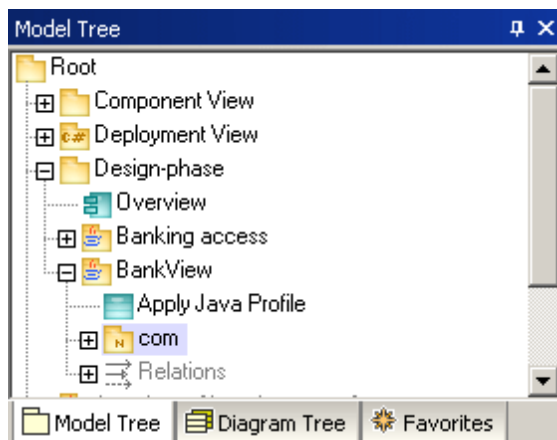
The "Deployment View" package is now visible in the new package. The packages' source project is displayed in parenthesis (BankView-start.ump).

Shared folders that have been included by reference can be changed to "Include by copy" at any time, by right clicking the folder and selecting **Subproject | Include as a Copy**.

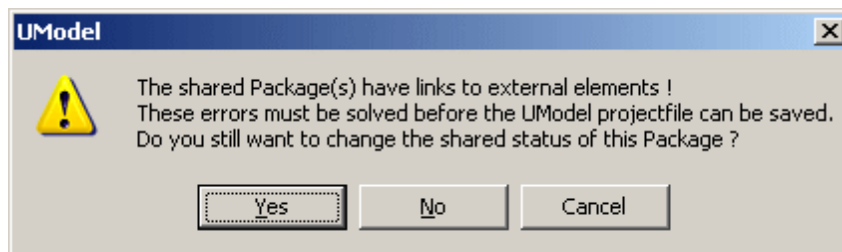
Please note:

All included projects of the source project, have also been included: Java Lang, Unknown Externals and Java Profile.

Shared packages - links to external elements:

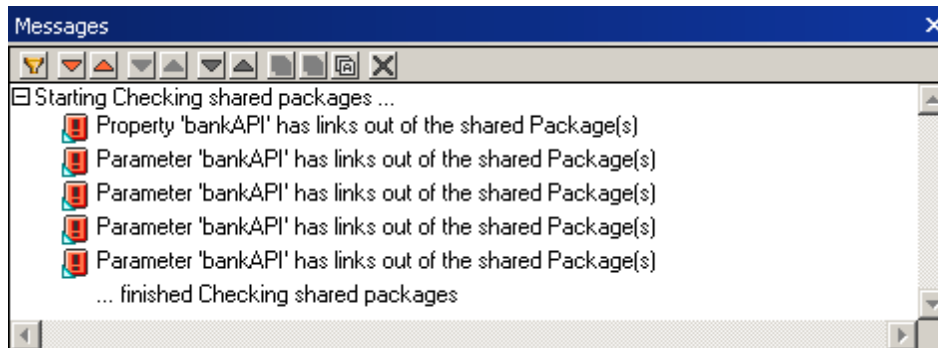


Attempting to share a package which has links to external elements causes a prompt to appear. E.g. trying to share the BankView package.

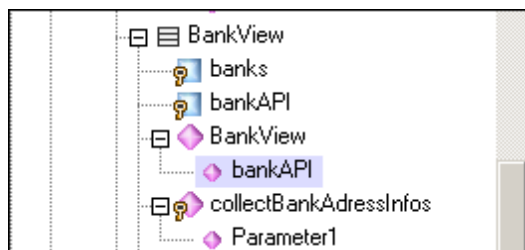


Clicking Yes, forces you to resolve the external links before you can save.

The Messages pane provides information on each of the external links.



Clicking an error entry, in the Messages pane, displays the relevant element in the Model Tree tab.

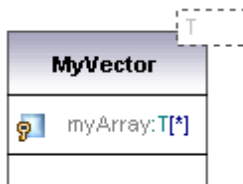


6.14 UML templates

UModel supports the use of UML templates and their mapping to or from Java, C# and Visual Basic generics.

- Templates are "potential" model elements with unbound formal parameters.
- These parameterized model elements, describe a group of model elements of a particular type: classifiers, or operations.
- Templates cannot be used directly as types, the parameters have to be bound.
- Instantiate means binding the template parameters to actual values.
- Actual values for parameters are expressions.
- The binding between a template and model element, produces a new model element (a bound element) based on the template.
- If multiple constraining classifiers exist in C#, then the template parameters can be directly edited in the Properties tab, when the template parameter is selected.

Template **signature** display in UModel:



- Class template called **MyVector**, with formal template parameter "**T**", visible in the dashed rectangle.
- Formal parameters without type info (T) are implicitly classifiers: Class, Datatype, Enumeration, PrimitiveType, Interface. All other parameter types must be shown explicitly e.g. Integer.
- Property **myArray** with unbounded number of elements of type T.

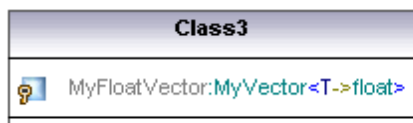
Right clicking the template and selecting **Show | Bound elements**, displays the actual bound elements.

Template **binding** display:



- A bound named template **intvector**
- Template of type, **MyVector**, where
- Parameter **T** is substituted/replaced by **int**.
- "**Substituted by**" is shown by **->**.

Template **use** in properties/operations:



An anonymous template binding:

- Property MyFloatVector of type **MyVector<T->float**

Templates can also be defined when defining properties or operations. The autocomplete function helps you with the correct syntax when doing this.



- Operation1 returns a vector of floats.

6.14.1 Template signatures

A Template signature is a string that specifies the formal template parameters. A template is a parameterized element that is used to generate new model elements by substituting/binding the formal parameters to actual parameters (values).

Formal template parameter

T

Template with a single untyped formal parameter
(stores elements of type T)

Multiple formal template parameters

KeyType:DateType, ValueType

Parameter substitution

T>aBaseClass

The parameter substitution must be of type "aBaseClass", or derived from it.

Default values for template parameters

T=aDefaultValue

Substituting classifiers

T>{contract}aBaseClass

allowsSubstitutable is true

Parameter must be a classifier that may be substituted for the classifier designated by the classifier name.

Constraining template parameters

T:Interface>anInterface

When constraining to anything other than a class, (interface, data type), the constraint is displayed after the colon ":" character. E.g. T is constrained to an interface (T:Interface) which must be of type "anInterface" (>anInterface).

Using wildcards in template signatures

T>vector<T->?<aBaseClass>

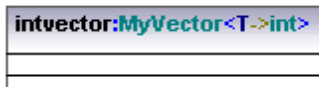
Template parameter T must be of type "vector" which contains objects which are a supertype of aBaseClass.

Extending template parameters

T>Comparable<T->T>

6.14.2 Template binding

Template binding involves the substitution of the formal parameters by actual values, i.e. the template is instantiated. UModel automatically generates anonymously bound classes, when this binding occurs. Bindings can be defined in the class name field as shown below.

**Substituting/binding formal parameters**

vector <T->int>

Create bindings using the class name

a_float_vector:vector<T->float>

Binding multiple templates simultaneously

Class5:vector<T->int, map<KeyType->int, ValueType<T->int>

Using wildcards ? as parameters (Java 5.0)

vector<T->?>

Constraining wildcards - upper bounds (UModel extension)

vector<T->?>aBaseClass>

Constraining wildcards - lower bounds (UModel extension)

vector<T->?<aDerivedClass>

6.14.3 Template usage in operations and properties

Operation returning a bound template

```
Class1  
Operation1():vector<T->int>
```

Parameter T is bound to "int". Operation1 returns a vector of ints.

Class containing a template operation

```
Class1  
Operation1<T>(in T):T
```

Using wildcards

```
Class1  
Property1:vector<T->?>
```

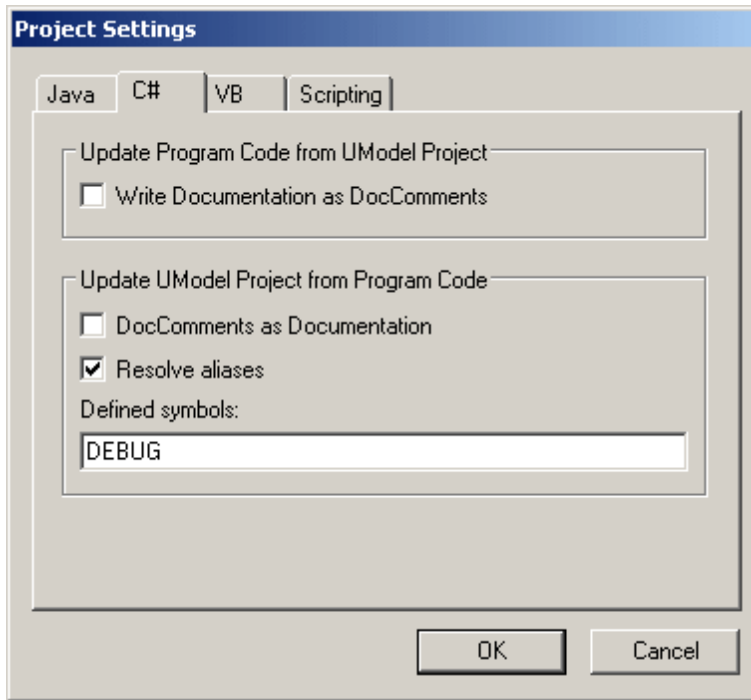
This class contains a generic vector of unspecified type (? is the wildcard).

Typed properties can be displayed as associations:

- Right click a property and select **Show | PropertyX as Association**, or
- Drag a property onto the diagram background.

6.15 Project Settings

This option allows you to define the global project settings.



Select the menu item **Tools | Options** to define your local settings, please see **Tools | [Options](#)** in the Reference section for more details on the local settings.

6.16 Enhancing performance

Due to the fact that some modeling projects can become quite large, there are a few ways you can enhance the modeling performance:

- Make sure that you are using the latest driver for your specific graphics card (resolve this before addressing the following tips)
- Disable syntax coloring - **Styles tab | Use Syntax Coloring = false.**
- Disable "gradient" as a background color for diagrams, use a solid color. E.g. **Styles tab | Diagram background color | White.**
- The automatically enabled autocompletion can be deactivated using **Tools | Options | Diagram Editing** then unchecking the "Enable automatic entry helper" check box.

Chapter 7

Transforming UML models

7 Transforming UML models

You can transform UML modeling elements between all of the programming languages that UModel supports, for example:

Java ⇔ C# ⇔ VB.NET ⇔ XSD ⇔ DB ⇔ UML, or any other combination.

The transformation paradigm extends to updating existing transformations and merging the updates into the specified target models. This means that existing diagrams in the target model are retained even if the target model is extended by functions not available in the source model. The new functions are also retained on future transformation updates.

If a class operation has the UML stereotype <create> applied in UML source model, it will have the stereotype <constructor> applied in the target model (C#, Java, VB).

When the target model is a database, a property named "id" in the source model will be converted to a primary or foreign key of matching data type in the target model.

Note that Model Transformations can be run **before** the forward engineering, and/or **after** the reverse engineering process (see [Automatically update transformation](#)).

All the code relevant element types are transformed between the programming languages, including classes, interfaces, attributes, operations, generalizations, and so on.

UModel is also able to transform class **diagrams** between the following languages:

Java ⇔ C# ⇔ VB.NET ⇔ UML

Two transformation examples are available to show how this works:

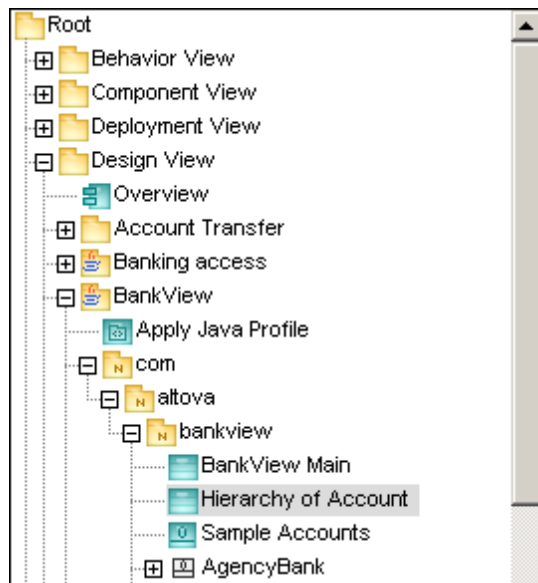
- [Transformation example - Java to C#](#)
- [C# to Java - model to model](#)

7.1 Transformation example - Java to C#

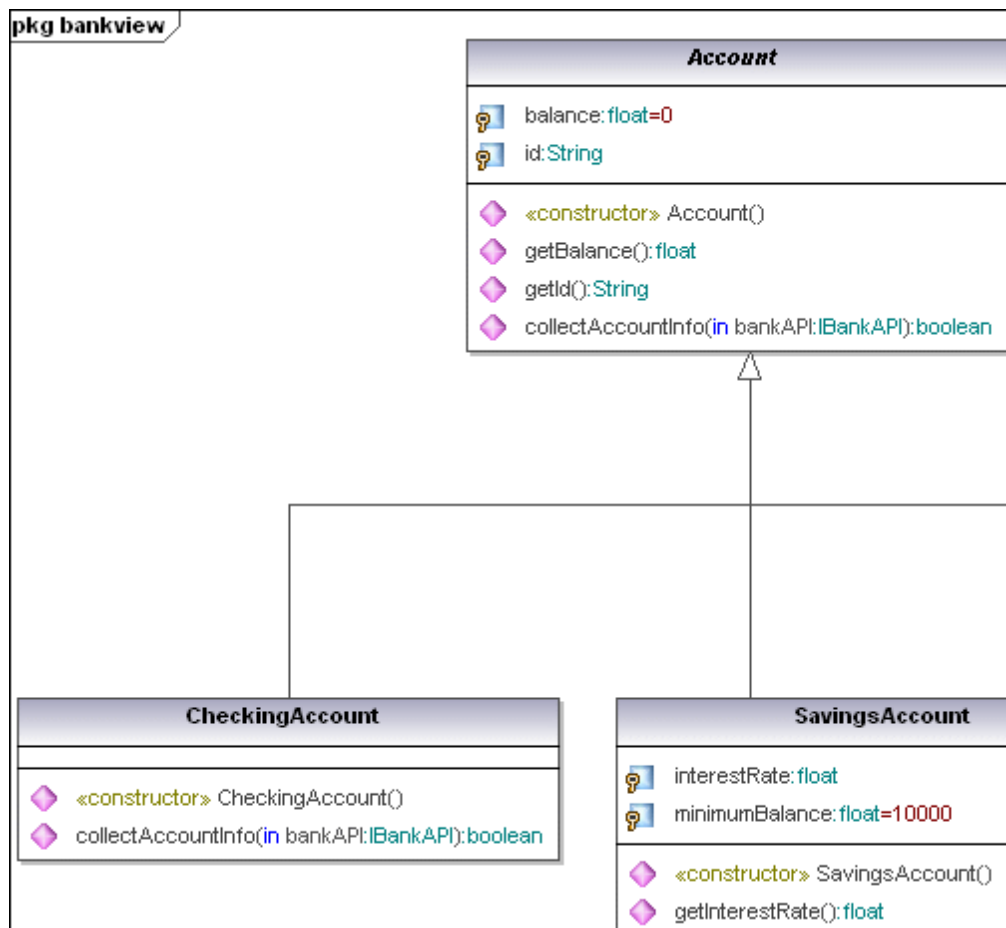
This section shows a simple transformation of a Java package to a C# package.

1. Open the **Bank_Java.ump** example file available in the UModelExamples folder.
2. Click in the Model Tree and press the **Ctrl + F** shortcut keys, and enter "hierarchy" in the "Find what" text box.

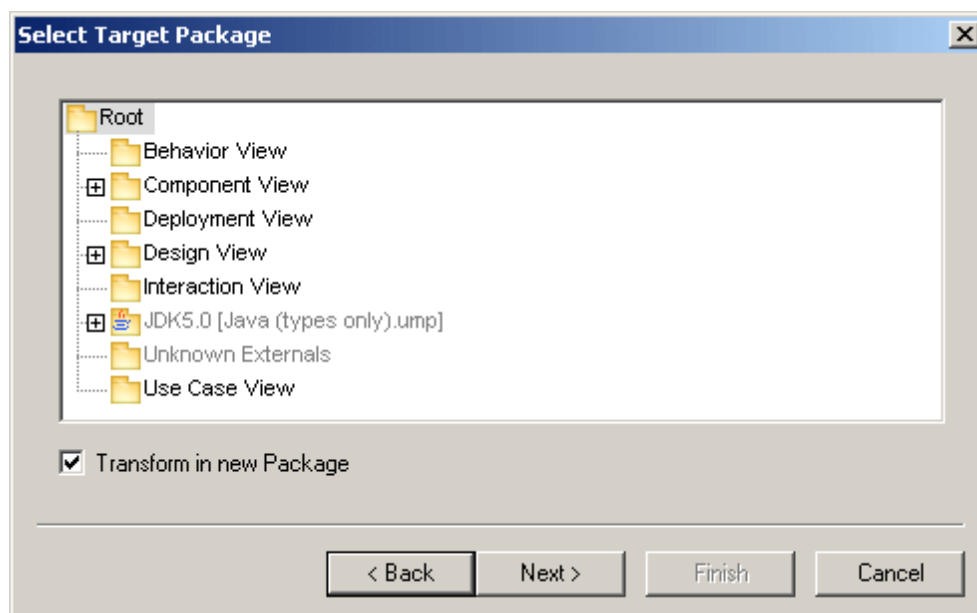
This expands the Model Tree and highlights the Hierarchy of Account diagram.



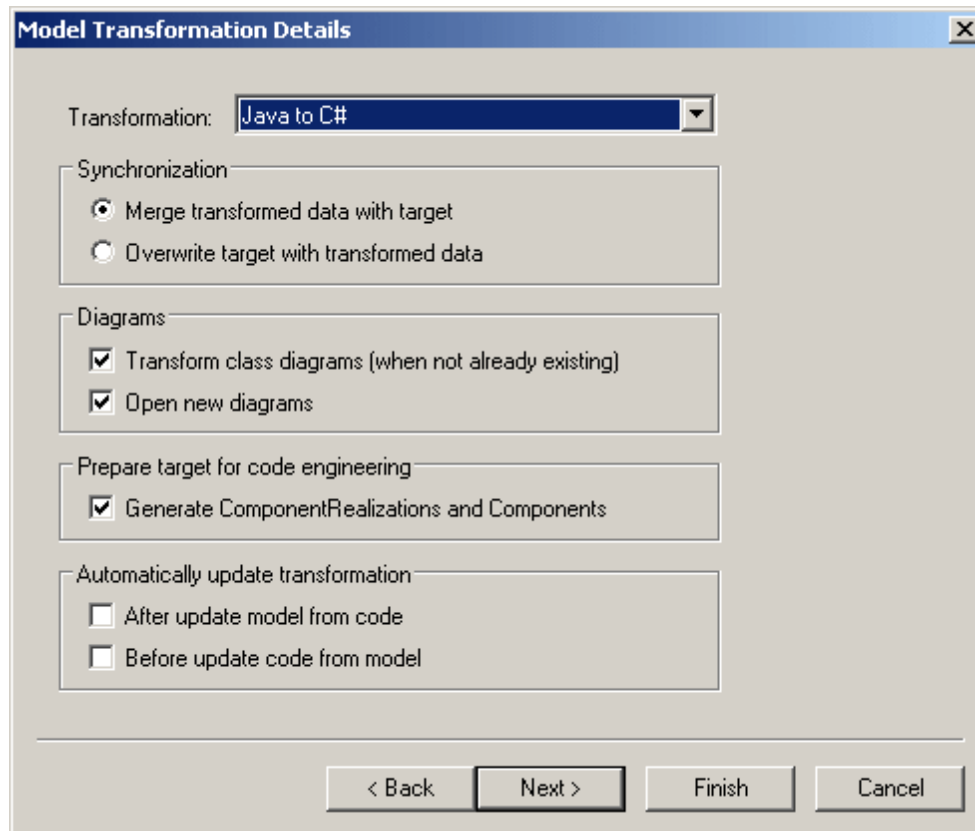
3. Double click the icon to open the Hierarchy of Account diagram.



4. Select the menu option **Project | Model Transformation**.
5. Select the Source Package "bankview" and click Next.
6. In the Select Target Package dialog box, click the "Transform into new Package" check box to add a new package, (or click an existing package to add it there), then click Next.

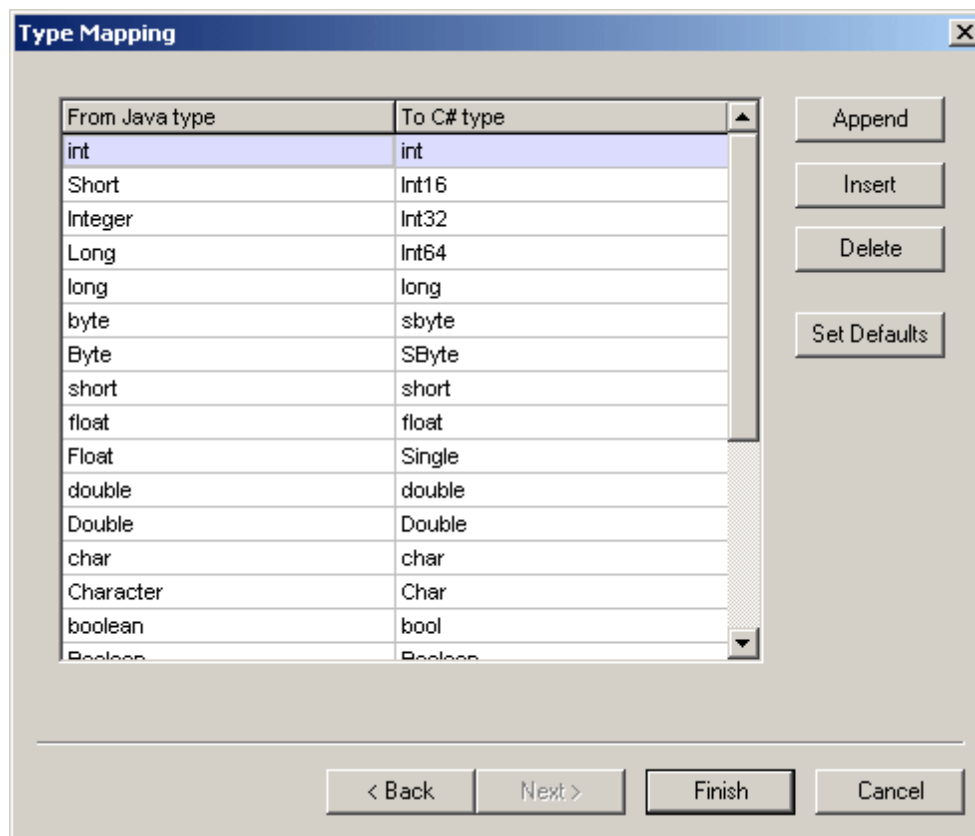


7. Click the "Transformation" combo box and select "Java to C#".

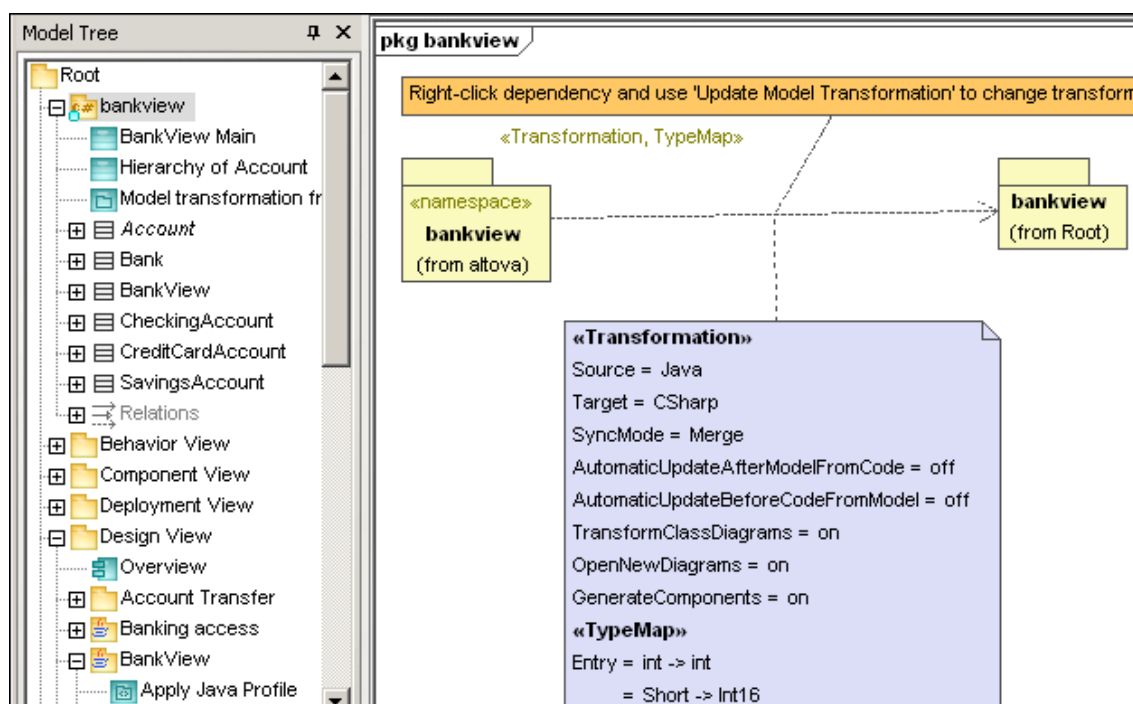


Note: The Automatically update transformation group lets you automatically perform a transformation before or after the code-engineering process takes place, please see [Automatically update transformation](#) for more information.

8. Click Next.
This opens the "Type Mapping" dialog box where you can define/edit code language mappings.

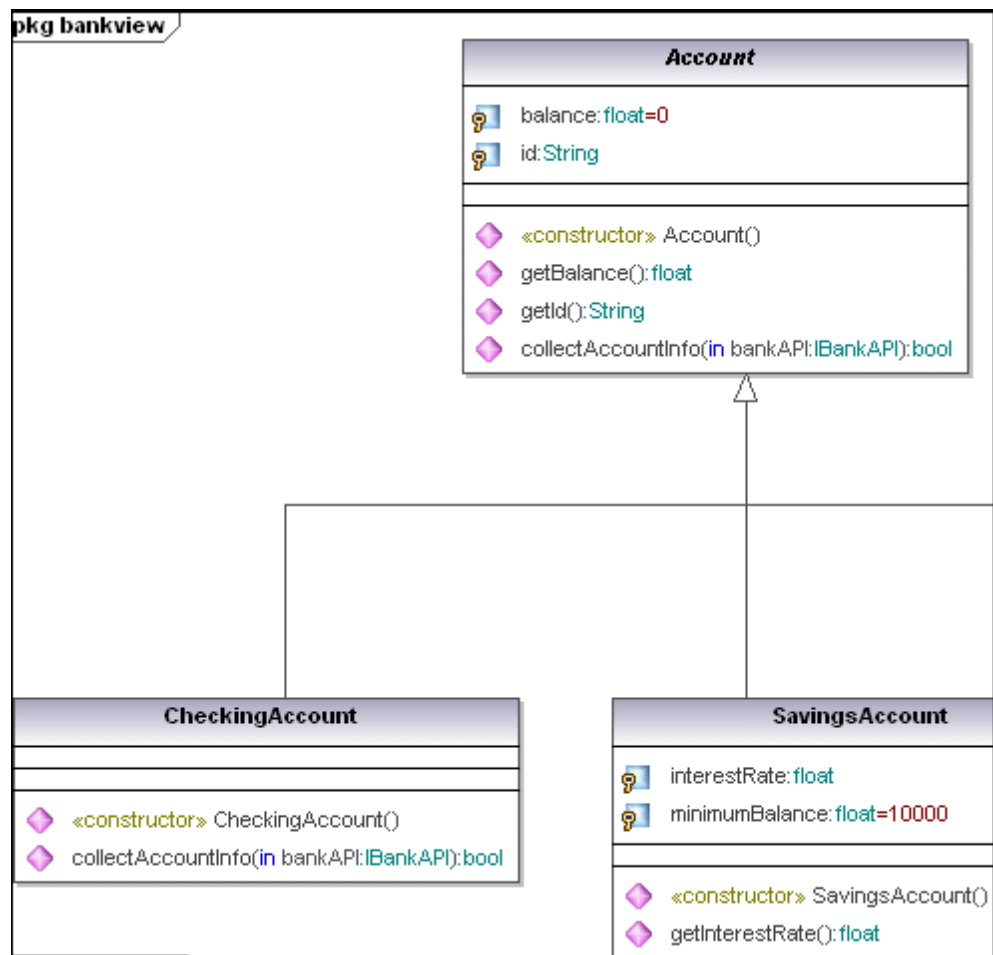


9. Click Finish to use the currently defined default settings.
A message appears stating that the UModel Transformation Profile will be included.
10. Click OK to continue.
This adds a new bankview package to the project and places it directly under the Root package.



The "Model transformation from..." diagram shown above, shows the source and target packages as well as the Transformation settings and code language mappings used.

11. Click the Hierarchy of Account tab (which was opened automatically) to see the newly generated class diagram.



You will notice that the new C# class diagram contains the "bool" type instead of the "boolean" type, used in the Java class diagram.

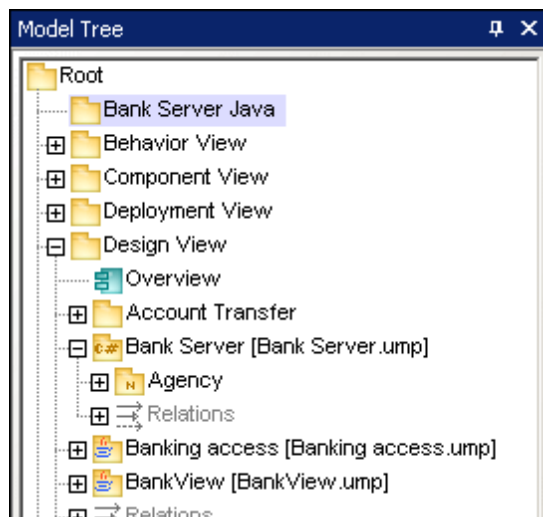
7.2 C# to Java - model to model

This example uses the Bank_multiLanguage.ump file available in the ...UModelExamples folder. The model contains a Banking Access implementation written in Java and a Bank Server written in C#.

The example supposes that the programming requirements have changed and that Bank Server must now be implemented in Java.

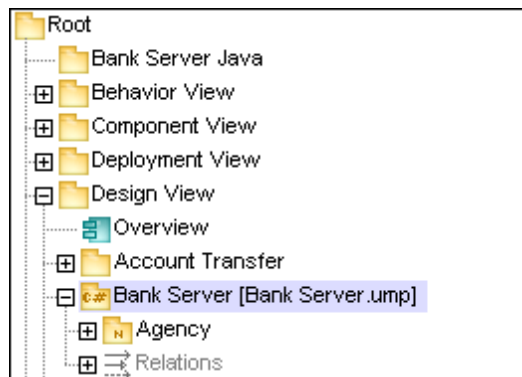
To convert Bank Server C# into Bank Server Java:

1. Open the UModel example file Bank_MultiLanguage.ump.
2. Right click the Root package and select **New element | Package** and enter the package name, e.g. "Bank Server Java".

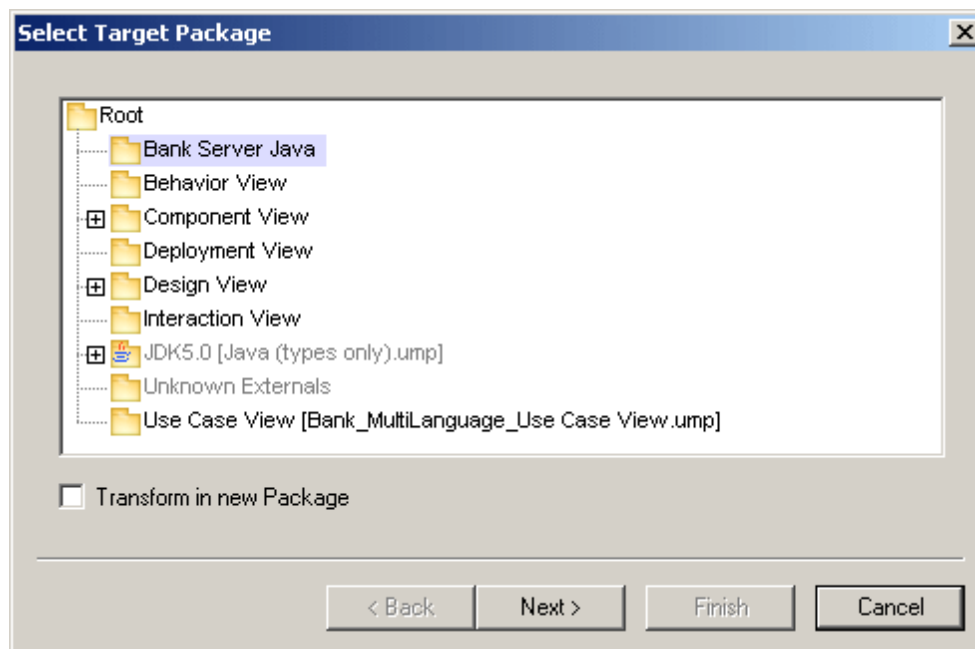


Bank Server Java will become the **target** package for the model transformation.

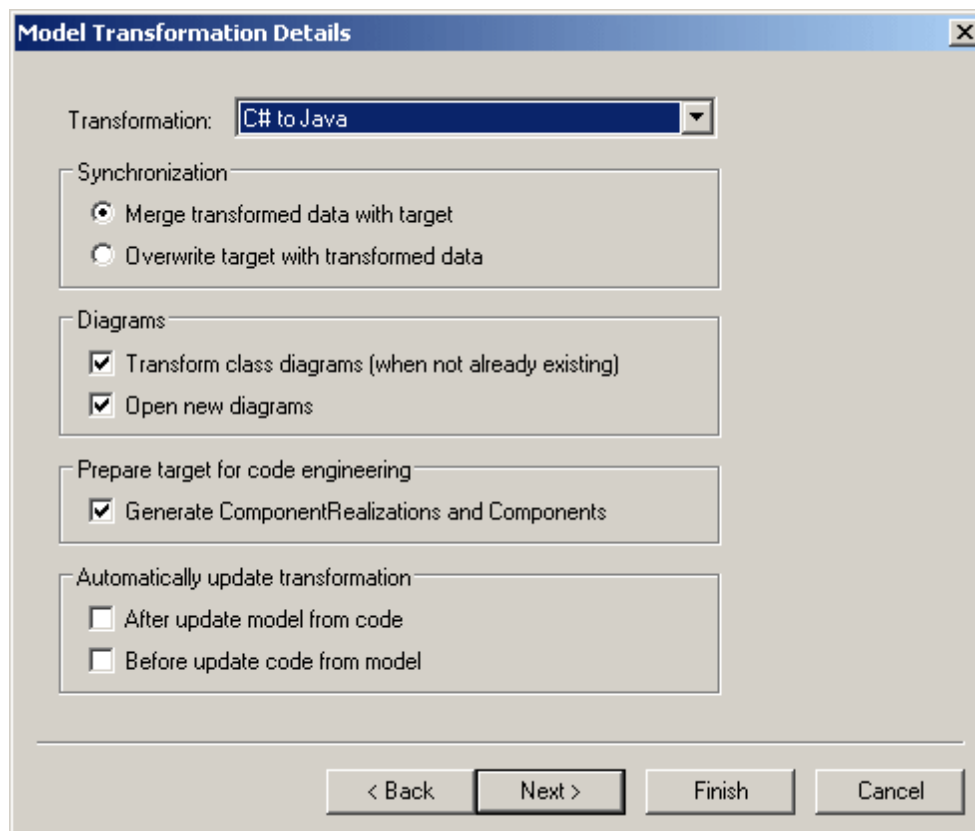
3. Right click the original/source Bank Server [Bank Server.ump] package, and select "Model Transformation" from the context menu.



4. Select the previously created "Bank Server Java" package as the target package, and click the Next button.



5. Select "C# to Java" as transformation type from the Transformation combo box. You can leave all the other options in the dialog box unchanged, then click the Finish button.



6. Click OK in the message box to include the Transformation Profile.

Please Note:

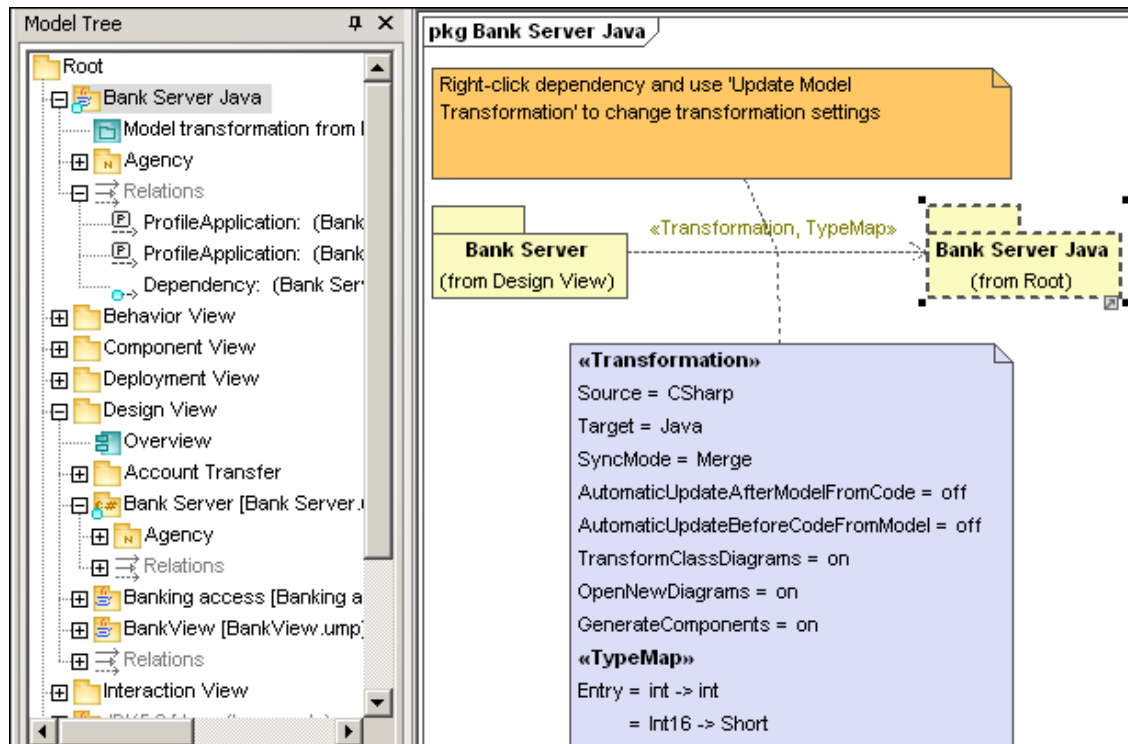
The wizard was completed without pressing the Next button, because we want to use the default Type Settings and clicking Finish automatically uses them.

Transformation result:

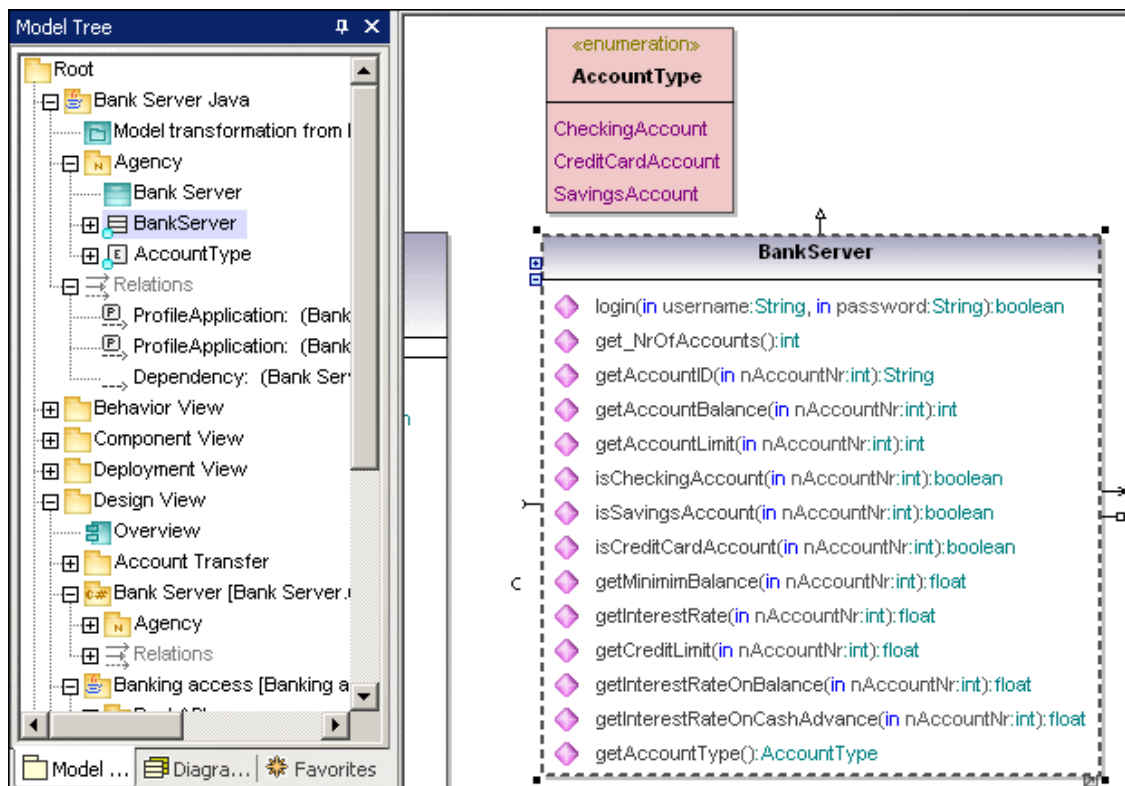
UModel transforms the package, generates two modeling diagrams and transforms the external "types" e.g. integer, Boolean into the corresponding types of the target language.

Two modeling diagrams are generated:

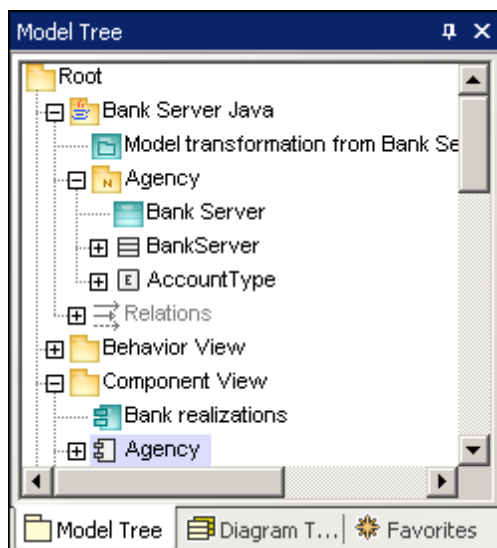
The Model Transformation diagram which shows the packages and their dependencies, as well as a Transformation dependency package diagram which shows the dialog box settings and the language mappings that are used.



The second diagram is a class diagram containing the BankServer class, IBankAPI interface and the AccountType enumeration.

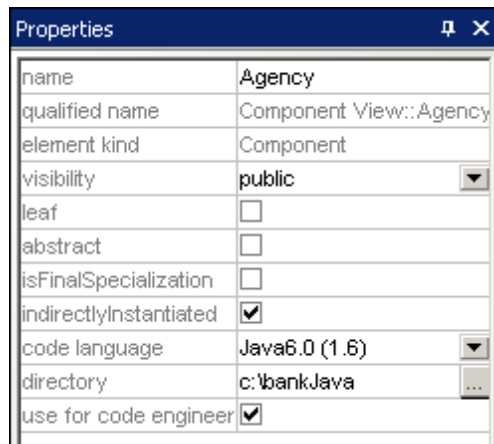


UModel automatically creates a new Component called "Agency" in the Component View package, because BankServer is in the Agency namespace.



To generate source code for the new Java BankServer class:

1. Click the Agency component in the Model Tree and enter the folder name that the generated code is to be placed in, e.g., C:\bankJava.



2. Right click the newly added Bank Server Java package and select **Code Engineering | Merge Program Code from UModel Package**, then click OK to accept the Synchronization settings.

This create two Java source code files, one for the BankServer class and one for the AccountType enumeration. Note that UModel only generates the classes, interfaces, properties and operation declarations for you, the implementation code itself is not transformed, this is naturally done manually.

Modifying the source package and updating the target

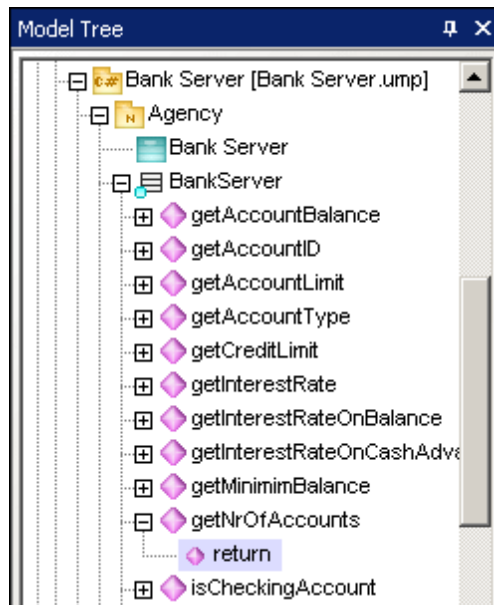
UModel supports the continuous updating of transformed models. This means that you can safely work in the source model (i.e. the C# Bank Server) and use the model transformation to keep the target model (i.e. the Java Bank Server) current with the source model.

Automatically updating the new Java model/package when updates are made to the original C# source

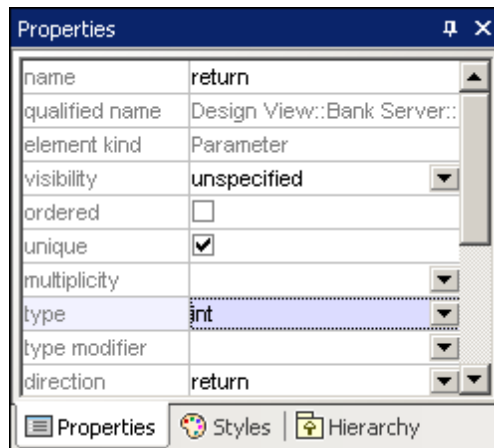
This section assumes that changes have been made to the C# version of Bank Server, that was previously transformed. We would also like these changes to appear in the transformed Java version of Bank Server.

Add a new operation e.g. getNrOfAccounts, with the return type "int", to the C# Bank Server class:

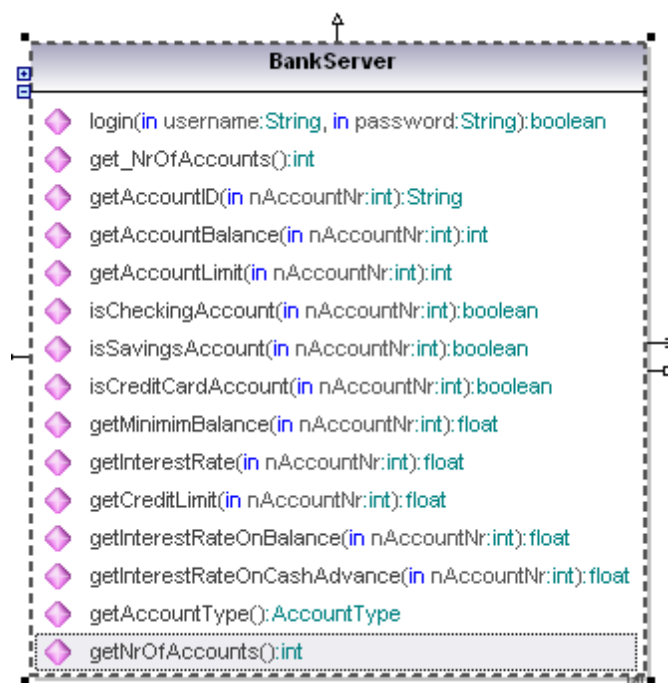
1. Right click the BankServer class, select New element | Operation and enter getNrOfAccounts as the operation name.
2. Right click getNrOfAccounts and select New element | Parameter (return).



3. Click the "return" parameter and enter "int" into the type field of the Properties tab.



4. Right click the Bank Server Java package and select **Update Model Transformation | From 'BankServer' to 'BankServer Java...'**.
5. Click the Finish button in the Model Transformation Details dialog box, to complete the transformation.
The new operation has been inserted into the BankServer Java class and updated your Java target model.



Please note:

You can also modify the target package of the model transformation. You can for example add some methods or some inner classes to the Java Bank Server class, and then re-run the transformation with the 'Merge transformed data with target' (which is the default) setting.

7.3 Transformation settings

The options in this dialog box define the transformation settings from source to target language.

Transformation:

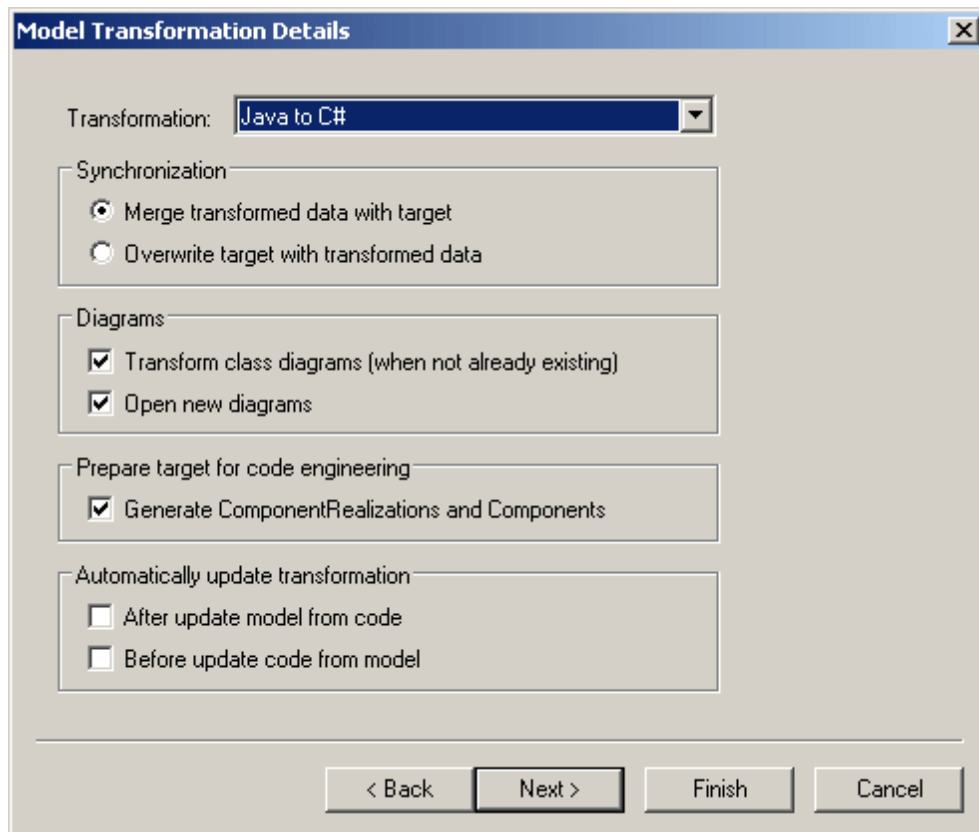
Selects the transformation source and target language. The items available in the combo box depend on the language of source package that you want to transform.

Synchronization:

Defines if you want to merge the transformed data with the target data, or if you want to overwrite the target with the transformed data.

Diagrams:

Defines if you want to generate new class diagrams if they do not exist in the target, and if all new diagrams should be opened in the target package after transformation.



Prepare target for code-engineering:

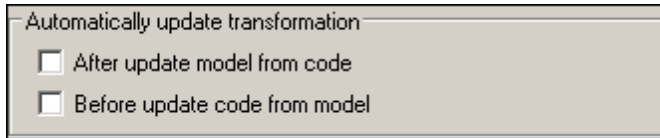
Allows you to define if ComponentRealizations and Components should be automatically generated prior to code-engineering.

Automatically update transformation:

These options let you define if you want to update your transformations automatically before and/or after, the code-engineering process. Please see [Automatically update transformation](#) for more information.

7.4 Automatically update transformation

Note that Model Transformations can also be run **before** the forward engineering, and/or **after** the reverse engineering process. This is achieved by activating the check boxes in the "Automatically update transformation" group.



After "Update Model from Code" - active

E.g.

- Add a new **operation** in the Java source **code**
- Start the reverse engineering process, which adds the new operation to the Java model (via the reverse engineering process)
- This then automatically adds the same operation to the C# model (because After "Update Model from Code" is active).

Before " Update Code from Model" - active

E.g

- Add a new **property** to the Java **model** and
- Start the forward-engineering process
- The property is automatically added to the C# model (because "Update Code from Model" is active).
- The property is also written to the source code; Java - one attribute, C# - attribute and operation, for the same reason.

These two options help keep everything up-to-date automatically, without you having to run transformations manually.

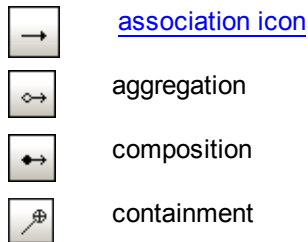
Chapter 8

Creating model relationships

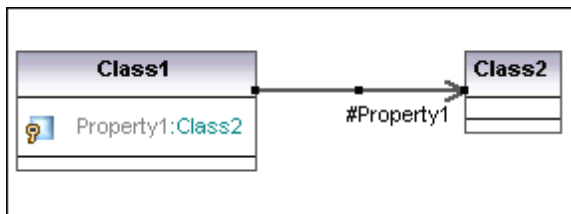
8 Creating model relationships

Model relationships can be created and inserted into diagrams using several methods:

- By using the connection handles, please see [Use cases](#) for an example.
- By clicking the specific icon in the icon bar, and using drag-and-drop to create the connections between the elements:



When an association has been created, a new attribute is automatically inserted in the originating (A:name) class, e.g. Property1:Class2, in the example below.



Having created the association it is shown as active, and the Properties tab displays its properties. A text label shows the default name of the member end of the association, i.e. Property1. Note that the context menu option **Text Label...** allows you to show / hide labels.

Clicking an association line, displays the association properties in the Properties tab. A:Name and B:Name indicate the role of each class in the other.



Depending on the "memberEndKind" - **property** (of A:name "Property1"):

the **attribute** either belongs to:

- the **class** - i.e. **A:memberEndKind = memberEnd**, (attribute is visible in class1), or
- the **association** - i.e. **B:memberEndKind = ownedEnd** (attribute not visible in class2).

If both attributes belongs to the **association**, i.e. both ends are defined as "ownedEnd", then this association becomes bi-directional, and the navigability arrow disappears. Both ends of the association are "ownedEnd".

If the memberEndKind of any of the association is set to "**navigableOwnedEnd**", then the attribute is still part of the association, but the navigability arrow reappears depending on which end (A:name or B:Name) it is set.

To define the type of association (association, aggregate, or composite)

1. Click the association arrow.
2. Scroll down to the **aggregation** item in the Properties tab.
3. Select: none, shared or composite.

None: a standard association
shared: an **aggregate** association
composite: a **composite** association.

Please note:

Associations can be created using the same class as both the source and target. This is a so-called self link. It describes the ability of an object to send a message to itself, for recursive calls.

Click the relationship icon, then drag from the element, dropping somewhere else on the same element. A self-link appears.

Displaying associations in Diagrams automatically

When inserting diagram elements in a diagram, the "Automatically create Associations" option in the **Tools | Options | Editing** tab, allows existing associations between modeling elements to be automatically created/displayed in the current diagram. This occurs if the attributes type is set, and the referenced "type" modeling element is in the current diagram.

Deleting relationships/associations:

1. Click the relationship in the diagram tab, or in the Model Tree.
2. Press the **Del.** keyboard key.
The dependency is deleted from the diagram and project.

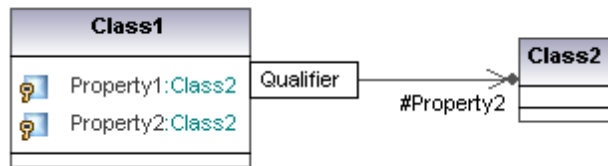
Deleting class associations:

Deleting a **class** association does not delete the **attribute/property** that was automatically generated, from the class!

1. Right click the attribute/property in the class.
2. Select the option "**Delete PropertyX**" from "ClassX" to delete it.

Creating association qualifiers:

1. Having defined an association between two classes
2. Right click the association line and select **New | Qualifier**.

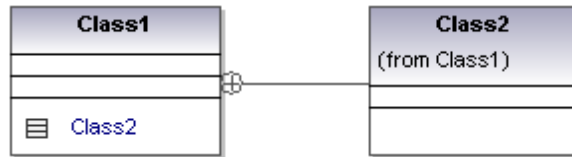


Please note that qualifiers are attributes of an association.

Creating a containment association:

The containment association is used to show parent-child relationships.

1. Click the containment icon  in the title bar.
2. Drag from the class that is to be "contained", and drop on the container class.

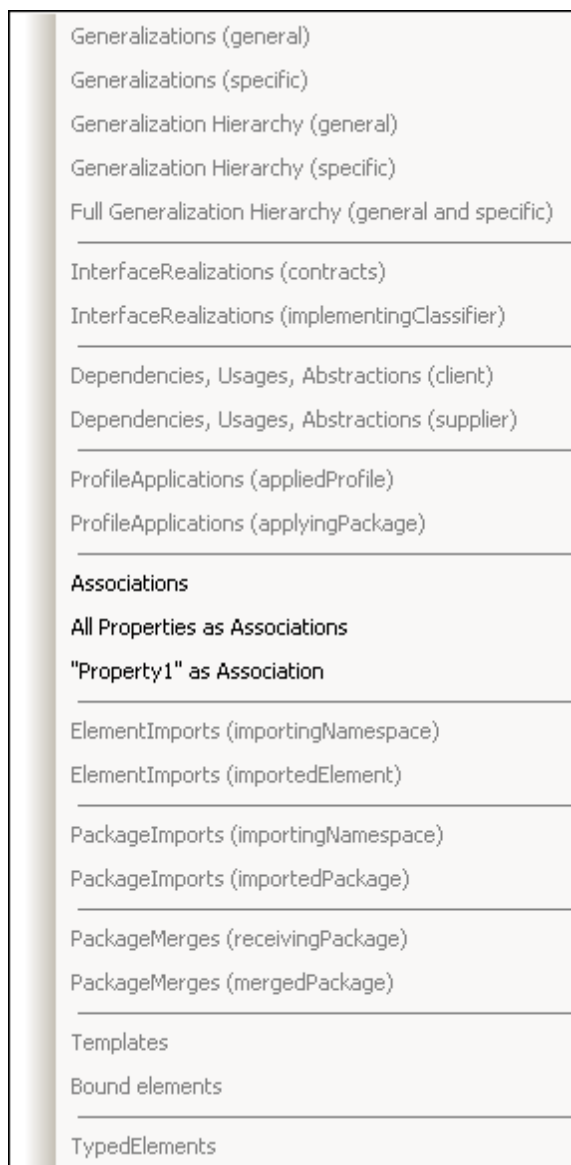


Note that the contained class, Class2 in this case, is now visible in a compartment of Class1. This generally places the contained class in the same namespace as the container class.

8.1 Showing model relationships

Showing relationships between modeling elements:

1. Right click the specific element and select **Show**.
The pop-up menu shown below is context specific, meaning that only those options are available that are relevant to the specific element.

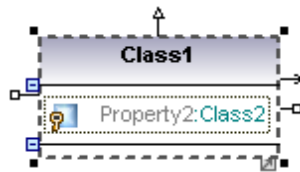


To show / hide text labels:

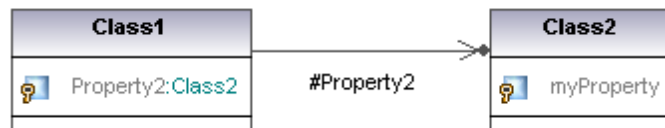
- Right click a class or association arrow and select **Text labels | Show (Hide) all Text labels**.

To show a class attribute/property as an association:

1. Right click the property in the class.



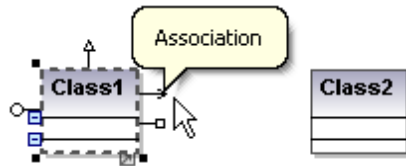
2. Select the menu option **Show | "PropertyXX" as Association**.
This inserts/opens the referenced class and shows the relevant association.



8.2 Associations, realizations and dependencies

Creating relationships using connection handles:

1. Given two classes in the class diagram,
2. Click the first class to make it the active class.
Connection handles appear on three sides.
3. Move the mouse pointer over the **top** handle on the right side of the class.

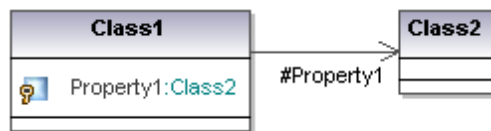


A tooltip appears, informing you of the type of relationship that this handle creates, Association in this case.

4. Drag to create a connector, and drop it on the second class.
The target class is highlighted if this type of association is possible.



An association has now been created between these two classes.



Note that the **lower** handle of the class element creates a Collection Association.

Elements in the various model diagrams supply you with different connection handles.

E.g. a class in a class diagram supplies the following relationship handles (in clockwise fashion):

- InterfaceRealization
- Generalization
- Association

An Artifact in the Deployment view supplies the following handles:

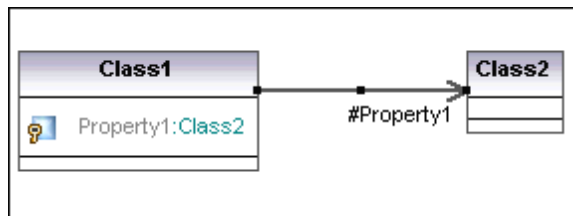
- Manifestation
- Association
- Deployment

Creating relationships using icons in the icon bar:

Given two elements in a modeling diagram,

1. Click the icon that represents the relationship you want to create e.g. association, aggregation, or composition.
2. Drag from the one object to the other, and drop when the target element is highlighted.

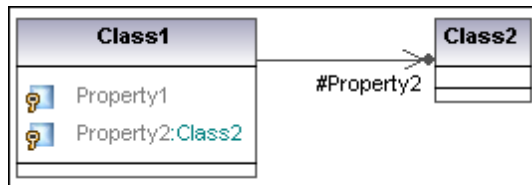
When creating a new association, a new attribute is automatically inserted in the originating (A:name) class, Property1:Class2, in the example below.



UModel always shows all attributes of a class!

Please note:

The screenshots in this manual do not show the Association **Ownership dot**.



To enable it, set the **Show Assoc. Ownership**, in the Styles tab, to true.

Deleting relationships/associations:

1. Click the relationship in the diagram tab, or in the Model Tree.
 2. Press the **Del.** keyboard key.
- The dependency is deleted from the diagram and project.

Deleting class associations:

Deleting a **class** association does not delete the **attribute/property** that was automatically generated, from the class!

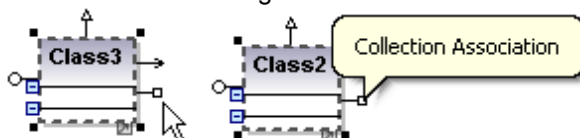
1. Right click the attribute/property in the class.
2. Select the option "**Delete PropertyX**" from "ClassX" to delete it.

Collection Associations

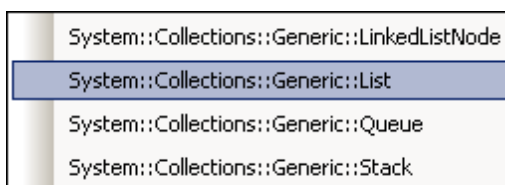
UModel supports a special type of viewing Collection Associations.

Collection associations are special associations for collection templates. A class property (e.g. interface) can generally be shown as an association to the "type" of the property. UModel also allows an alternate method of viewing this type of association.

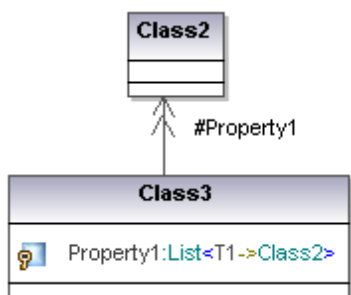
Click on the lower of the two class handles on the right of the class, then drag and drop the connector onto the target class.



Select the collection type from the pop-up menu.

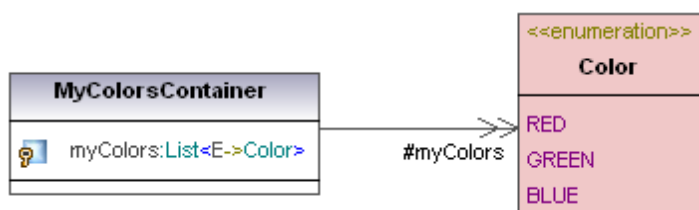


A different type of association arrow is created.



Example:

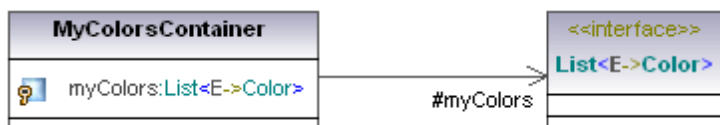
If associations are automatically created during reverse engineering, you will see them as Collection Associations if the settings in the Diagram Editing tab are set to: Resolve collections and are also available in the Collection Templates dialog box.



The double arrow head shows that the "type" of myColors is not only Color, but a "collection of" Colors.

The association will not be shown as List<E->Color>, but directly to the enumeration "Color", thus hiding the information that Color is used in a template binding.

The concrete collection type of myColors can still be seen in the myColorsContainer, but not in the association.



The Diagram Editing tab of the **Tools | Options** dialog box allows you to specify the templates where you want this behavior and whether you want the collections to be resolved.

Chapter 9

Generating UML documentation

9 Generating UML documentation

Altova web site:  [UML project documentation](http://www.altova.com/uml-project-documentation)

The **Generate Documentation** command generates detailed documentation about your UML project in HTML, MS Word, RTF or PDF. The documentation generated by this command can be freely altered and used; permission from Altova to do so is not required.

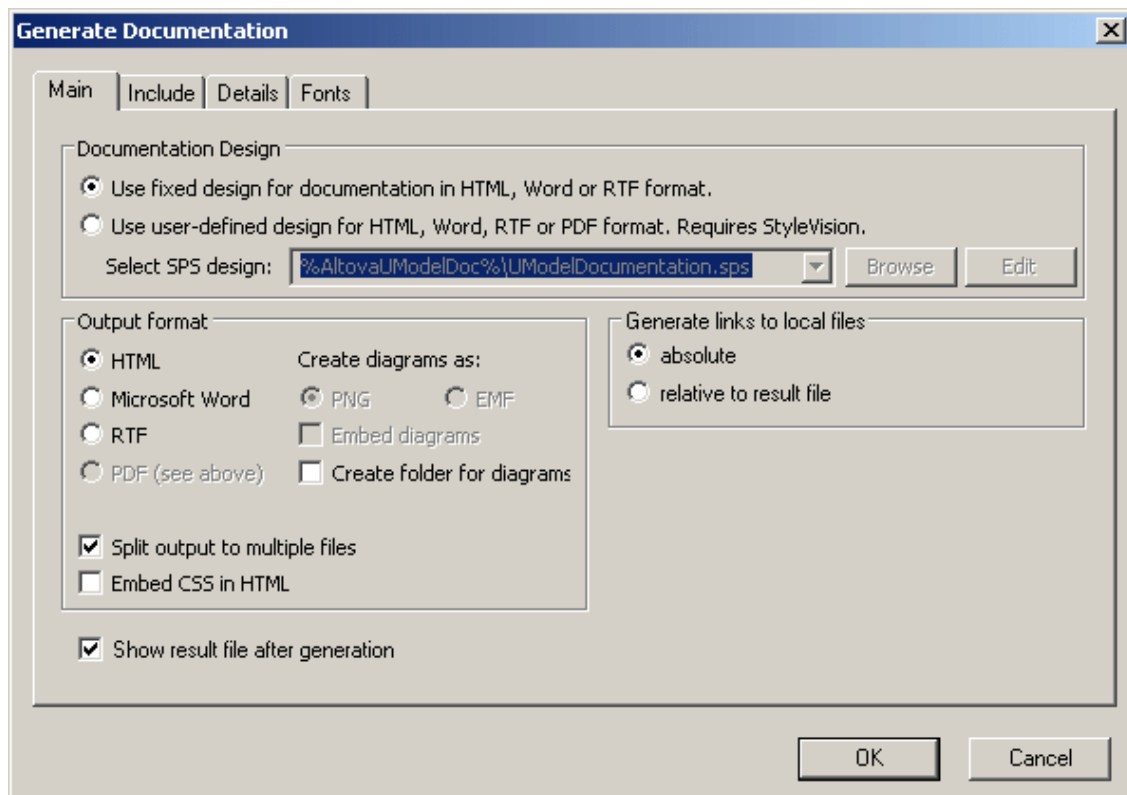
Documentation is generated for the modeling elements you select in the Generate Documentation dialog box. You can either use the fixed design, or use a StyleVision SPS for the design. Using a StyleVision SPS enables you to customize the output of the generated documentation. How to do this is explained in the section, [User-Defined Stylesheets](#).

Note: To use an SPS to generate documentation, you must have StyleVision installed on your machine. Related elements are typically hyperlinked in the onscreen output, enabling you to navigate from component to component.

To generate documentation in MS Word format, you must have MS Word (version 2000 or later) installed.

Having opened a UML project e.g. Bank_Multilanguage.ump:

1. Select the menu option **Project | Generate Documentation**.
This opens the "Generate documentation" dialog box. The screenshot below shows the default dialog box settings.



Note that you can also create **partial documentation** of modeling elements by right clicking an

element (or **multiple** elements using **Ctrl + Click**) in the Model Tree and selecting "Generate Documentation". The element can be a folder, class, interface etc. The documentation options are the same in both cases.

Related elements are hyperlinked in the onscreen output, enabling you to navigate from component to component. All manually created hyperlinks also appear in the documentation.

Note also:

documentation is also generated for **included** C#, Java and/or VB subprojects (profiles) if enabled in the Include tab!

Main tab:

Documentation Design

- Select "Use fixed design..." to use the built-in documentation template.
- Select "Use user-defined..." to use a predefined StyleVision Power Stylesheet created in StyleVision. The SPS files are available in the ...**\My Documents\Altova\UModel2016\Documentation\UModel** folder.
- Click **Browse** to browse for a predefined SPS file.
- Click **Edit** to launch StyleVision and open the selected SPS in a StyleVision window.

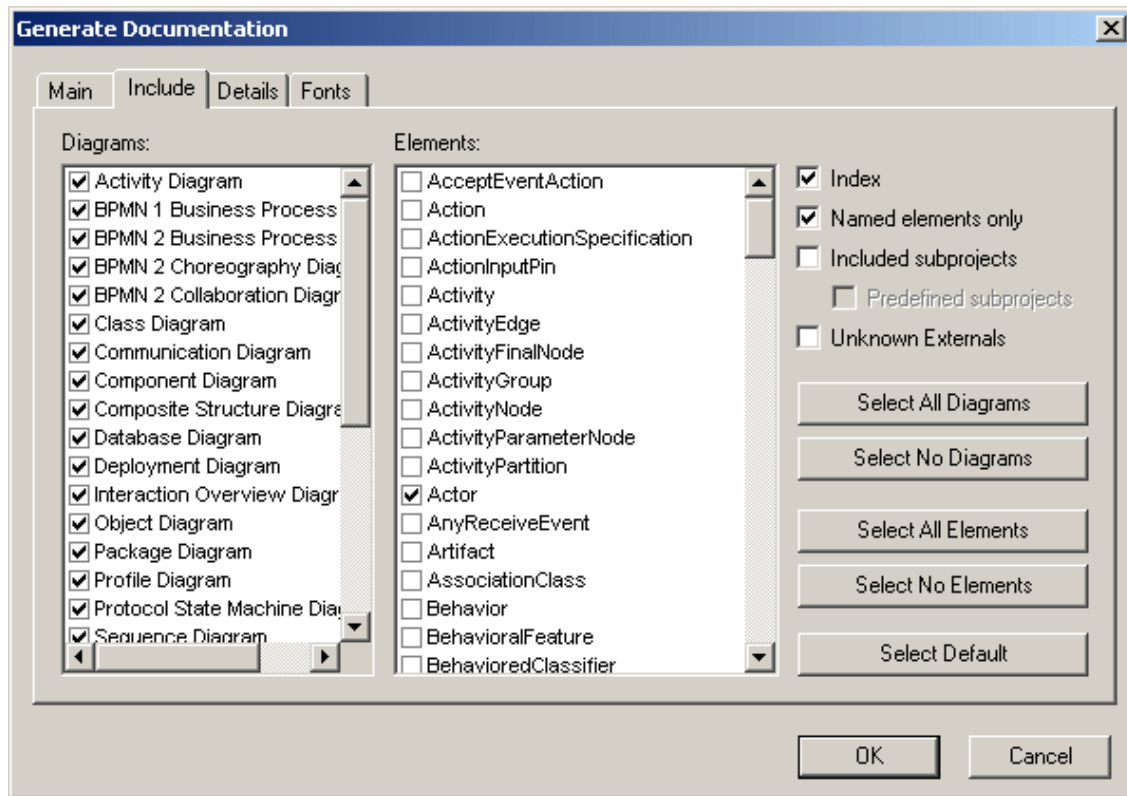
Output format:

- The output format is specified here: either HTML, Microsoft Word, RTF, or PDF.
Microsoft Word documents are created with the **.doc** file extension when generated using a fixed design, and with a **.docx** file extension when generated using a StyleVision SPS.
The **PDF** output format is only available if you use a StyleVision SPS to generate the documentation.
- **"Split output to multiple files"** generates an output file for each modeling element that would appear in the TOC overview when generating a single output file e.g. a class C1 with a nested class CNest exists; C1.html contains all info pertaining to C1 and CNest as well as all their attributes, properties etc
- The **Embed CSS in HTML** option allows you to embed an existing (referenced) CSS file in the HTML documentation, when active; or keep the reference to the CSS file, and the file itself, external when inactive.
- The **Embed diagrams** option is enabled for the Microsoft Word and RTF output options. When this option is selected, diagrams are embedded in the generated file. Diagrams are created as PNG files (for HTML), or PNG/EMF files (for MS Word and RTF), which are displayed in the result file via object links.
- **Create folder for diagrams** generates a subfolder, below the selected output folder, that will contain all diagrams when the documentation is output.
- The **"Show Result File..."** option is enabled for all output options. When checked, the result files are displayed in Browser View (HTML output), MS Word (MS Word output), and the default application for .rtf files (RTF output).

- **Generate links to local files** option allows you to specify if the generated links are to be absolute, or relative, to the output file.

Include tab

allows you to select which diagrams and modeling elements are to appear in the documentation.

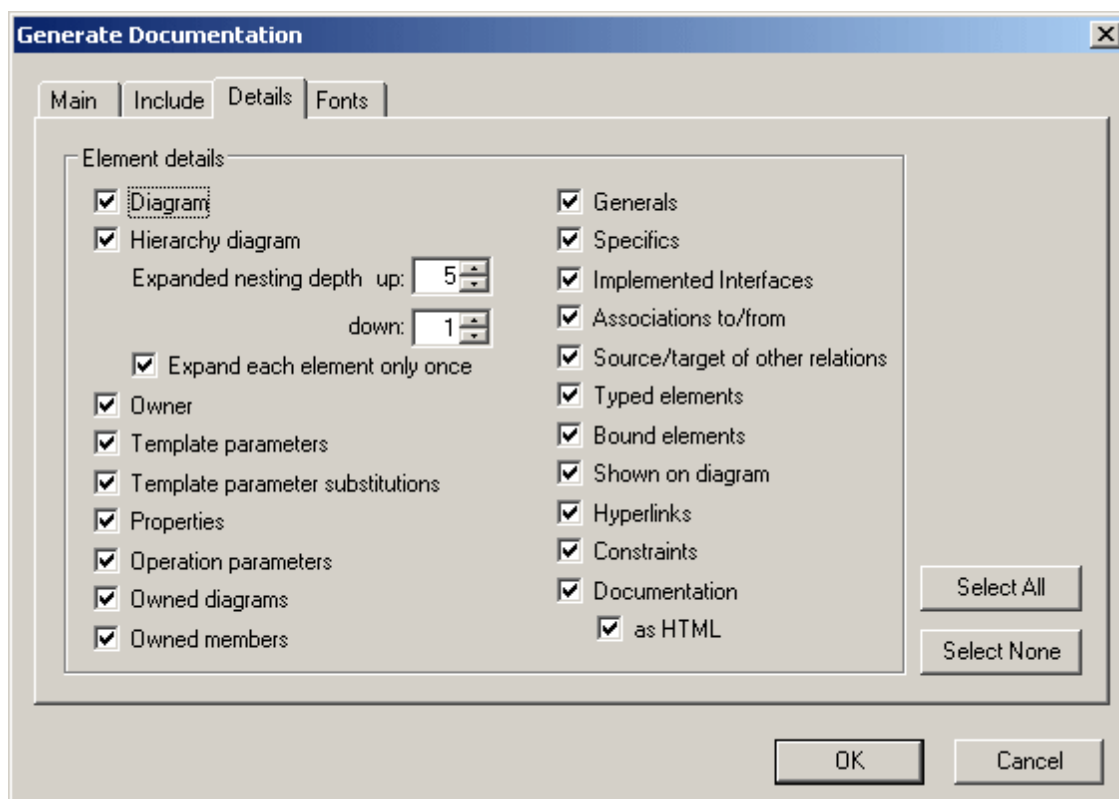


Note that documenting subprojects can be disabled by deselecting the "Included subprojects" check box.

Details tab

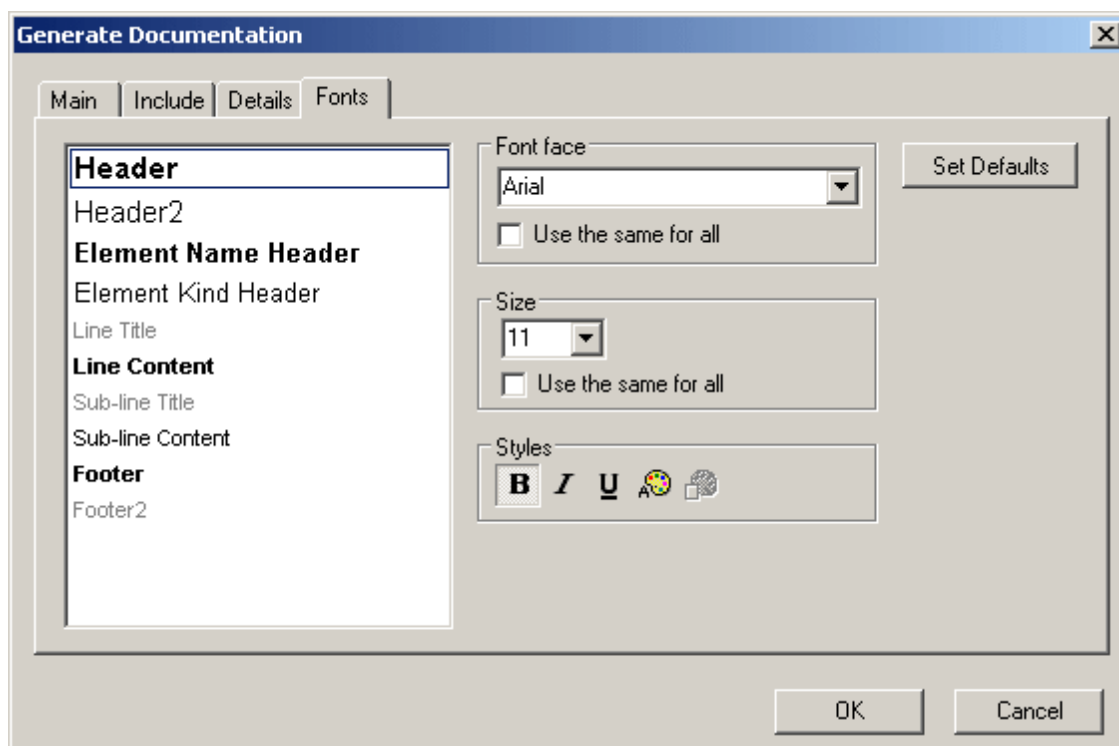
allows you to select the element details that are to appear in the documentation.

- If you intend to import **XML tags** text in your documentation, please **de**-activate the "as HTML" option under the Documentation option.
- The up / down fields allow you to define the nesting depth shown above/below the current class in the hierarchy diagram.
- The "expand each element only once" option, only allows one of the same classifiers to be expanded in the same image/diagram.



Fonts tab

allows you to customize the font settings for the various headers and text content.



The following screenshots show the fixed-design documentation for the Bank_MultiLanguage.ump file that is included in the ...\\UModelExamples directory.

Bank_MultiLanguage.ump			
project location Bank_MultiLanguage.ump			
Index of diagrams:			
Activity Diagram	collectData Draft		
Class Diagram	Apply Java Profile	Bank Server	BankView Main
Component Diagram	Bank realizations	Overview	
Composite Structure Diagram	Account Transfer		
Deployment Diagram	Deployment		
Object Diagram	Sample Accounts		
Sequence Diagram	Collect Account Information	Connect to BankAPI	
State Machine Diagram	BankAPI Draft	Query BankServer Draft	
UseCase Diagram	Overview Account Balance		
Index of elements:			
Activity	BankView		
Actor	Bank	Standard User	
Artifact	BankAddresses.ini	BankAPI.jar	BankServer
Class	Account CheckingAccount	Bank CreditCardAccount	BankServer SavingsAccount
Collaboration	Account Transfer	Bank Account Transfer	
Component	Bank API client	BankServer	BankView
Enumeration	AccountType		
InstanceSpecification	AgencyBank	John's 1st	John's 2nd
Interaction	Collect Account Information	Connect to BankAPI	
Interface	IBankAPI		

The screenshot above shows the generated documentation with the diagram and element index links at the top of the HTML file.

The screenshot below shows the specifics of the Account class and its relation to other classes.

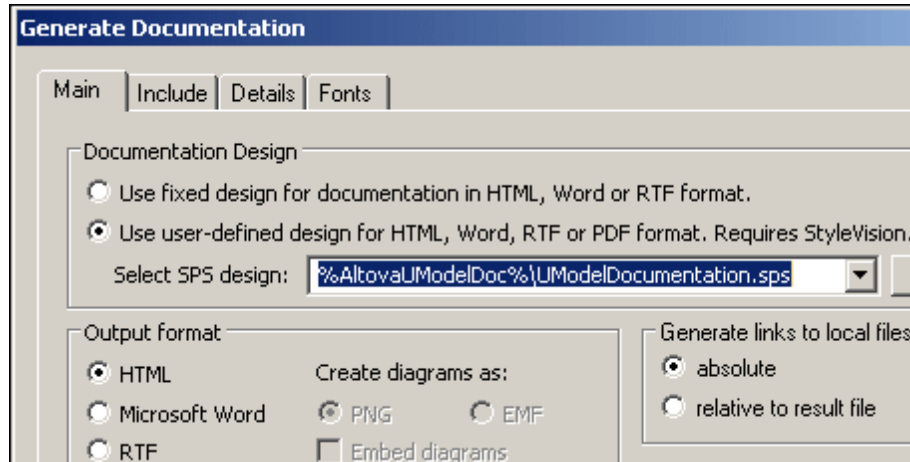
Note that the individual attributes and properties in the class diagrams are also hyperlinked to their definitions. Clicking a property takes you to its definition. The individual hierarchy classes, as well as all underlined text, are hyperlinked.

Class Account	
diagram	<pre> classDiagram class Account { balance: float=0 id: String <<constructor>> Account() getBalance(): float getId(): String collectAccountInfo(in bankAPI: IBankAPI): boolean } class CheckingAccount class SavingsAccount class CreditCardAccount Account < -- CheckingAccount Account < -- SavingsAccount Account < -- CreditCardAccount </pre>
hierarchy	<pre> classDiagram class Account class CheckingAccount class SavingsAccount class CreditCardAccount Account < -- CheckingAccount Account < -- SavingsAccount Account < -- CreditCardAccount </pre>
owner	<u>bankview</u>
properties	qualified name Design View::BankView::com::altova::bankview::Account visibility public leaf false abstract true active false code file name Account.java code file path <u>..\\..\\..\\..\\UML_Bank_Sample\\MultiLanguage\\JavaCode\\com\\altova\\bankview\\Account.java</u> <<annotations>> false <<static>> false <<final>> false <<strictfp>> false
ownedMember	<u>Account balance collectAccountInfo getBalance getId id</u>
specific	<u>CheckingAccount CreditCardAccount SavingsAccount</u>
associations from	from Association name <u>Bank (accounts)</u>
target of relation	ComponentRealization <u>BankView</u>
typedElements	Class <u>Bank</u> Property <u>accounts</u>

9.1 Supplied SPS stylesheet

To generate documentation based on the supplied SPS file:

1. Select the menu option **Project | Generate Documentation**.
2. Select the "Use User-Defined Design..." radio button.
3. Select the **UModelDocumentation.sps** stylesheet (if not automatically selected) in the Generate Documentation dialog box.



You are now prompted to save the file.

4. Enter the file name and select the location you want to save it.

Bank_MultiLanguage.ump

project location **C:\Documents and Settings\My\My Documents\Altova\UModel2011\UModelDocumentation\Bank_MultiLanguage.ump**

Index of diagrams:

Activity Diagram	<u>collectData Draft</u>
Component Diagram	<u>Bank realizations</u> <u>Overview</u>
Composite Structure Diagram	<u>Account Transfer</u>
Deployment Diagram	<u>Deployment</u>
Sequence Diagram	<u>Collect Account Information</u> <u>Connect to BankAPI</u>
State Machine Diagram	<u>BankAPI Draft</u> <u>Query BankServer Draft</u>

Index of elements:

Component	<u>BankView GUI</u>
Package	<u>Root</u> <u>Behavior View</u> <u>Design View</u>

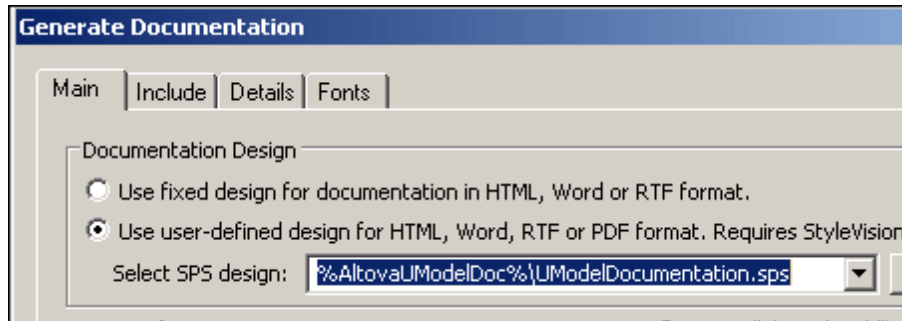
Clicking a link, in the generated documentation, navigates to the modeling element in the browser.

9.2 User-defined Stylesheets

Instead of using the fixed design, you can create a customized design for the UModel documentation. The customized design is created in a StyleVision SPS. Note that the predefined SPS Stylesheet is supplied with UModel.

Specifying the SPS to use for UModel documentation

The SPS you wish to use for generating the documentation is specified in the Generate Documentation dialog (accessed via **Project | Generate Documentation**). Select the "Use User-Defined Design..." radio button. The default selection is the **UModelDocumentation.sps** entry.



The predefined SPS file is located in the ...**Documentation\UModel** folder.

Please note:

To use an SPS to generate documentation, you must have StyleVision installed on your machine.

Creating the SPS

A StyleVision Power Stylesheet (or SPS) is created using [Altova StyleVision](#). An SPS for generating UModel documentation must be based on the XML Schema that specifies the structure of the XML document that contains the UModel documentation.

This schema is called **UModelDocumentation.xsd** and is delivered with your UModel installation package. It is stored in the ...*My Documents\Altova\UModel2016\Documentation\UModel* folder.

When creating the SPS design in StyleVision, nodes from the UModelDocumentation.xsd schema are placed in the design and assigned styles and properties. Note that the UModelDocumentation.xsd **includes** the Documentation.xsd file located in the folder above it.

Additional components, such as links and images, can also be added to the SPS design. How to create an SPS design in StyleVision is described in detail in the StyleVision user manual.

The advantage of using an SPS for generating schema documentation is that you have complete control over the design of the documentation. Note also that PDF output of the documentation is available only if an SPS is used; PDF output is not available if the fixed design is used.

Chapter 10

UML Diagrams

10 UML Diagrams

There are two major groups of UML diagrams, Structural diagrams, which show the static view of the model, and Behavioral diagrams, which show the dynamic view. UModel supports all fourteen diagrams of the UML 2.4 specification as well as: XML Schema and Business Modeling Notation diagrams.

[Behavioral diagrams](#) include Activity, State machine, Protocol State Machine and Use Case diagrams; as well as the Interaction, Communication, Interaction Overview, Sequence, and Timing diagrams.

[Structural diagrams](#) include: Class, Composite Structure, Component, Deployment, Object, and Package diagrams.

[Additional diagrams](#) XML schema diagrams, Business Processing Modeling Notation (BPMN), SysML diagrams, Database diagrams.

Please note:

The **Ctrl+Enter** keys can be used to create multi-line labels for most of the modeling diagrams, e.g. Lifeline labels in sequence diagrams, timing diagrams; guard conditions, state names, activity names etc.

10.1 Behavioral Diagrams

These diagrams depict behavioral features of a system or business process, and include a subset of diagrams which emphasize object interactions.

Behavioral Diagrams

-  [Activity Diagram](#)
-  [State Machine Diagram](#)
-  [Protocol State Machine Diagram](#)
-  [Use Case Diagram](#)

A subset of the Behavioral diagrams are those that depict the object interactions, namely:

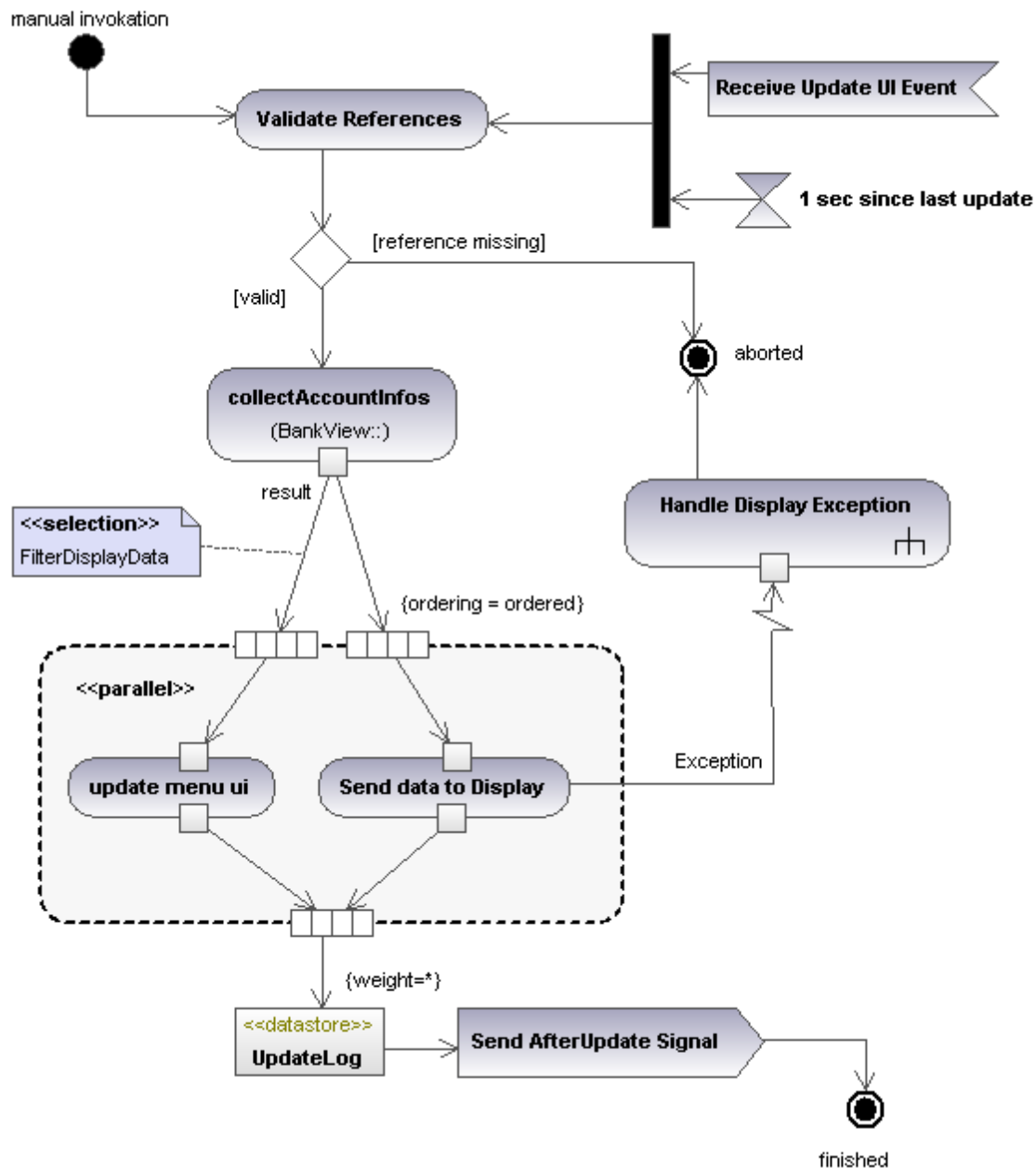
-  [Communication Diagram](#)
-  [Interaction Overview Diagram](#)
-  [Sequence Diagram](#)
-  [Timing Diagram](#)

10.1.1 Activity Diagram

Altova web site:  [UML Activity diagrams](http://www.altova.com/uml-activity-diagrams)

Activity diagrams are useful for modeling real-world workflows of business processes, and display which actions need to take place and what the behavioral dependencies are. The Activity diagram describes the specific sequencing of activities and supports both conditional and parallel processing. The Activity diagram is a variant of the State diagram, with the states being activities.

Please note that the Activity diagram shown in the following section is available in the **Bank_MultiLanguage.ump** sample, in the ...\\UModelExamples folder supplied with UModel.



Inserting Activity Diagram elements



Using the toolbar icons:

1. Click the specific activity diagram icon in the Activity Diagram toolbar.
2. Click in the Activity Diagram to insert the element. To insert multiple elements of the selected type, hold down the **Ctrl** key and click in the diagram window.

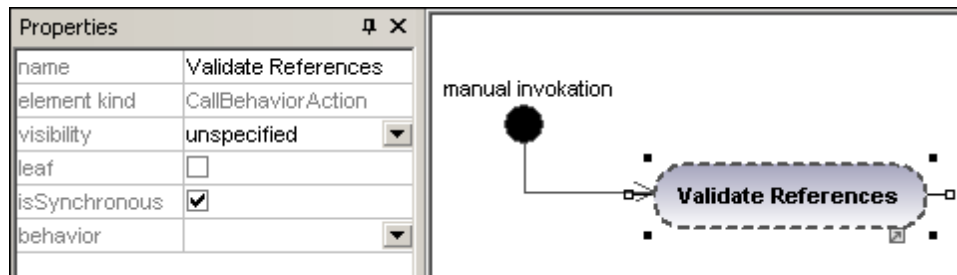
Dragging existing elements into the activity diagram:

Most elements occurring in other activity diagrams, can be inserted into an existing activity diagram.

1. Locate the element you want to insert in the Model Tree tab (you can use the search function text box, or press **Ctrl+F** to search for any element).
2. Drag the element(s) into the activity diagram.

Inserting an action (CallBehavior):

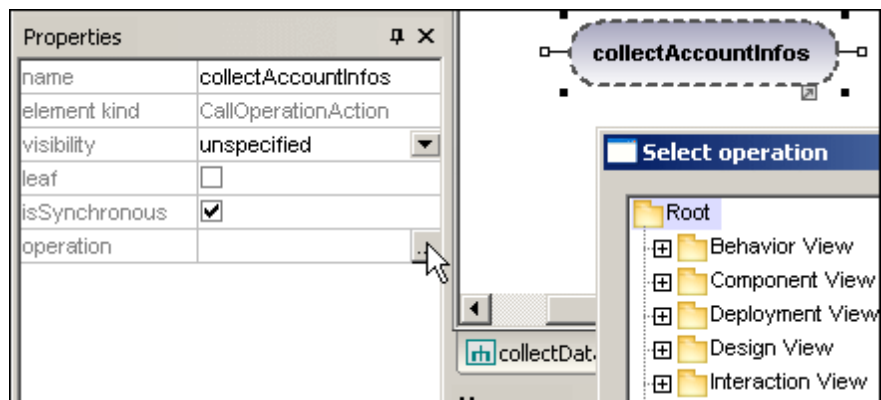
1. Click the Action (CallBehavior) icon  in the icon bar, and click in the Activity diagram to insert it.
2. Enter the name of the Action, e.g. Validate References, and press Enter to confirm.



Note: Use **Ctrl+Enter** to create a multi-line name.

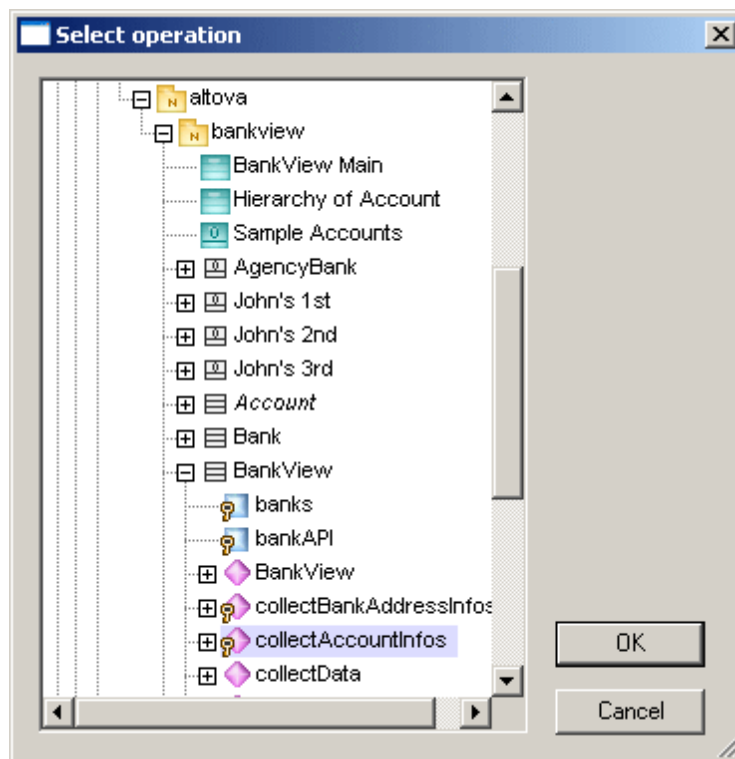
Inserting an action (CallOperation) and selecting a specific operation:

1. Click the Action (CallOperation) icon  in the icon bar, and click in the Activity diagram to insert it.
2. Enter the name of the Action, e.g. collectAccountInfo, and press Enter to confirm.
3. Click the Browse button to the right of the operation field in the Properties tab.

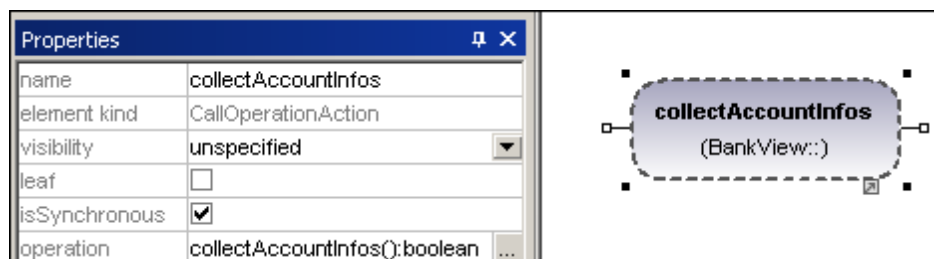


This opens the "Select Operation" dialog box in which you can select the specific operation.

4. Navigate to the specific operation that you want to insert, and click OK to confirm.



In this example the operation "collectAccountInfos" is in the BankView class.



Creating branches and merges

Creating a branch (alternate flow)

A branch has a single incoming flow and multiple outgoing guarded flows. Only one of the outgoing flows can be traversed, so the guards should be mutually exclusive.

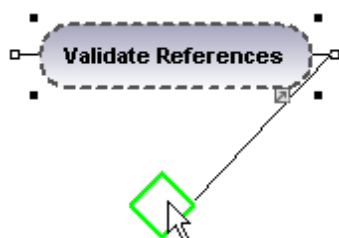
In this example the (BankView) references are to be validated:

- branch1 has the guard "reference missing", which transitions to the abort activity
- branch2 has the guard "valid", which transitions to the collectAccountInfos activity.

1. Click the **DecisionNode** icon  in the title bar, and insert it in the Activity diagram.

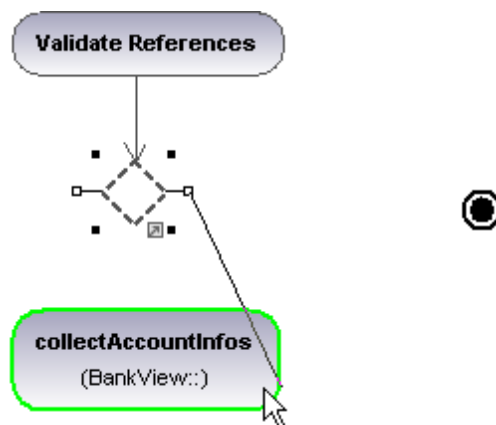


2. Click the ActivityFinalNode icon  which represents the abort activity, and insert it into the Activity diagram.
3. Click the Validate References activity to select it, then click the right-hand handle, **ControlFlow**, and drag the resulting connector onto the DecisionNode element.

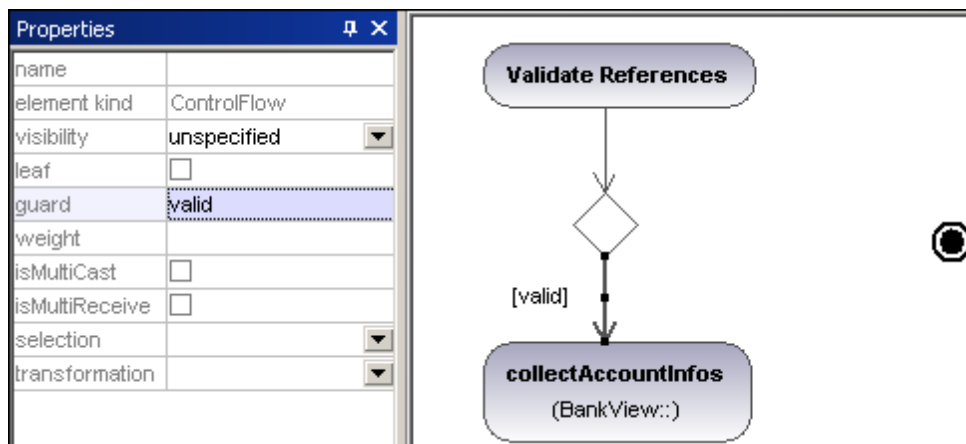


The element is highlighted when you can drop the connector.

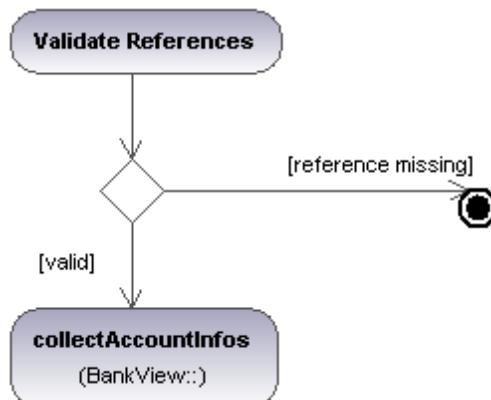
4. Click the DecisionNode element, click the right-hand connector, **ControlFlow**, and drop it on the collectAccountInfos action. Please see "[Inserting an Action \(CallOperation\)](#)" for more information.



5. Enter the guard condition "valid", in the guard field of the Properties tab.



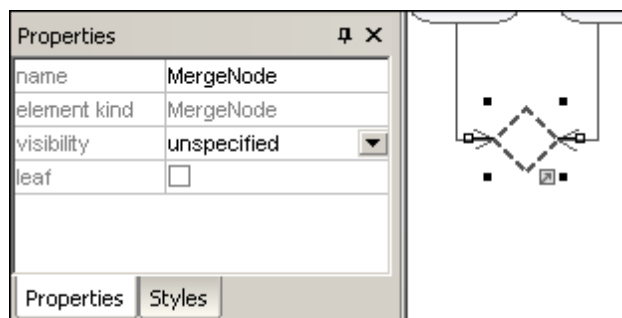
6. Click the **DecisionNode** element and drag from the right-hand handle, **ControlFlow**, and drop it on the ActivityFinalNode element. The guard condition on this transition is automatically defined as "else". Double click the guard condition in the diagram to change it e.g. "reference missing".



Please note that UModel does not validate, or check, the number of Control/Object Flows in a diagram.

Creating a merge:

1. Click the MergeNode icon  in the icon bar, then click in the Activity diagram to insert it.



2. Click the ControlFlow (ObjectFlow) handles of the actions that are to be merged, and drop the arrow(s) on the MergeNode symbol.

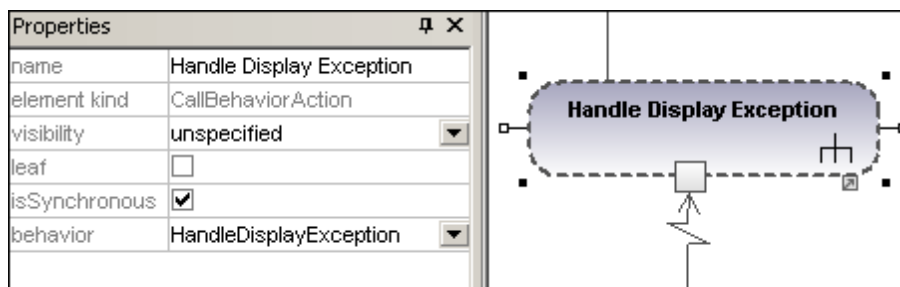
Activity Diagram elements



Action (CallBehavior)

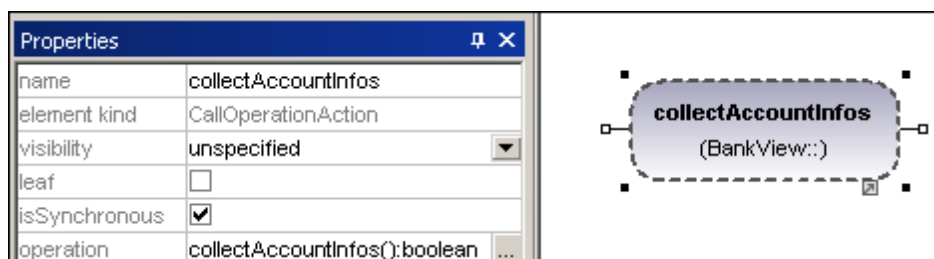
Inserts the Call Behavior Action element which directly invokes a specific behavior.

Selecting an existing behavior using the **behavior** combo box, e.g. HandleDisplayException, and displays a rake symbol within the element.



Action (CallOperation)

Inserts the Call Operation Action which indirectly invokes a specific behavior as a method. Please see "[Inserting an action \(CallOperation\)](#)" for more information.



Action (OpaqueAction)

A type of action used to specify implementation information. Can be used as a placeholder until you decide which specific action type you want to use.

**Action (ValueSpecificationAction)**

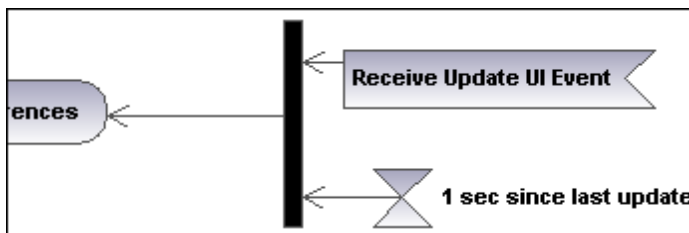
A type of action that evaluates(/generates) a specific value at the output pin. (Defined by the specific properties, e.g. upperBound.)

**AcceptEventAction**

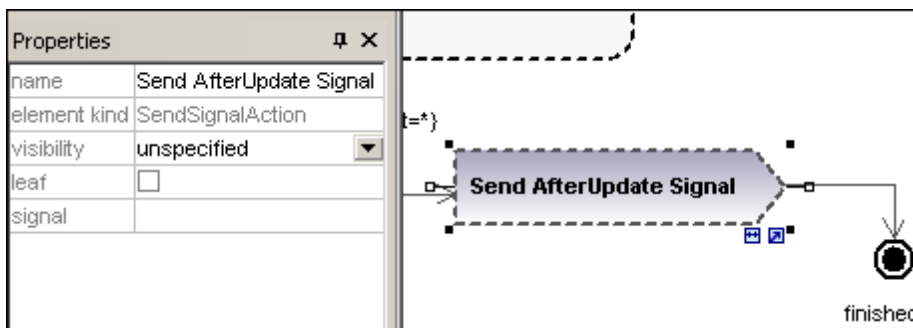
Inserts the Accept Event action which waits for the occurrence of an event which meets specific conditions.

**AcceptEventAction (TimeEvent)**

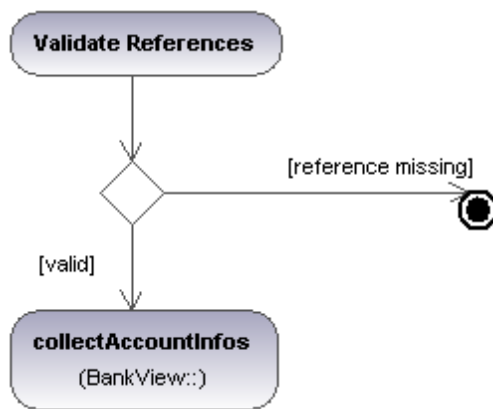
Inserts a AcceptEvent action, triggered by a time event, which specifies an instant of time by an expression e.g. 1 sec. since last update.

**SendSignalAction**

Inserts the Send Signal action, which creates a signal from its inputs and transmits the signal to the target object, where it may cause the execution of an activity.

**DecisionNode**

Inserts a Decision Node which has a single incoming transition and multiple outgoing guarded transitions. Please see "[Creating a branch](#)" for more information.

**MergeNode**

Inserts a Merge Node which merges multiple alternate transitions defined by the Decision Node. The Merge Node does not synchronize concurrent processes, but selects one of the processes.

**InitialNode**

The beginning of the activity process. An activity can have more than one initial node.

**ActivityFinalNode**

The end of the activity process. An activity can have more than one final node, all flows in the activity stop when the "first" final node is encountered.

**FlowFinalNode**

Inserts the Flow Final Node, which terminates a flow. The termination does not affect any other flows in the activity.

**ForkNode**

Inserts a vertical Fork node.
Used to divide flows into multiple concurrent flows.

**ForkNode (Horizontal)**

Inserts a horizontal Fork node.
Used to divide flows into multiple concurrent flows.

**JoinNode**

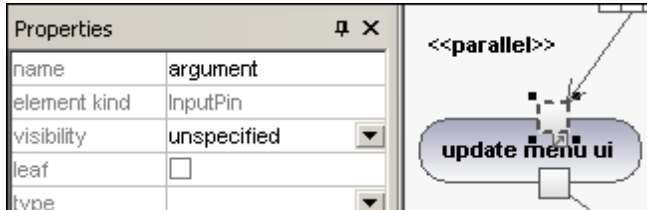
Inserts a vertical Fork node.
A Join node synchronizes multiple flows defined by the Fork node.

**Join Node (horizontal)**

Inserts a horizontal Fork node.
A Join node synchronizes multiple flows defined by the Fork node.

**InputPin**

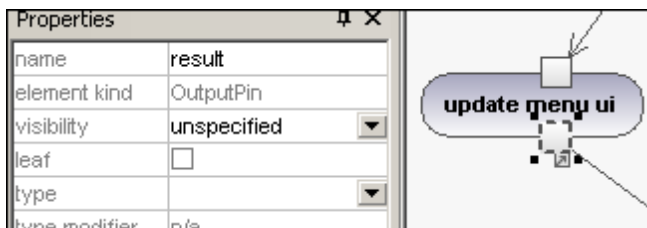
Inserts an input pin onto a Call Behavior, or Call Operation action. Input pins supply input values that are used by an action. A default name, "argument", is automatically assigned to an input pin.



The input pin symbol can only be placed onto those activity elements where the mouse pointer changes to the hand symbol . Dragging the symbol repositions it on the element border.

**OutputPin**

Inserts an output pin action. Output pins contain output values produced by an action. A name corresponding to the UML property of that action e.g. result, is automatically assigned to the output pin.



The output pin symbol can only be placed onto those activity elements where the mouse pointer changes to the hand symbol . Dragging the symbol repositions it on the element border.

Exception Pin

An OutputPin can be changed to an Exception pin by clicking the pin and selecting "isExceptionPin" from the Properties pane.

**ValuePin**

Inserts a Value Pin which is an input pin that provides a value to an action, that does not come from an incoming object flow. It is displayed as an input pin symbol, and has the same properties as an input pin.

**ObjectNode**

Inserts an object node which is an abstract activity node that defines object flow in an activity. Object nodes can only contain values at runtime that conform to the type of the object node.

**CentralBufferNode**

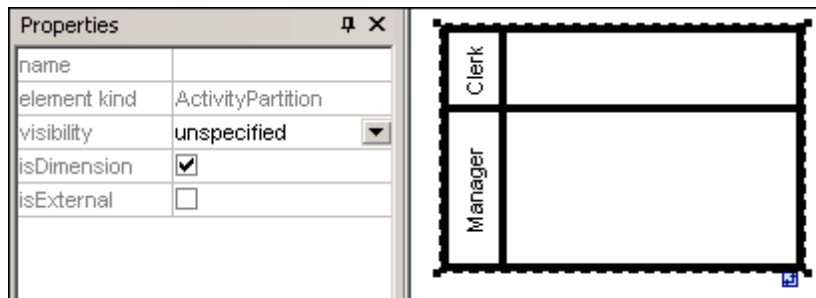
Inserts a Central Buffer Node which acts as a buffer for multiple in- and out flows from other object nodes.

**DataStoreNode**

Inserts a Data Store Node which is a special "Central Buffer Node" used to store persistent (i.e. non transient) data.

**ActivityPartition (horizontal)**

Inserts a horizontal Activity Partition, which is a type of activity group used to identify actions that have some characteristic in common. This often corresponds to organizational units in a business model.



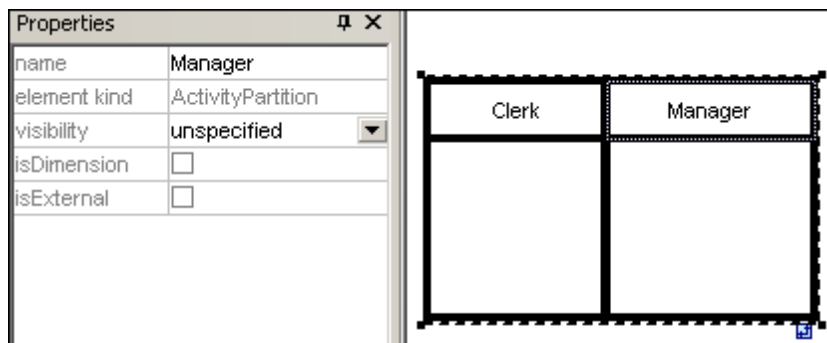
Double clicking a label allows you to edit it directly; pressing Enter orients the text correctly.

Please note that Activity Partitions are the UML 2.0 update to the "swimlane" functionality of previous UML versions.

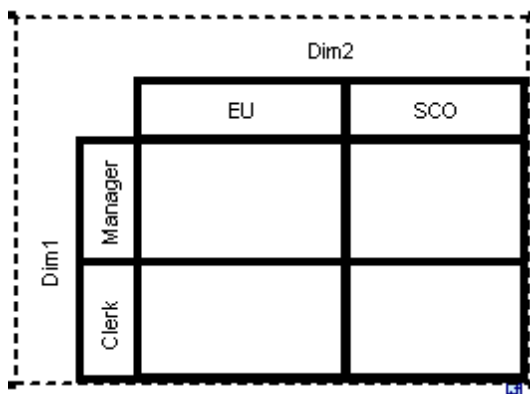
- Elements placed within a ActivityPartition become part of it when the boundary is highlighted.
- Objects within an ActivityPartition can be individually selected using **Ctrl+Click**, or by dragging the marquee inside the boundary.
- Click the ActivityPartition boundary, or title, and drag to reposition it.

**ActivityPartition (vertical)**

Inserts a vertical Activity Partition, which is a type of activity group used to identify actions that have some characteristic in common. This often corresponds to organizational units in a business model.

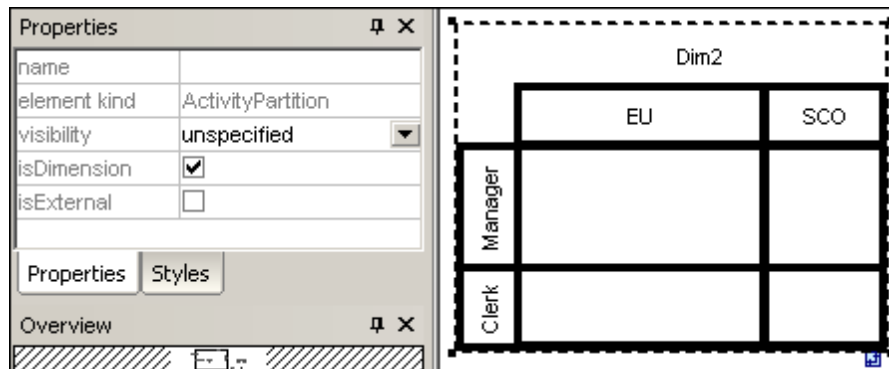
**ActivityPartition (2 Dimensional)**

Inserts a two dimensional Activity Partition, which is a type of activity group used to identify actions that have some characteristic in common. Both axes have editable labels.



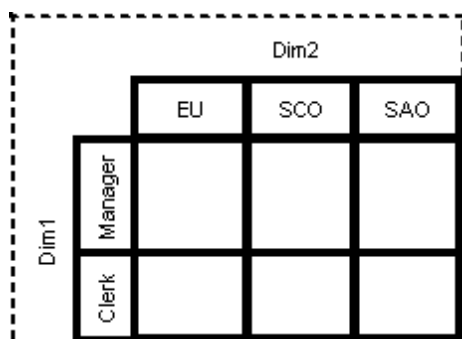
To remove the Dim1, Dim2 dimension labels:

1. Click the dimension label you want to remove e.g. Dim1
2. Double click in the Dim1 entry in the Properties tab, delete the Dim1 entry, and press Enter to confirm.



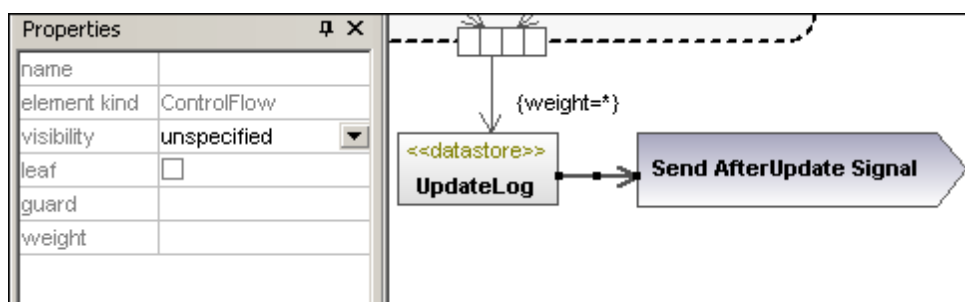
Note that Activity Partitions can be nested:

1. Right click the label where you want to insert a new partition.
2. Select **New | ActivityPartition**.



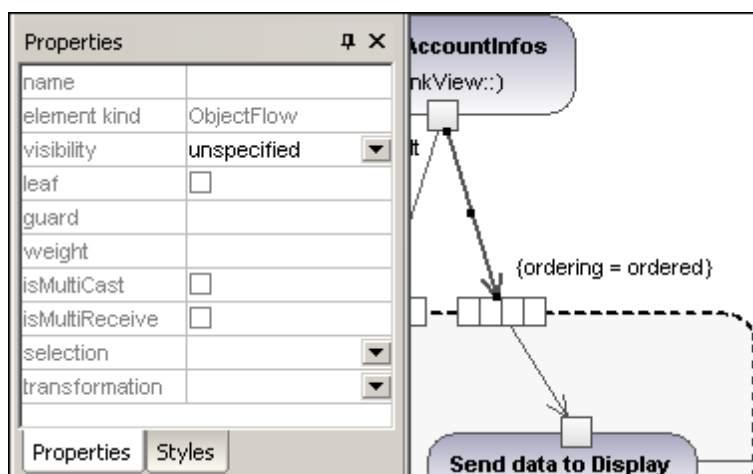
ControlFlow

A Control Flow is an edge, i.e. an arrowed line, that connects two activities/behaviours, and starts an activity after the previous one has been completed.



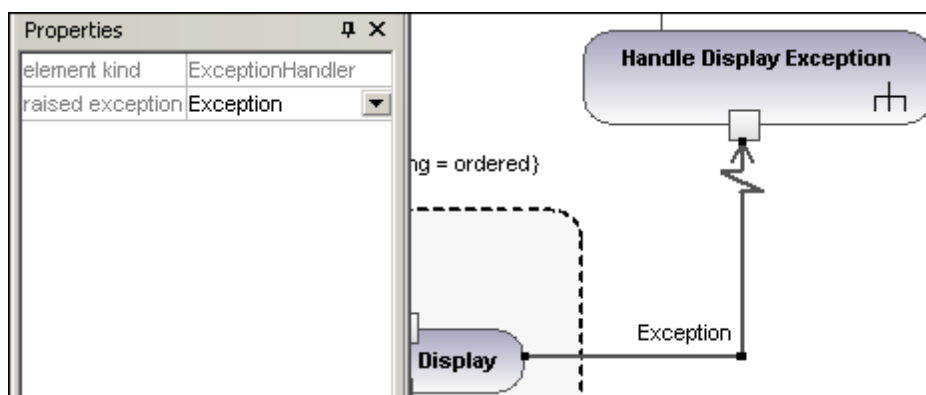
ObjectFlow

A Object Flow is an edge, i.e. an arrowed line, that connects two actions/object nodes, and starts an activity after the previous one has been completed. Objects or data can be passed along an Object Flow.



ExceptionHandler

An Exception Handler is an element that specifies what action is to be executed if a specified exception occurs during the execution of the protected node.

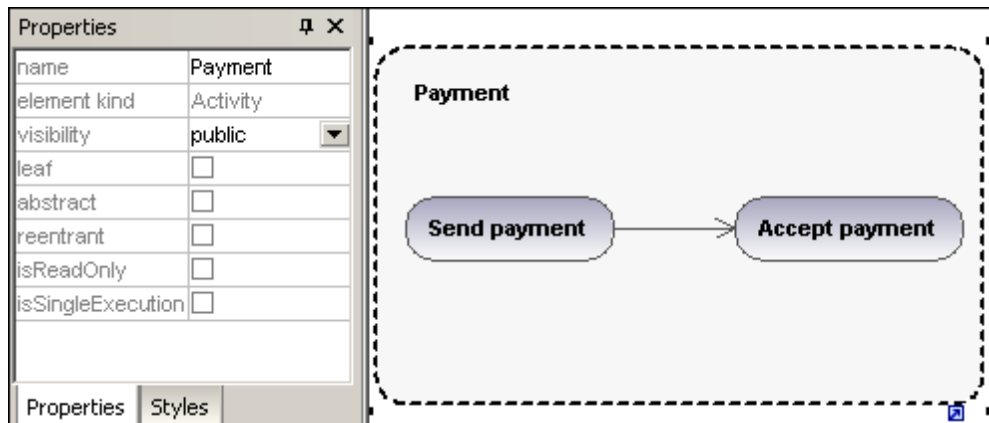


An Exception Handler can only be dropped on an Input Pin of an Action.



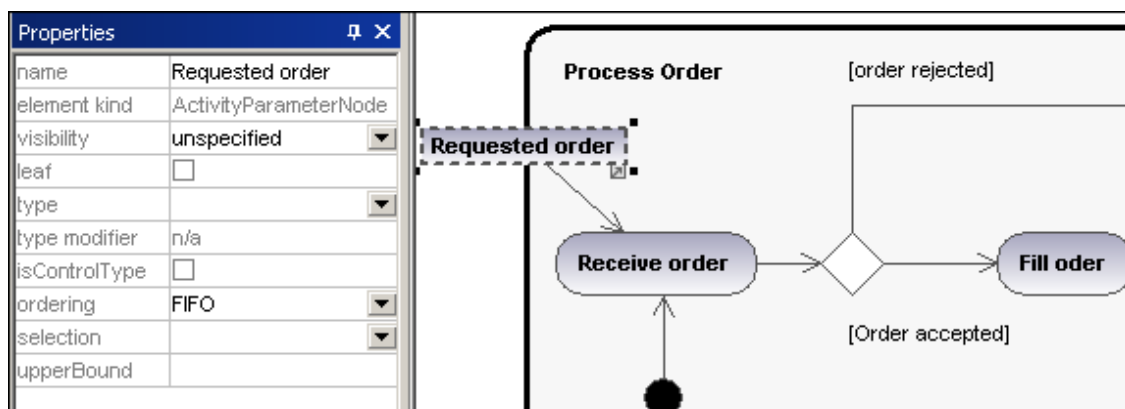
Activity

Inserts an Activity into the activity diagram.



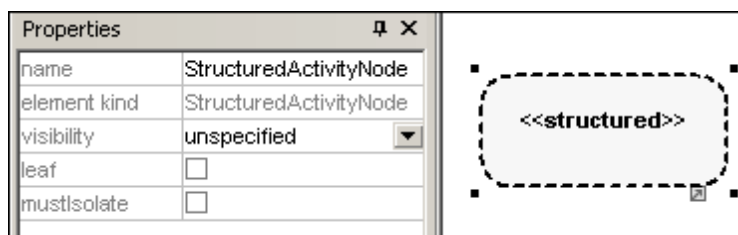
ActivityParameterNode

Inserts an Activity Parameter node onto an activity. Clicking anywhere in the activity places the parameter node on the activity boundary.



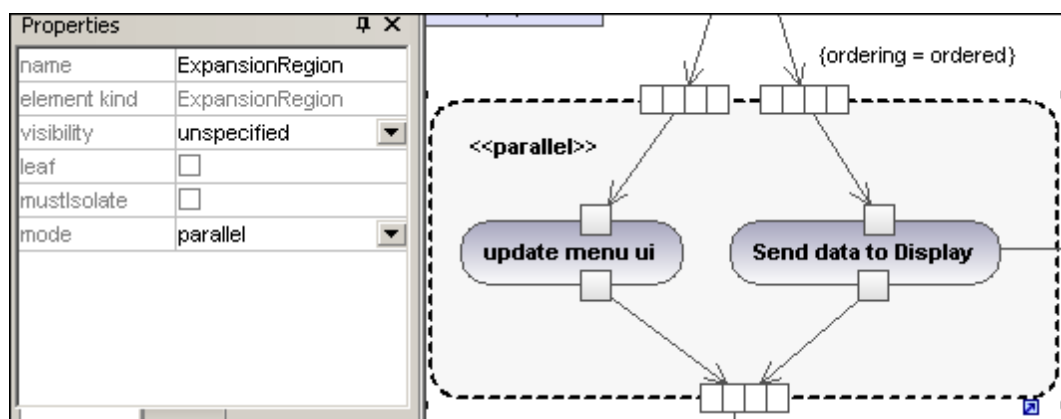
StructuredActivityNode

Inserts a Structured Activity Node which is a structured part of the activity, that is not shared with any other structured node.



ExpansionRegion

An expansion region is a region of an activity having explicit input and outputs (using ExpansionNodes). Each input is a collection of values.

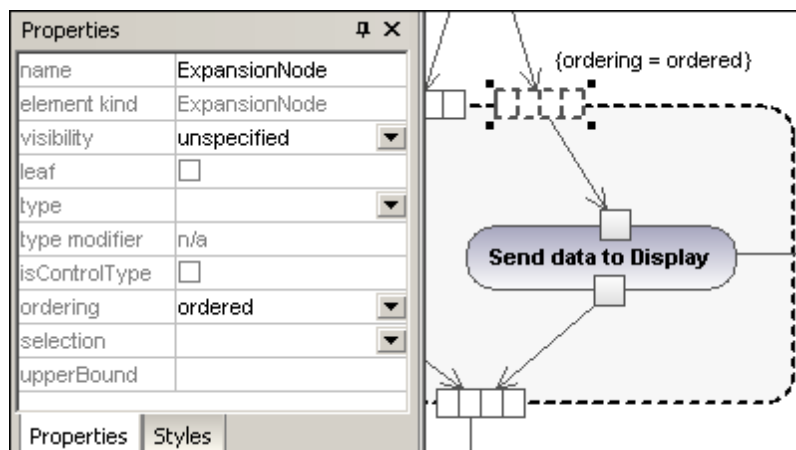


The expansion region mode is displayed as a keyword, and can be changed by clicking the "mode" combo box in the Properties tab. Available settings are: parallel, iterative, or stream.



ExpansionNode

Inserts an Expansion Node onto an Expansion Region. Expansion nodes are input and output nodes for the Expansion Region, where each input/output is a collection of values. The arrows into, or out of, the expansion region, determine the specific type of expansion node.



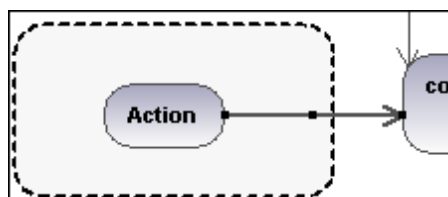
InterruptableActivityRegion

An interruptible region contains activity nodes. When a control flow leaves an interruptible region all flows and behaviors in the region are terminated.

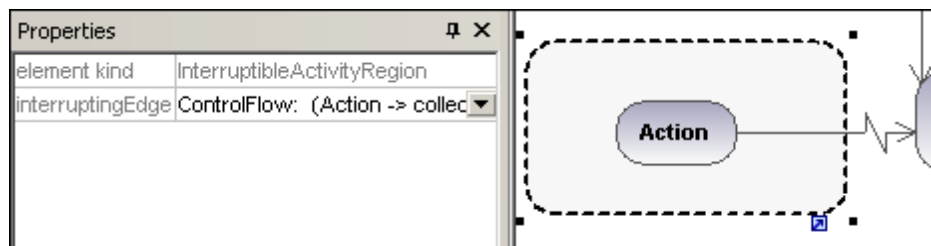
To add an interrupting edge:

Making sure that:

- an Action element is present in the InterruptableActivityRegion, as well as an outgoing Control Flow to another action:



1. Right click the Control Flow arrow, and select **New | InterruptingEdge**.



Please note:

You can also add an InterruptingEdge by clicking the InterruptibleActivityRegion, right clicking in the Properties window, and selecting Add InterruptingEdge from the pop-up menu.

10.1.2 State Machine Diagram

Altova web site:  [UML State Machine diagrams](#)

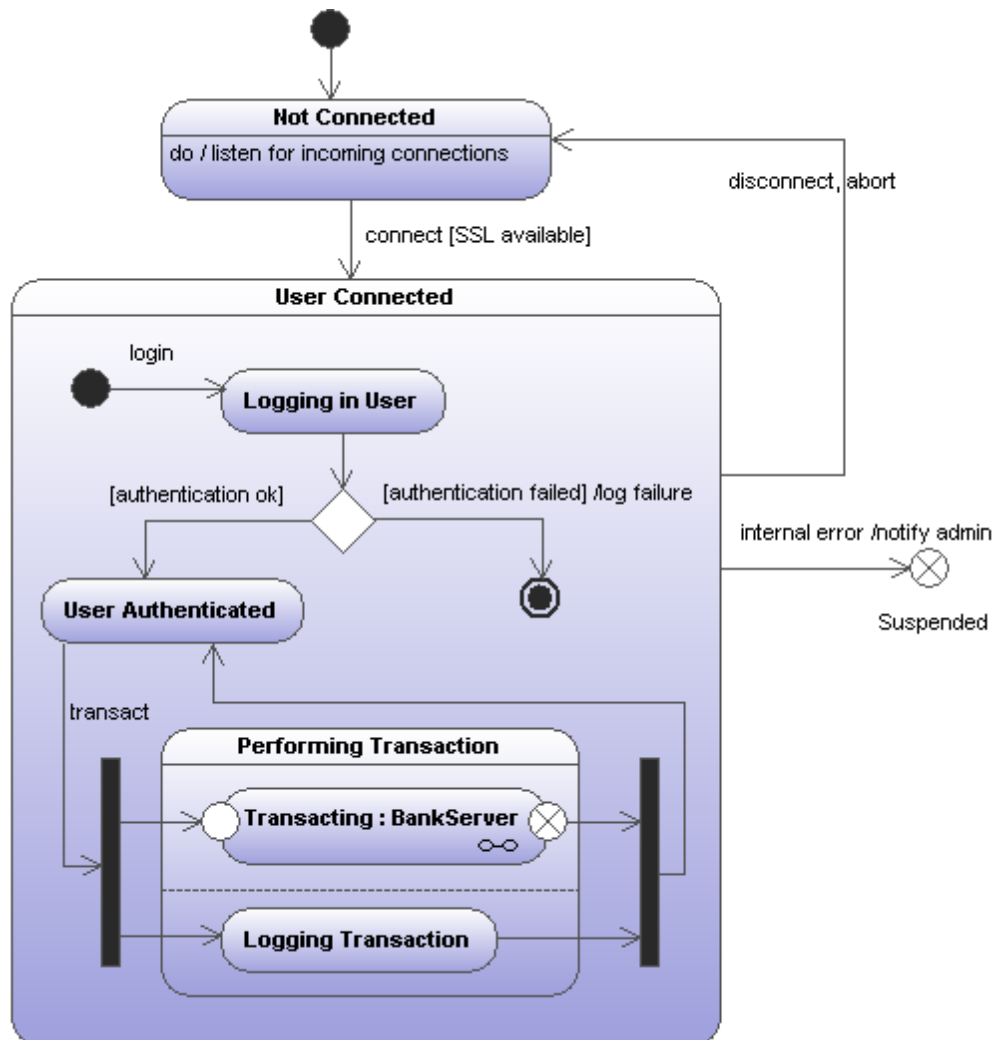
The State Machine Diagram models the behavior of a system by describing the various states an object may be in, and the transitions between those states. They are generally used to describe the behavior of an object spanning several use cases. A state machine can have any number of State Machine Diagrams (or State Diagrams) UModel.

Two types of processes can achieve this:

Actions, which are associated to **transitions**, are short-term processes that cannot be interrupted. E.g. an initial transition, **internal error /notify admin**.

State **Activities** (behaviors), which are associated to **states**, are longer-term processes that may be interrupted by other events. E.g. **listen for incoming connections**.

Please note that the State machine diagrams shown in the following section are available in the **Bank_MultiLanguage.ump** sample, in the ...**UModelExamples** folder supplied with UModel.



Inserting state machine diagram elements

Using the toolbar icons:

1. Click the specific state machine diagram icon in the State Machine Diagram toolbar.



2. Click in the State Diagram to insert the element. To insert multiple elements of the selected type, hold down the **Ctrl** key and click in the diagram window.

Dragging existing elements into the state machine diagram:

Most elements occurring in other state machine diagrams, can be inserted into an existing state machine.

1. Locate the element you want to insert in the Model Tree tab (you can use the search function text box, or press **Ctrl+F** to search for any element).
2. Drag the element(s) into the state diagram.

Creating states, activities and transitions

To add a simple state:

1. Click the **State** toolbar icon (), and then click inside the diagram.
2. Enter the name of the state and press **Enter** to confirm.

To add an activity to a state:

- Right-click the state element, select **New**, and then one of the entries from the context menu.

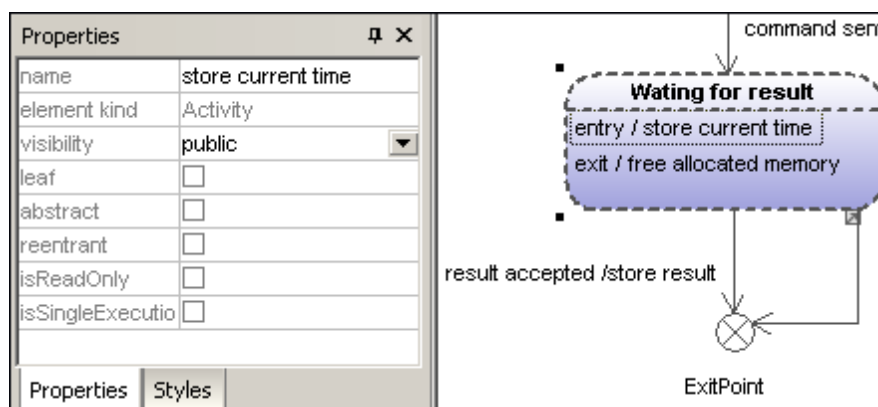


The **Entry**, **Exit**, and **Do** activities are associated with one of the following possible behaviors: "Activity", "Interaction", and "StateMachine". Therefore, the options available in the context menu are:

- Do: Activity
- Do: Interaction
- Do: StateMachine
- Entry: Activity
- Entry: Interaction
- Entry: StateMachine
- Exit: Activity
- Exit: Interaction
- Exit: StateMachine

These options originate in the UML specification. Namely, each of these internal actions are behaviors, and, in the UML specification, three classes derive from the "Behavior" class: Activity, StateMachine, and Interaction. In the generated code, it does not make a difference which particular behavior (Activity, StateMachine, or Interaction) has been selected.

You can select one action from the **Do**, **Entry** and **Exit** action categories. Activities are placed in their own compartment in the state element, though not in a separate region. The type of activity that you select is used as a prefix for the activity e.g. **entry / store current time**.

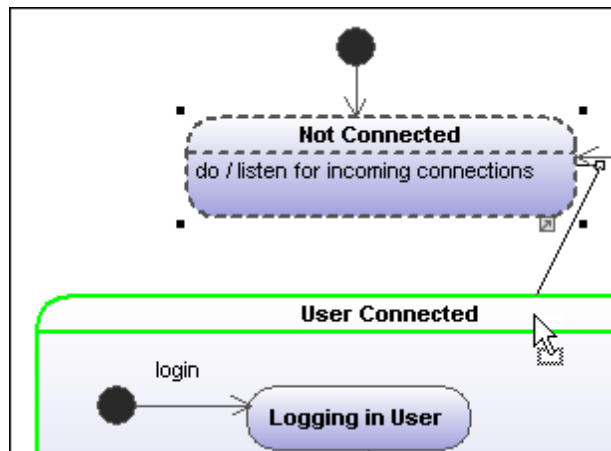


To delete an activity:

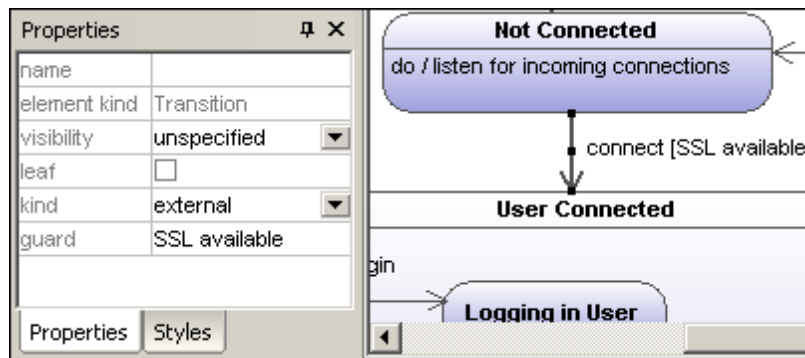
- Click the respective activity in the state element and press the Del. key.

To create a transition between two states:

1. Click the Transition handle of the source state (on the right of the element).
2. Drag-and-drop the transition arrow onto the target state.



The Transition properties are now visible in the Properties tab. Clicking the "kind" combo box, allows you to define the transition type: external, internal or local.



Transitions can have an event trigger, a guard condition and an action in the form **eventTrigger [guard condition] /activity**.

To automatically create operations from transitions:

Activating the "Toggle automatic creation of operations in target by typing operation names" icon

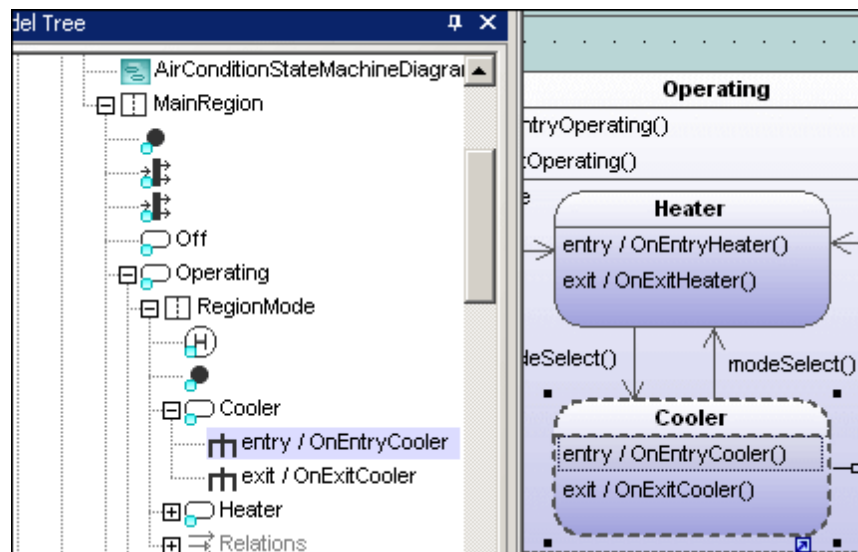


, automatically creates the corresponding operation in the referenced class, when creating a transition and entering a name e.g. myOperation().

Note: operations can only be created automatically when the state machine is inside a class or interface.

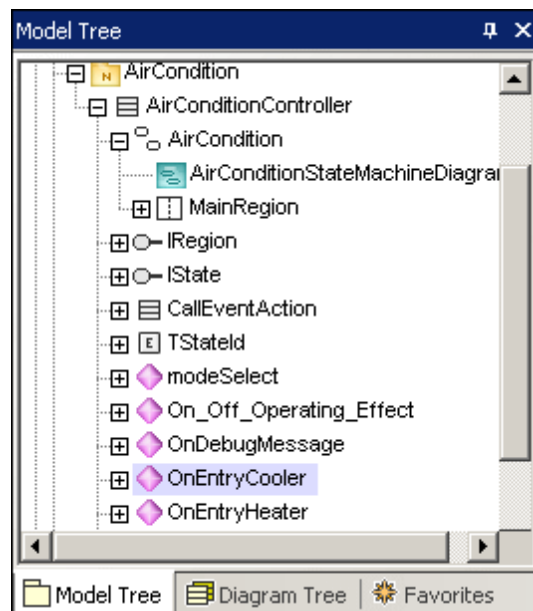
To automatically create operations from activities:

1. Right click the State and select the specific action/activity, e.g. New | Entry:Activity.
2. Enter the name of the activity making sure to finish with the open/close brackets "()", e.g. **entry / OnEntryCooler()**.



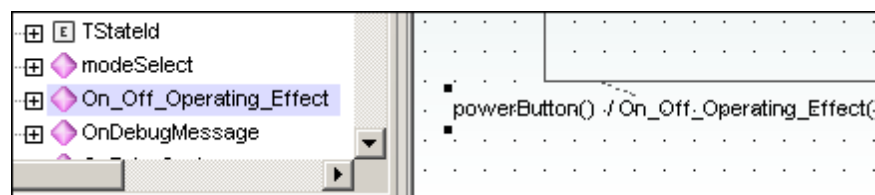
The new element is also visible in the Model Tree.

Scrolling down the Model Tree, you will notice that the OnEntryCooler operation has been added to the parent class AirConditionController.



Please note:

Operations are automatically added for: Do:Activity, Entry:Activity, Exit:Activity, as well as guard condition activities and effects (on transitions).



To create a transition trigger:

1. Right click a previously created transition (arrow).
2. Select **New | Trigger**.



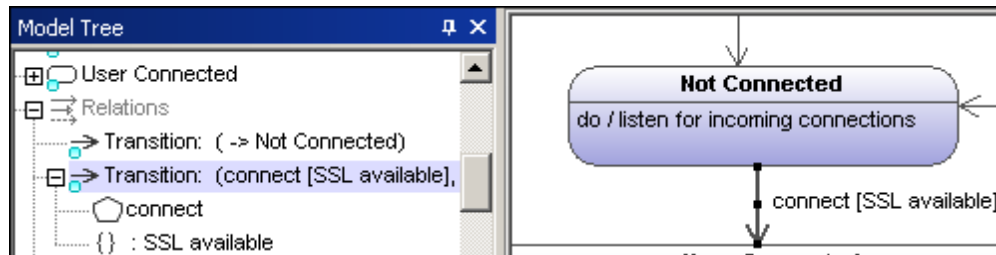
An "a" character appears in the transition label above the transition arrow, if it is the first trigger in the state diagram. Triggers are assigned default values of the form alphabetic letter, source state -> target state.

3. Double click the new character and enter the transition properties in the form **eventTrigger [guard condition] / activity**.

Transition property syntax; the text entered before the square brackets is the trigger, between brackets the guard condition, and after the slash, the activity. Manipulating this string automatically creates or deletes the respective elements in the Model Tree.

Please note:

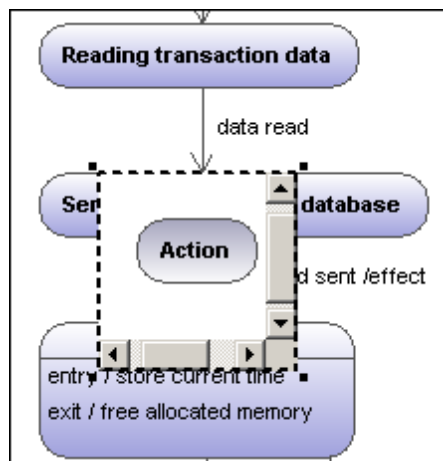
To see the individual transition properties, right click the transition (arrow) and select "Select in Model Tree". The event, activity and constraint elements are all shown below the selected transition.



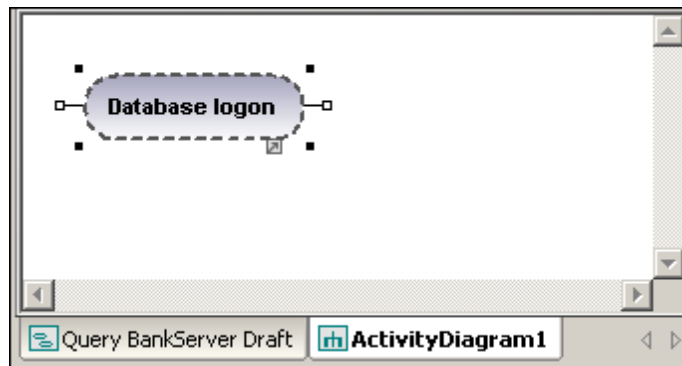
Adding an Activity diagram to a transition:

UModel has the unique capability of allowing you to add an Activity diagram to a transition, to describe the transition in more detail.

1. Right click a transition arrow in the diagram, and select **New | Activity Diagram**. This inserts an Activity diagram window into the diagram at the position of the transition arrow.
2. Click the inserted window to make it active. You can now use the scroll bars to scroll within the window.

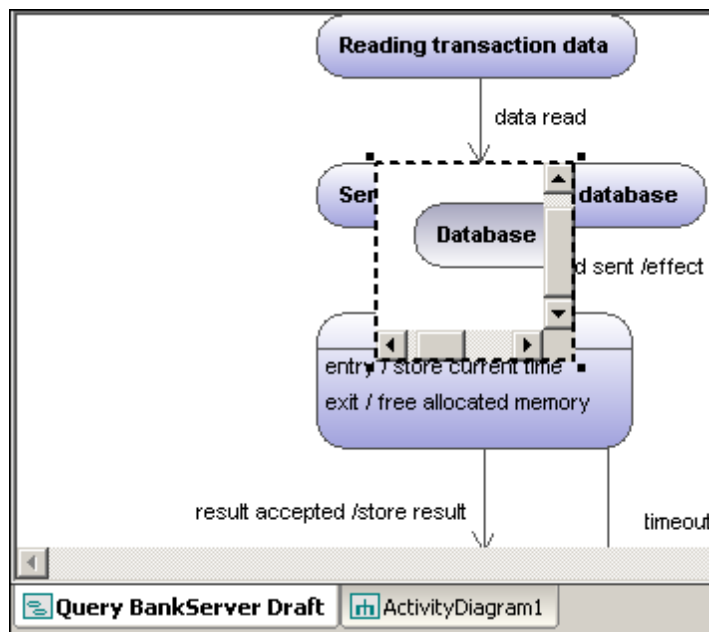


3. Double click the Action window to switch into the Activity diagram and further define the transition, e.g. change the Action name to "Database logon".

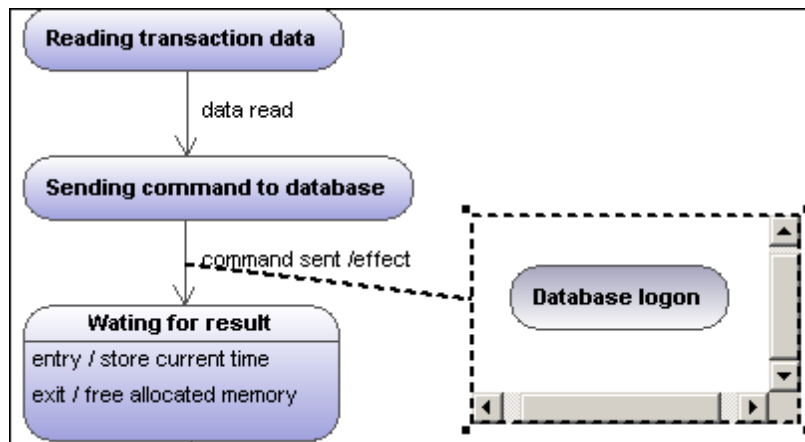


Note that a new Activity Diagram tab has now been added to the project. You can add any activity modeling elements to the diagram, please see "[Activity Diagram](#)" for more information.

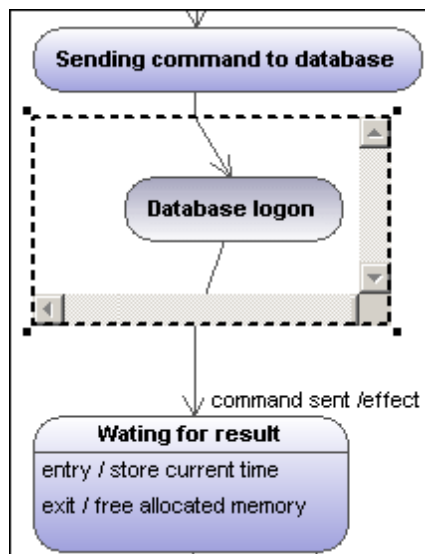
4. Click the State Machine Diagram tab to switch back to see the update transition.



5. Drag the Activity window to reposition it in the diagram, and click the resize handle if necessary.



Dragging the Activity window between the two states, displays the transition in and out of the activity.



Composite states



Composite state

This type of state contains a second compartment comprised of a single region. Any number of states may be placed within this region.

To add a region to a composite state:

1. Right click the composite state and select **New | Region** from the context menu. A new region is added to the state. Regions are divided by dashed lines.

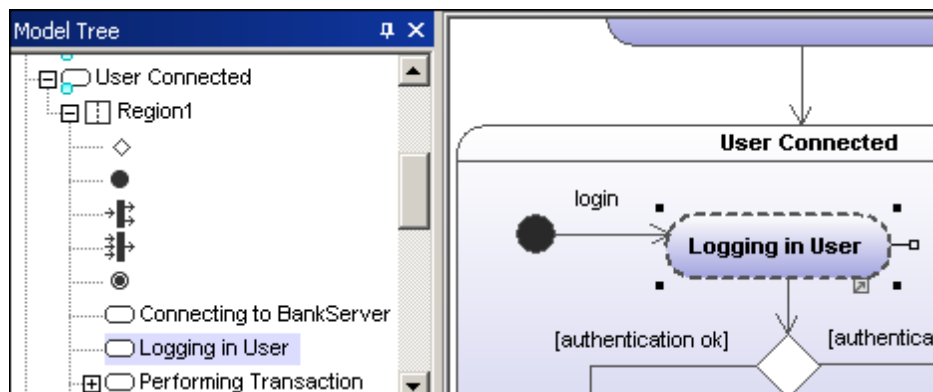
To delete a region:

1. Click the region you want to delete in the composite state and press the Del. key. Deleting a region of an orthogonal state reverts it back to a composite state; deleting the last region of a composite state changes it back to a simple state.

To place a state within a composite state:

1. Click the state element you want to insert (e.g. Logging in User), and drop it into the region compartment of the composite state.

The region compartment is highlighted when you can drop the element. The inserted element is now part of the region, and appears as a child element of the region in the Model Tree pane.



Moving the composite state moves all contained states along with it.



Orthogonal state

This type of state contains a second compartment comprised of two or more regions, where the separate regions indicate concurrency.

Right clicking a state and selecting **New | Region** allows you add new regions.



To show/hide region names:

Click the Styles tab, scroll to the "Show region names on states" entry, and select true/false.

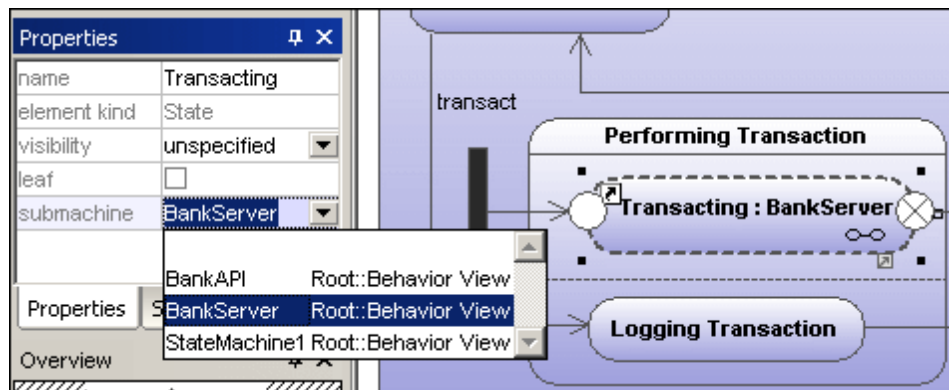


Submachine state

This state is used to hide details of a state machine. This state does not have any regions but is associated to a separate state machine.

To define a submachine state:

1. Having selected a state, click the **submachine** combo box in the Properties tab.
A list containing the currently defined state machines appears.
2. Select the state machine that you want this submachine to reference.

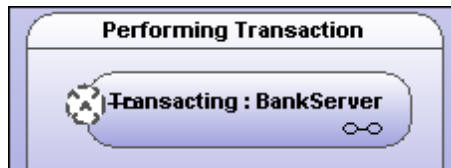


Please note that a hyperlink icon automatically appears in the submachine, clicking it opens the referenced state machine, BankServer in this case.

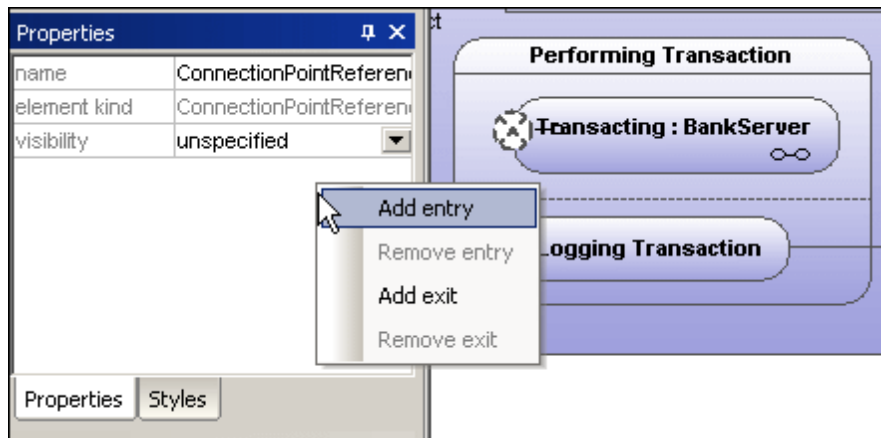
To add entry / exit points to a submachine state:

- The state which the point is connected to, must itself reference a submachine State Machine (visible in the Properties tab).
- This submachine must contain one or more Entry and Exit points

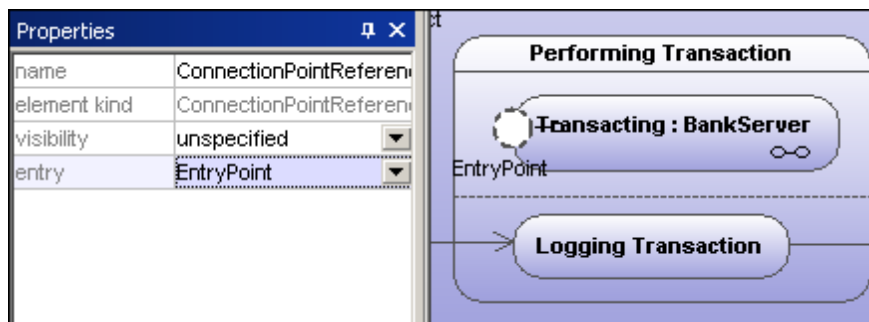
1. Click the **ConnectionPointReference** icon  in the title bar, then click the submachine state that you want to add the entry/exit point to.



2. Right click in the Properties tab and select Add entry. Please note that another Entry, or Exit Point has to exist elsewhere in the diagram to enable this pop-up menu.



This adds an EntryPoint row to the Properties tab, and changes the appearance of the ConnectionPointReference element.



3. Use the same method to insert an ExitPoint, by selecting "Add exit" from the context menu.

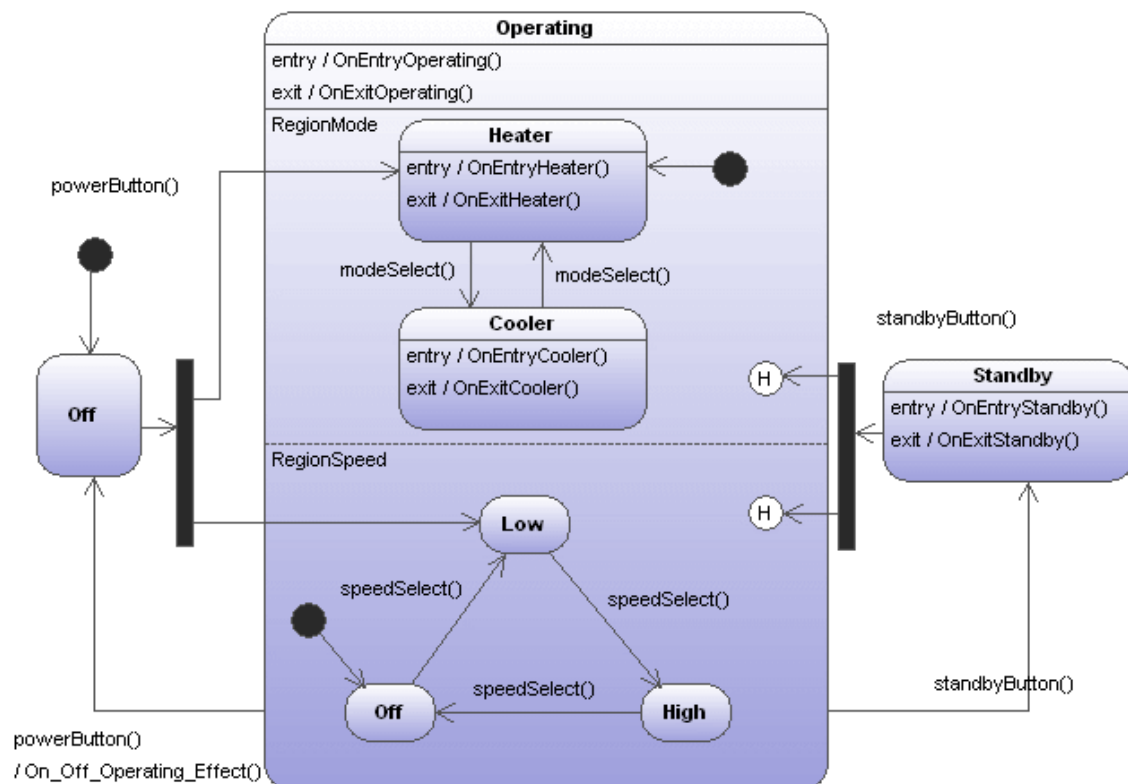
Code generation from State Machine diagrams

UModel is now able to generate executable code from State Machine diagrams (Java, VB.NET, or C#). Almost all of the State Machine diagram elements and features are supported:

- State
- CompositeState, with any hierarchical level
- OrthogonalState, with any number of regions
- Region
- InitialState
- FinalState
- Transition

- Guard
- Trigger
- Call-Event
- Fork
- Join
- Choice
- Junction
- DeepHistory
- ShallowHistory
- Entry/exit/do actions
- Effects

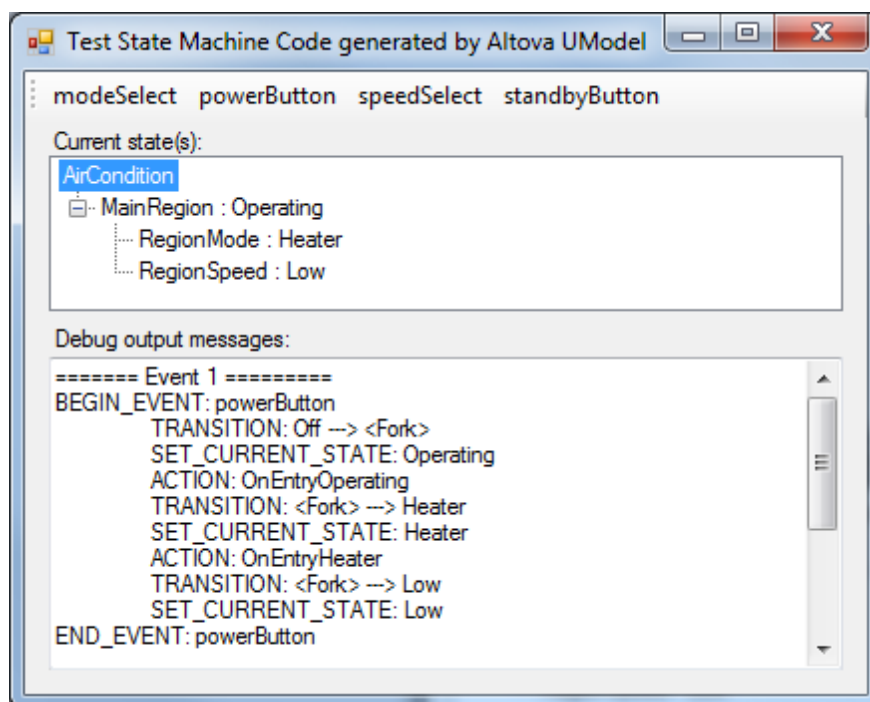
State Machine code generation is integrated into the "normal" round-trip engineering process. This means that State Machine code can be automatically updated on every forward-engineering process.



The screenshot above shows the AirCondition State Machine diagram which is available in the ..\StateMachineCodeGeneration directory under ...UModelExamples. A separate directory exists for each of the code generation languages supported by UModel, i.e. C#, Java, and VB.

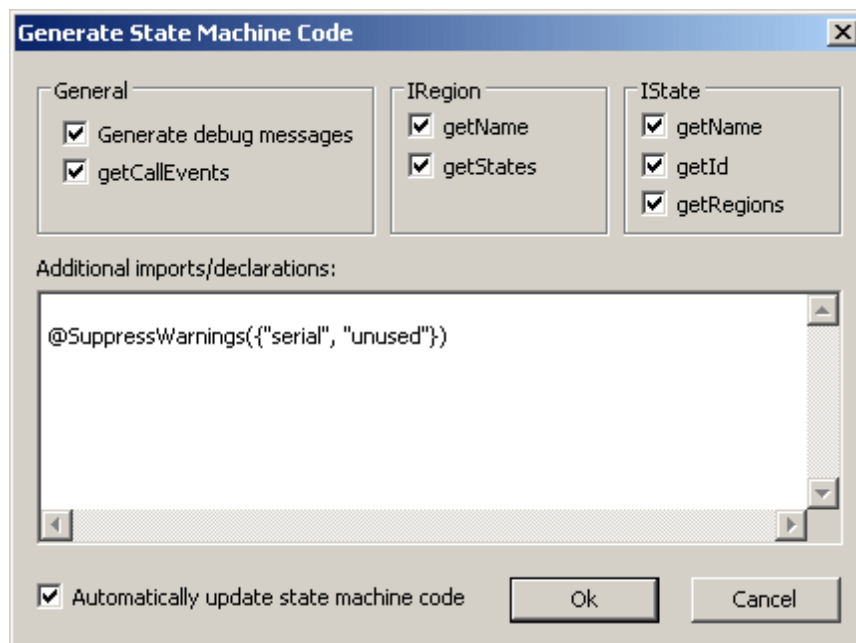
Each directory contains an AirCondition and Complex folder, which contains the respective UModel project, programming language project files, as well as the generated source files. The Complex.ump project file contains almost all of the modeling elements and functionality that UModel supports when generating code from State Machine diagrams.

Each directory also contains a test application, e.g. TestSTMAirCondition.sln for C#, allowing you to work with the generated source files immediately.



To generate code from a State Machine diagram:

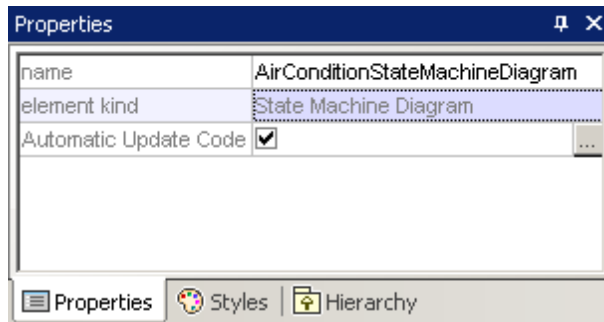
- **Right click** in the State Machine diagram and select "Generate State Machine code", or
- Select the menu option **Project | Generate State Machine Code**



The default settings are shown above. Click OK to generate the code.

State Machine code is automatically updated when you start the forward engineering process. You can however change this setting by clicking on the State Machine diagram background and clicking the "Automatic Update Code" check box.

Changes should not be made manually in the generated code, as these changes are not reflected in the State Machine diagram during the reverse-engineering process.



Clicking the  icon of the Automatic Update field, opens the Generate State Machine Code dialog box, allowing you to change the code generation settings.

Please note:

You can perform a syntax check on a State Machine diagram by right clicking the diagram and selecting **Check State Machine Syntax**.

Working with state machine code

The parent class of the state machine (i.e. the "controller class", or "context class") is the one, and only, "interface" between the state machine user and the state machine implementation.

The controller class provides methods which can be used from "outside" to change the states (e.g. after external events occur).

The state machine implementation however, calls controller class methods ("callbacks") to inform the state machine user about state changes (OnEntry, OnExit, ...), transition effects, and the possibility to override and implement methods for conditions (guards).

UModel can automatically create simple operations (without any parameter) for entry/exit/do behaviors, transition effects, ... when the corresponding option is turned on (also see [Creating states, activities and transitions](#)). These methods can be changed to whatever you want in UModel (add parameters, set them as abstract, etc.).

A state machine (i.e. its controller class) can be instantiated several times. All instances work independently of each other.

- The UML State machine execution is designed for the "Run-to-completion execution model".
- UML state machines assume that processing of each event is completed before the next event is processed.
- This also means no entry/exit/do action or transition effect may directly trigger a new transition/state change.

Initialization

- Every region of a state machine has to have an initial state.
- The code generated by UModel automatically initializes all regions of the state machine (or when the `Initialize()` method of the controller class is called).
- If OnEntry events are not wanted during initialization, you can call the `Initialize()` method manually and ignore OnEntry events during the startup.

Getting the current state(s)

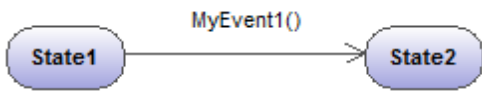
UModel supports composite states as well as orthogonal states, so there is not just one current state—every region (in any hierarchy level) can have one current state.

The AirCondition example shows how to walk through the regions to the current state(s):

```
TreeNode rootNode = m_CurrentStateTree.Nodes.Add(m_STM.getRootState().getName());
UpdateCurrentStateTree(m_STM.getRootState(), rootNode);

private void UpdateCurrentStateTree(AirCondition.AirConditionController.IState
state, TreeNode node)
{
    foreach (AirCondition.AirConditionController.IRegion r in state.getRegions())
    {
        TreeNode childNode = node.Nodes.Add(r.getName() + " : " +
        r.getCurrentState().getName());
        UpdateCurrentStateTree(r.getCurrentState(), childNode);
    }
}
```

Example 1 - a simple transition



The corresponding operation is automatically generated in UModel

TestController

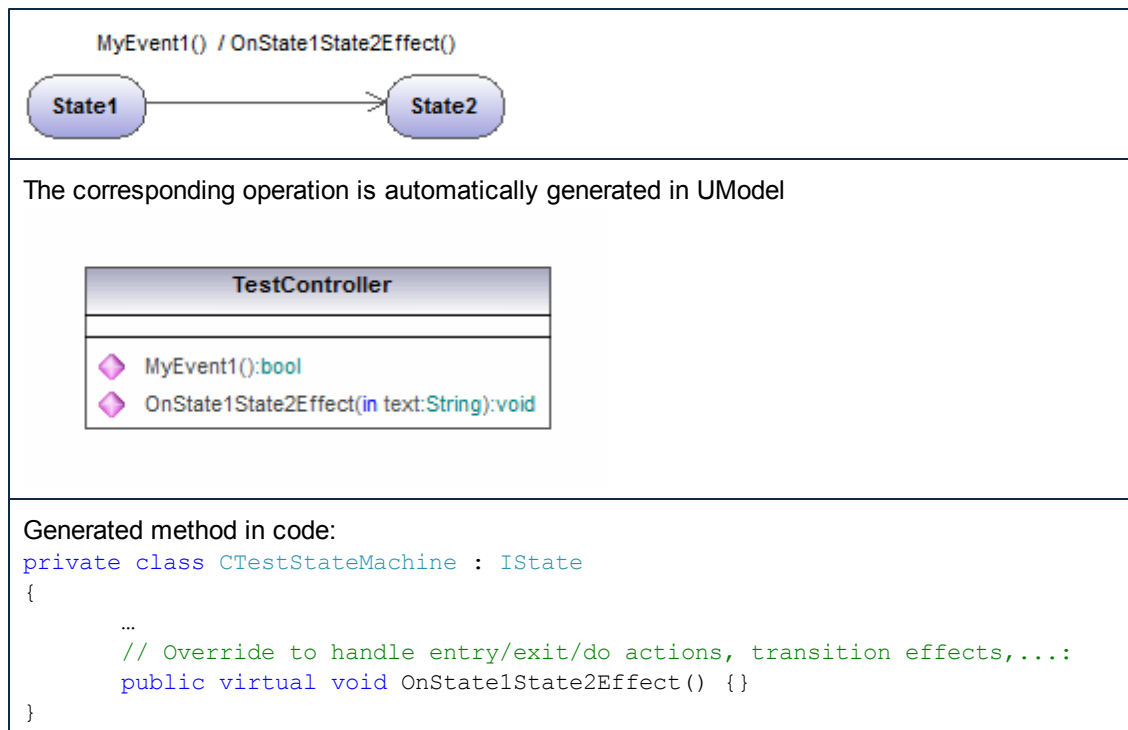
◆ `MyEvent1():bool`

Generated method in code:

```
private class CTestStateMachine : IState
{
    ...
    public bool MyEvent1()
    {
        ...
    }
}
```

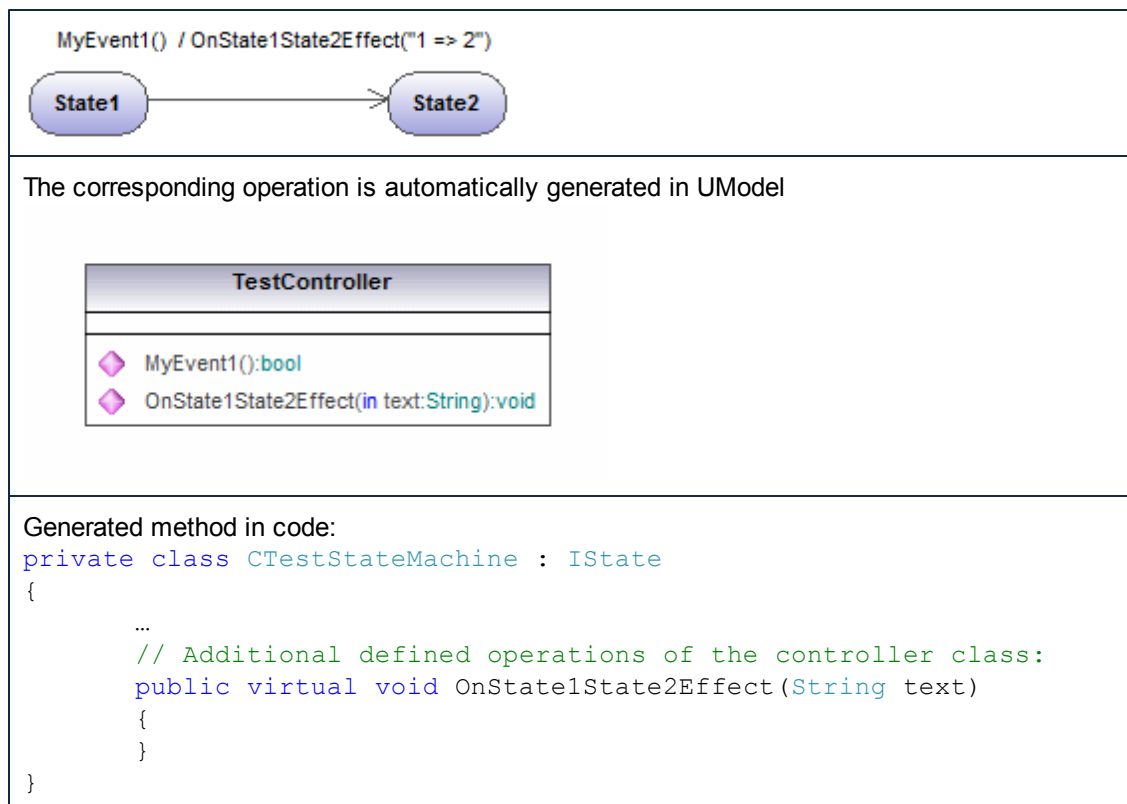
- The state machine user should call the generated method "MyEvent1" when the corresponding event occurs (outside the state machine).
- The return parameter of these event-methods provides information if the event caused a state change (i.e. if it had any effect on the state machine) or not. For example, if "State1" is active and event "MyEvent1()" occurs, the current state changes to "State2" and "MyEvent1()" returns true. If "State2" is active and "MyEvent1()" occurs, nothing changes in the state machine and MyEvent1() returns false.

Example 2 - a simple transition with an effect

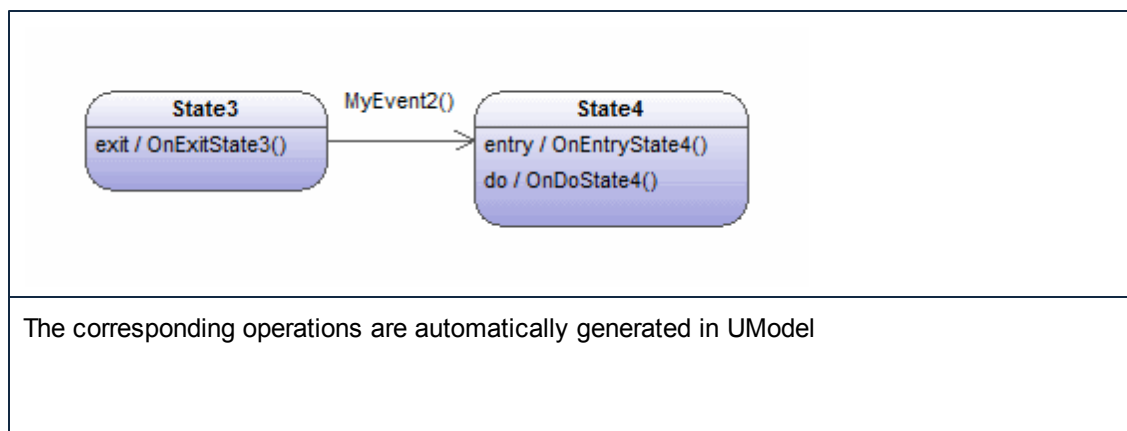


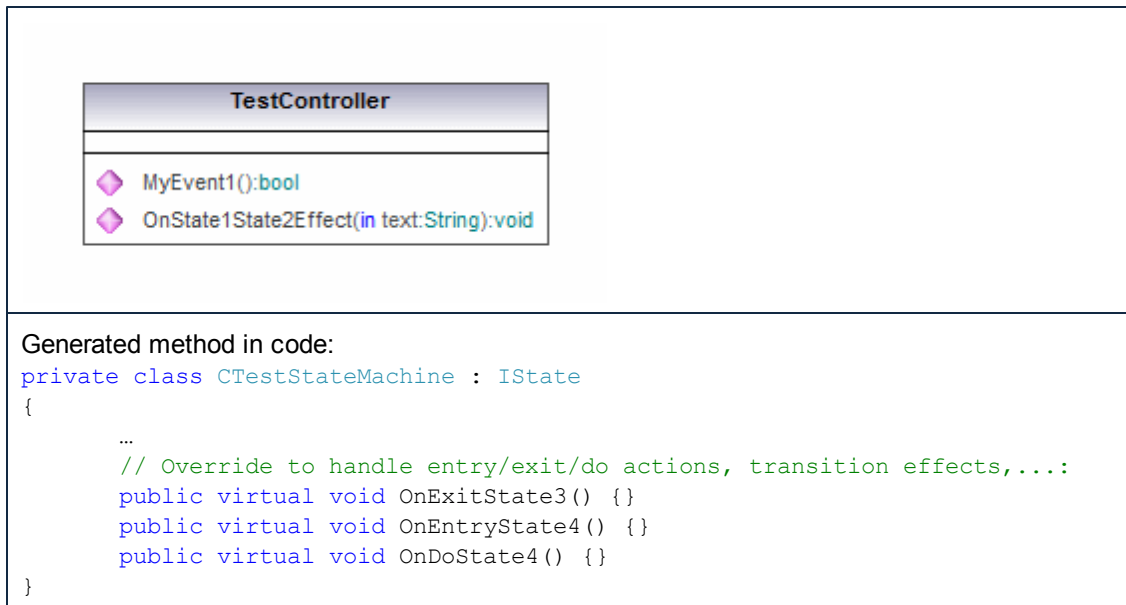
- "OnState1State2Effect()" will be called by the state machine implementation, whenever the transition between "State1" and "State2" is fired.
- To react to this effect, "OnState1State2Effect()" should be overridden in a derived class of "CTestStateMachine".
- "CTestStateMachine:: OnState1State2Effect()" can also be set to abstract, and you will get compiler errors until the method is overridden.
- When "OnState1State2Effect()" is not abstract, and the "Generate debug messages" option is active, UModel will generate following debug output:

```
// Override to handle entry/exit/do actions, transition
effects,...:
public virtual void OnState1State2Effect()
{OnDebugMessage("ACTION: OnState1State2Effect");}
```


Example 3 - a simple transition with an effect and parameter

- To effect operations (automatically created by UModel) parameters can be added manually (UModel cannot know the required type).
- In this sample, the parameter "text:String" has been added to the Effect method in TestController. A proper argument has to be specified when calling this method (here: "1 => 2").
- Another possibility would be: e.g. to call static methods ("MyStatic.OnState1State2Effect("1 => 2)"), or methods of singletons ("getSingleton().OnState1State2Effect("1 => 2)").

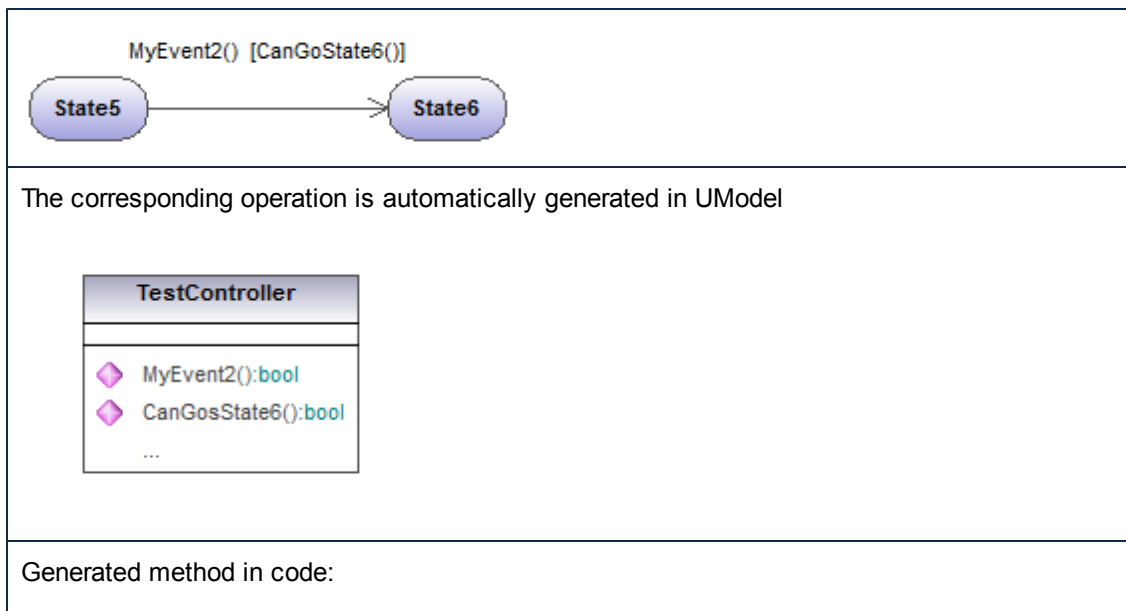
Example 4 - entry/exit/do actions



- States can have entry/exit/do behaviors. UModel automatically creates the corresponding operations to handle them.
- When "MyEvent2()" occurs in the sample above, the state machine implementation calls "OnExitState3()", if "MyEvent2" would have an Effect, it would be subsequently called, then "OnEntryState4" and "OnDoState4" would be called.
- Normally, these methods should be overridden. When they are not abstract and the "Generate debug messages" option is active, UModel provides default debug output as described in Example 2.
- These methods can also have parameters as shown in Example 3.

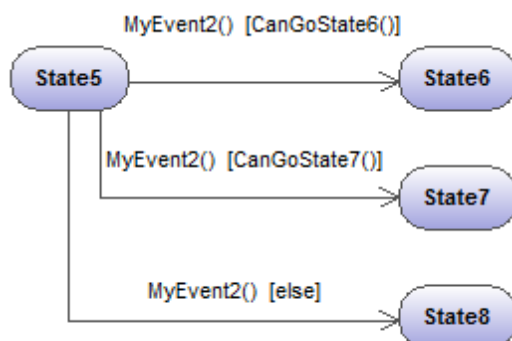
Example 5 - guards

Transitions can have guards, which determine if the transition really can fire.



```
private class CTestStateMachine : IState
{
    ...
    // Additional defined operations of the controller class:
    public virtual bool CanGoState6()
    {
        return true; // Override!
    }
}
```

- If "State5" is the active state and "MyEvent2" occurs, the state machine implementation will call "CanGoState6" and, depending on its result, the transition will fire or not.
- Normally, these methods should be overridden. When they are not abstract and the "Generate debug messages" option is active, UModel provides default debug output as described in Example 2.
- These methods also can have parameters as shown in Example 3.
- Multiple transitions with the same event, but having different guards, are possible. The order in which the different guards are polled is undefined. If a transition does not have a guard, or the guard is "else", it will be considered as the last (i.e., only when all other transition guards return false, will this one will fire). For example, in the diagram below, it is undefined whether `CanGoState6()` or `CanGoState7()` is called first. The third transition will only fire if `CanGoState6()` and `CanGoState7()` return false.



Additional constructs and functionality can be found in the **AirCondition.ump** and **Complex.ump** samples.

State Machine Diagram elements



InitialState (pseudostate)
The beginning of the process.



FinalState
The end of the sequence of processes.



EntryPoint (pseudostate)
The entry point of a state machine or composite state.



ExitPoint (pseudostate)

The exit point of a state machine or composite state.



Choice

This represents a dynamic conditional branch, where mutually exclusive guard triggers are evaluated (OR operation).



Junction (pseudostate)

This represents an end to the OR operation defined by the Choice element.



Terminate (pseudostate)

The halting of the execution of the state machine.



Fork (pseudostate)

Inserts a vertical Fork bar.

Used to divide sequences into concurrent subsequences.



Fork horizontal (pseudostate)

Inserts a horizontal Fork bar.

Used to divide sequences into concurrent subsequences.



Join (pseudostate)

Joins/merges previously defined subsequences. All activities have to be completed before progress can continue.



Join horizontal (pseudostate)

Joins/merges previously defined subsequences. All activities have to be completed before progress can continue.



DeepHistory

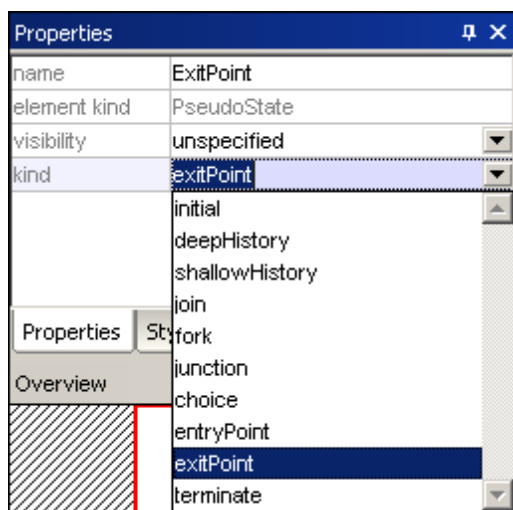
A pseudostate that restores the previously active state within a composite state.



ShallowHistory

A pseudostate that restores the initial state of a composite state.

All pseudostate elements can be changed to a different "type", by changing the **kind** combo box entry in the Properties tab.



ConnectionPointReference

A connection point reference represents a usage (as part of a submachine state) of an entry/exit point defined in the state machine reference by the submachine state.

To add Entry or Exit points to a connection point reference:

- The state which the point is connected to, must itself reference a submachine State Machine (visible in the Properties tab).
- This submachine must contain one or more Entry and Exit points



Transition

A direct relationship between two states. An object in the first state performs one or more actions and then enters the second state depending on an event and the fulfillment of any guard conditions.

Transitions have an event trigger, guard condition(s), an action (behavior), and a target state.

Supported event subelements are:

- ReceiveSignalEvent, SignalEvent, SendSignalEvent, ReceiveOperationEvent, SendOperationEvent and ChangeEvent.



Toggle automatic creation of operations in target by typing operation names

Activating the "Toggle automatic creation of operations in target by typing operation names" icon, automatically creates the corresponding operation in the referenced class, when creating a transition and entering a name myOperation().

Note: operations can only be created automatically when the state machine is inside a class or interface.

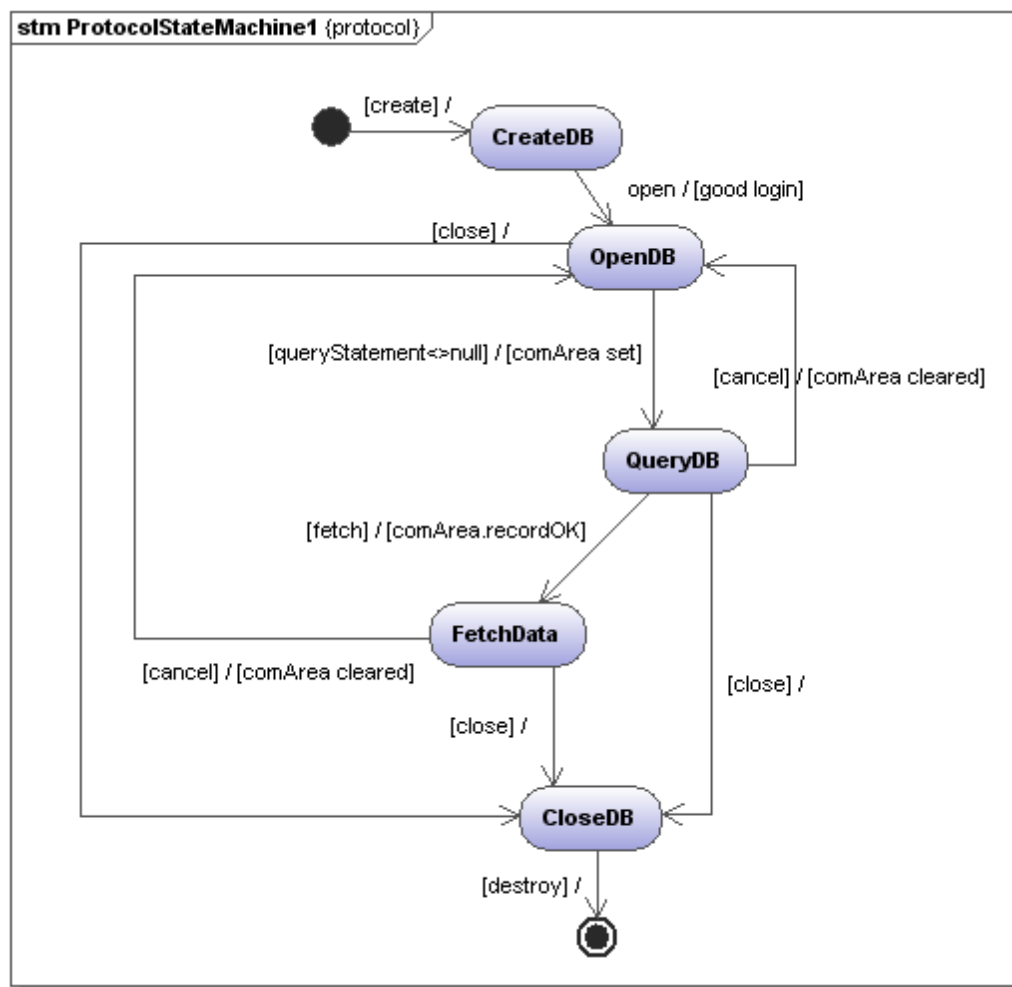
10.1.3 Protocol State Machine

Altova web site:  [UML Protocol State Machine diagrams](http://www.altova.com/uml/ProtocolStateMachineDiagrams.htm)

Protocol State Machines are used to show a **sequence** of events that an object responds to, without having to show the specific behavior. The required sequence of events, and the resulting changes in the state of the object, are modeled in this diagram.

Protocol State Machines are most often used to describe complex protocols, e.g. database access through a specific interface, or communication protocols such as TCP/IP.

Protocol State Machines are created in the same way as State Machine diagrams, but have fewer modeling elements. Protocol-Transitions between states can have pre- or post conditions which define what must be true for a transition to another state to occur, or what the resulting state must be, once the transition has taken place.



Inserting Protocol State Machine elements



Using the toolbar icons:

1. Click the Protocol State Machine icon in the toolbar.
2. Click in the Protocol State Machine Diagram to insert the element. To insert multiple elements of the selected type, hold down the **Ctrl** key and click in the diagram window.

Dragging existing elements into the Protocol State Machine diagram:

Most elements occurring in other Protocol State Machine diagrams, can be inserted into an existing diagram.

1. Locate the element you want to insert in the Model Tree tab (you can use the search function text box, or press **Ctrl+F** to search for any element).
2. Drag the element(s) into the Protocol State Machine diagram.

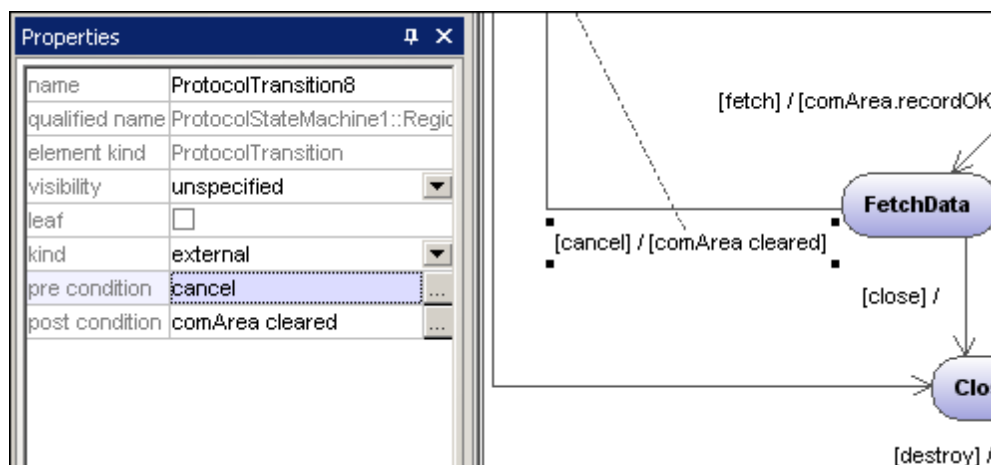
To insert a simple state:

1. Click the State icon  in the icon bar and click in the Protocol State Machine diagram to insert it.
2. Enter the name of the state and press Enter to confirm.
Simple states do not have any regions or any other type of substructure.

To create a Protocol Transition between two states:

1. Click the Transition handle of the source state (on the right of the element), or use the Protocol Transition icon in the icon bar.
2. Drag-and-drop the transition arrow onto the target state.
The text cursor is automatically set for you to enter the pre and/or post condition. Please make sure to use the square brackets `[]` and slash character when entering the conditions directly.

Entering the pre/post conditions in the Properties window, automatically inserts the square brackets and slash character in the diagram.

**To create and insert composite state elements and submachine states:**

- Please see [Composite states](#)

Protocol State Machine Diagram elements



State

A simple state element with one compartment.



Composite state

This type of state contains a second compartment comprised of a single region. Any number of states may be placed within this region.



Orthogonal state

This type of state contains a second compartment comprised of two or more regions, where the separate regions indicate concurrency.

Right clicking a state and selecting **New | Region** allows you add new regions.



Submachine state

This state is used to hide details of a state machine. This state does not have any regions but is associated to a separate state machine.



InitialState (pseudostate)

The beginning of the process.



FinalState

The end of the sequence of processes.



EntryPoint (pseudostate)

The entry point of a state machine or composite state.



ExitPoint (pseudostate)

The exit point of a state machine or composite state.



Choice

This represents a dynamic conditional branch, where mutually exclusive guard triggers are evaluated (OR operation).



Junction (pseudostate)

This represents an end to the OR operation defined by the Choice element.



Terminate (pseudostate)

The halting of the execution of the state machine.



Fork (pseudostate)

Inserts a vertical Fork bar.

Used to divide sequences into concurrent subsequences.



Fork horizontal (pseudostate)

Inserts a horizontal Fork bar.

Used to divide sequences into concurrent subsequences.



Join (pseudostate)

Joins/merges previously defined subsequences. All activities have to be completed before progress can continue.



Join horizontal (pseudostate)

Joins/merges previously defined subsequences. All activities have to be completed before progress can continue.



ConnectionPointReference

A connection point reference represents a usage (as part of a submachine state) of an entry/exit point defined in the state machine reference by the submachine state.

To add Entry or Exit points to a connection point reference:

- The state which the point is connected to, must itself reference a submachine State Machine (visible in the Properties tab).
- This submachine must contain one or more Entry and Exit points



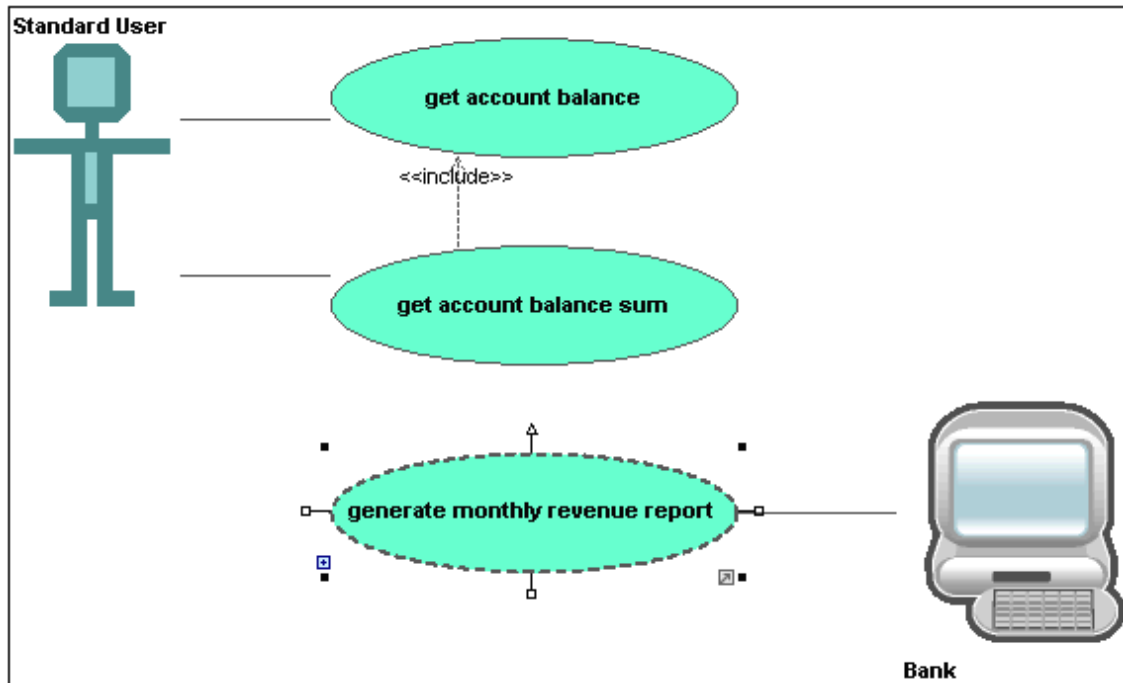
Protocol Transition

A direct relationship between two states. An object in the first state performs one or more operations and then enters the second state depending on an event and the fulfillment of any pre- or post conditions.

Please see [Inserting Protocol State Machine elements](#) for more information.

10.1.4 Use Case Diagram

Please see the [Use Cases](#) section in the tutorial for more information on how to add use case elements to the diagram.



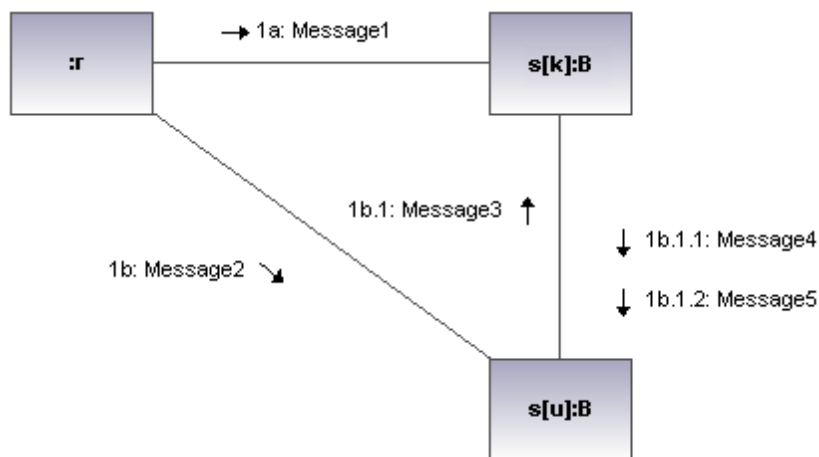
10.1.5 Communication Diagram

Altova web site:  [UML Communication diagrams](#)

Communication diagrams display the interactions i.e. message flows, between objects at run-time, and show the relationships between the interacting objects. Basically, they model the dynamic behavior of use cases.

Communication diagrams are designed in the same way as sequence diagrams, except that the notation is laid out in a different format. Message numbering is used to indicate message sequence and nesting.

UModel allows you to generate Communication diagrams from Sequence diagrams and vice versa, in one simple action see "[Generating Sequence diagrams](#)" for more information.



Inserting Communication Diagram elements

Using the toolbar icons:

1. Click the specific communication icon in the Communication Diagram toolbar.



2. Click in the Communication diagram to insert the element. To insert multiple elements of the selected type, hold down the **Ctrl** key and click in the diagram window.

Dragging existing elements into the Communication Diagram:

Elements occurring in other diagrams, e.g. classes, can be inserted into a Communication diagram.

1. Locate the element you want to insert in the Model Tree tab (you can use the search function text box, or press **Ctrl+F** to search for any element).
2. Drag the element(s) into the Communication diagram.



Lifeline

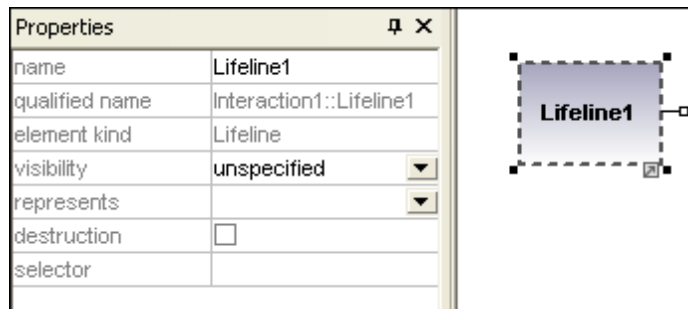
The lifeline element is an individual participant in an interaction. UModel allows you to insert other

elements into the sequence diagram, e.g. classes. Each of these elements then appear as a new lifeline. You can redefine the lifeline colors/gradient using the "Header Gradient" combo boxes in the Styles tab.

To create a **multiline** lifeline, press **Ctrl+Enter** to create a new line.

To insert a Communication lifeline:

1. Click the Lifeline icon in the title bar, then click in the Communication diagram to insert it.



2. Enter the lifeline name to change it from the default name, Lifeline1, if necessary.

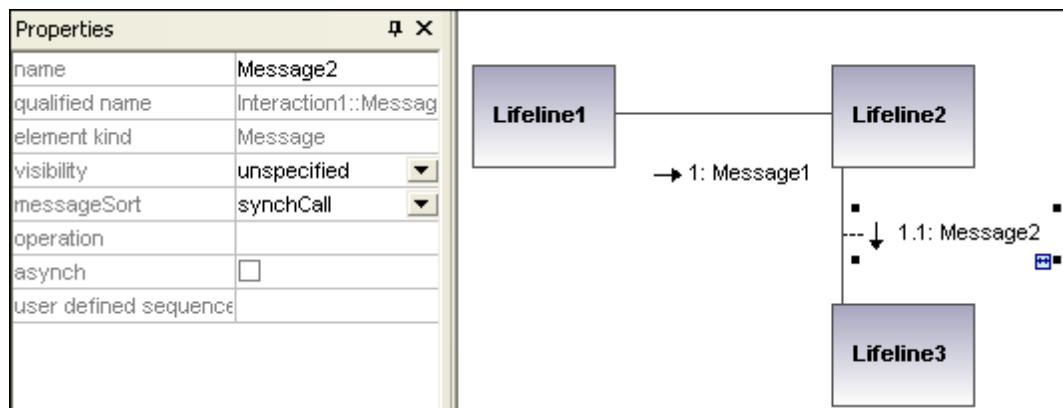
Messages

A Message is a modeling element that defines a specific kind of communication in an interaction. A communication can be e.g. raising a signal, invoking an Operation, creating or destroying an instance. The message specifies the type of communication as well as the sender and the receiver.



To insert a message:

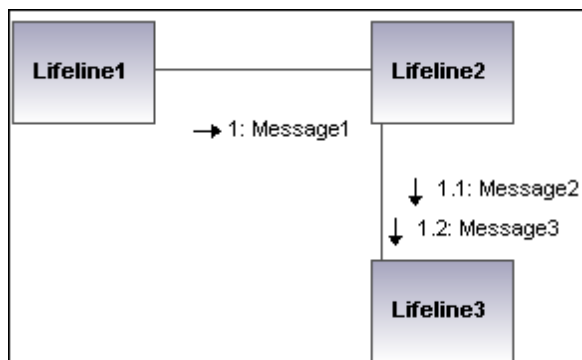
1. Click the specific message icon in the toolbar.
 2. Drag and drop the message line onto the receiver objects.
- Lifelines are highlighted when the message can be dropped.



Note: holding down the Ctrl key allows you to insert a message with each click.

To insert additional messages:

1. Right click an existing communication link and select **New | Message**.



- The direction in which you drag the arrow defines the message direction. Reply messages can point in either direction.
- Having clicked a message icon and holding down **Ctrl** allows you to insert multiple messages by repeatedly clicking and dragging in the diagram tab.

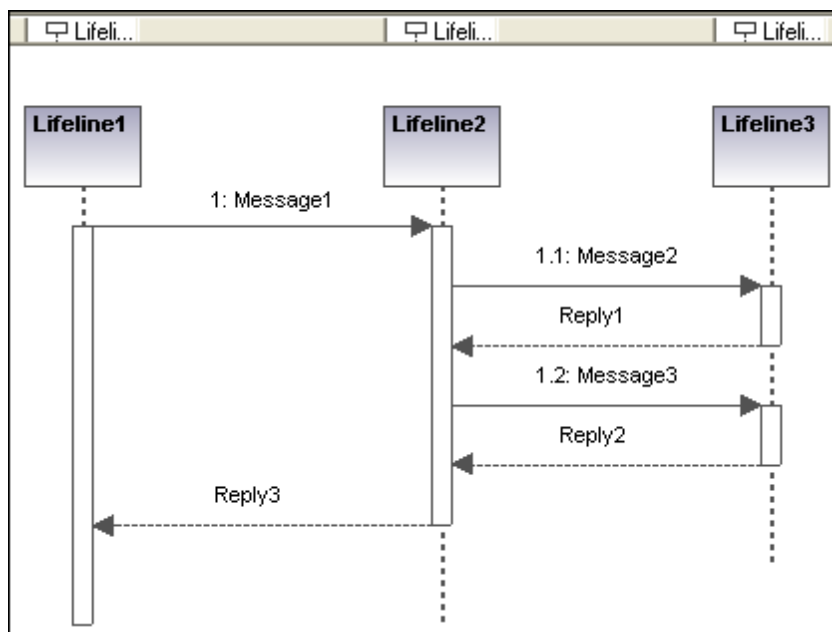
Message numbering

The Communication diagram uses the decimal numbering notation, which makes it easy to see the hierarchical structure of the messages in the diagram. The sequence is a dot-separated list of sequence numbers followed by a colon and the message name.

Generating Sequence diagrams from Communication diagrams:

UModel allows you to generate Communication diagrams from Sequence diagrams and vice versa, in one simple action:

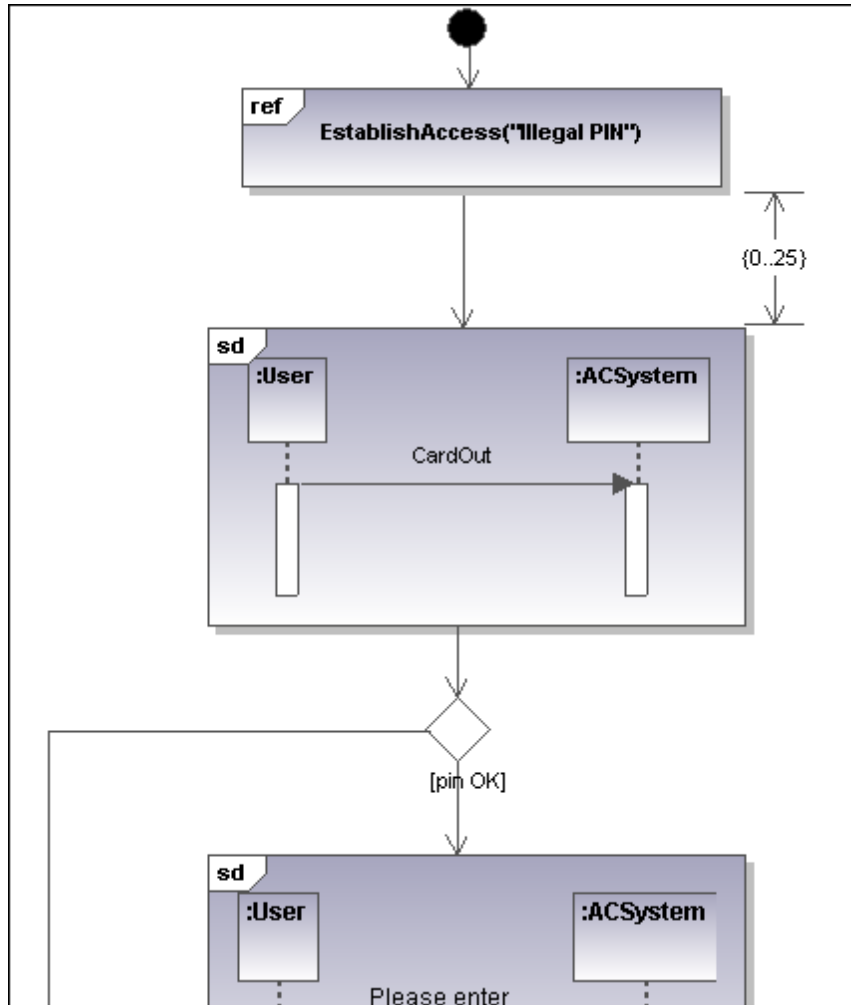
- Right click anywhere in a Communication diagram and select **Generate Sequence Diagram** from the context menu.



10.1.6 Interaction Overview Diagram

Altova web site:  [UML Interaction Overview diagrams](http://www.altova.com/uml/interaction-overview-diagrams)

Interaction Overview Diagrams are a variant of Activity diagrams and give an overview of the interaction between other interaction diagrams such as Sequence, Activity, Communication, or Timing diagrams. The method of constructing a diagram is similar to that of Activity diagram and uses the same modeling elements: start/end points, forks, joins etc.



Two types of interaction elements are used instead of activity elements: Interaction elements and Interaction use elements.

Interaction elements are displayed as iconized versions of a Sequence, Communication, Timing, or Interaction Overview diagram, enclosed in a frame with the "SD" keyword displayed in the top-left frame title space.

Interaction occurrence elements are references to existing Interaction diagrams with "Ref" enclosed in the frame's title space, and the occurrence's name in the frame.

Inserting Interaction Overview elements

Using the toolbar icons:

1. Click the specific icon in the Interaction Overview Diagram toolbar.



2. Click in the diagram to insert the element. To insert multiple elements of the selected type, hold down the **Ctrl** key and click in the diagram window.

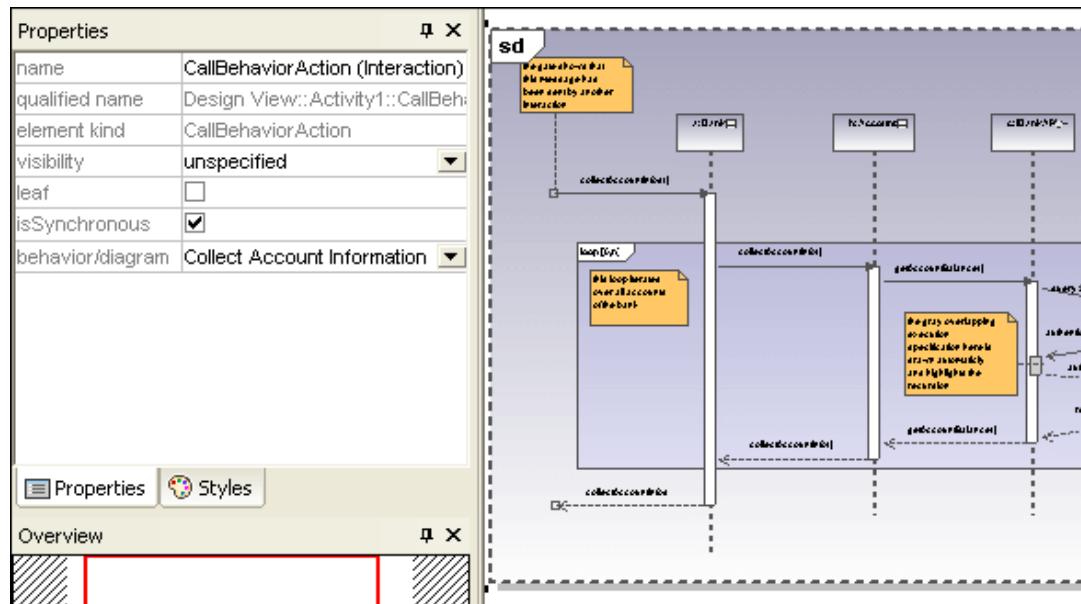
Dragging existing elements into the Interaction Overview Diagram:

Elements occurring in other diagrams, e.g. Sequence, Activity, Communication, or Timing diagrams can be inserted into a Interaction Overview diagram.

1. Locate the element you want to insert in the Model Tree tab (you can use the search function text box, or press Ctrl+F, to search for any element).
2. Drag the element(s) into the diagram.

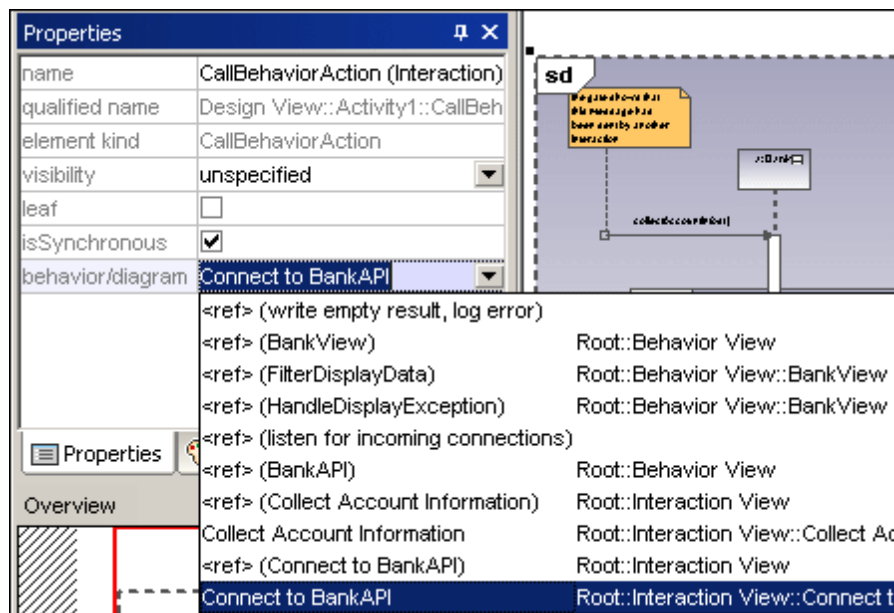
Inserting an Interaction element:

1. Click the CallBehaviorAction (Interaction) icon  in the icon bar, and click in the Interaction Overview diagram to insert it.

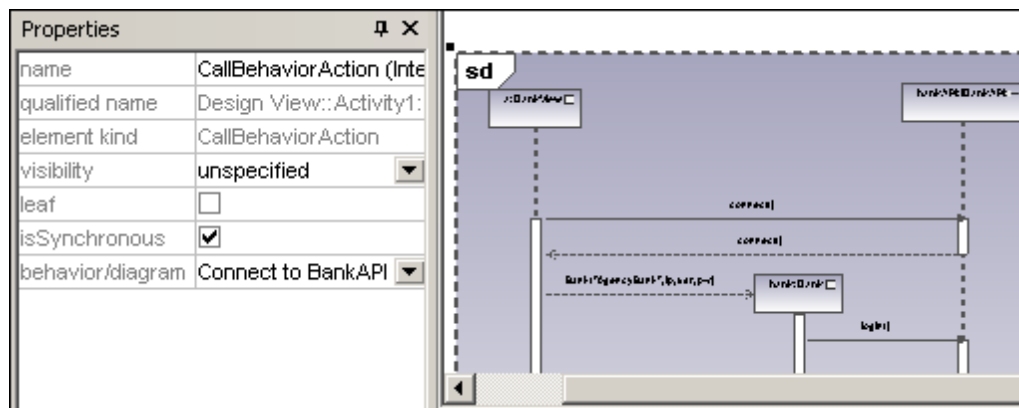


The Collect Account Information sequence diagram is automatically inserted if you are using the Bank_Multilanguage.ump example file from the ...\\UModelExamples folder. The first sequence diagram, found in the model tree, is selected per default.

2. To change the default interaction element: Click the **behavior/diagram** combo box in the Properties tab.
A list of all the possible elements that can be inserted is presented.

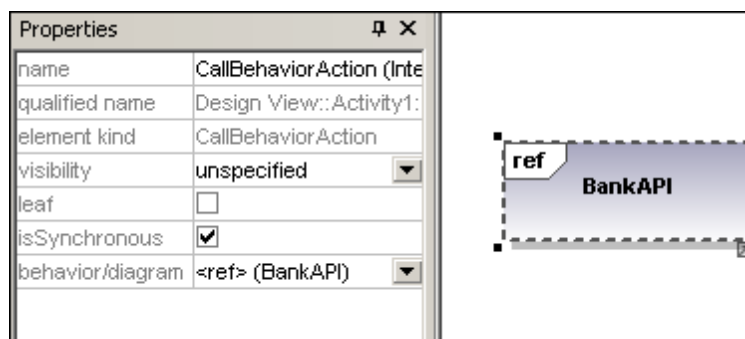


3. Click the element you want to insert to e.g. Connect to BankAPI.



As this is also a sequence diagram, the Interaction element appears as an iconized version of the sequence diagram.

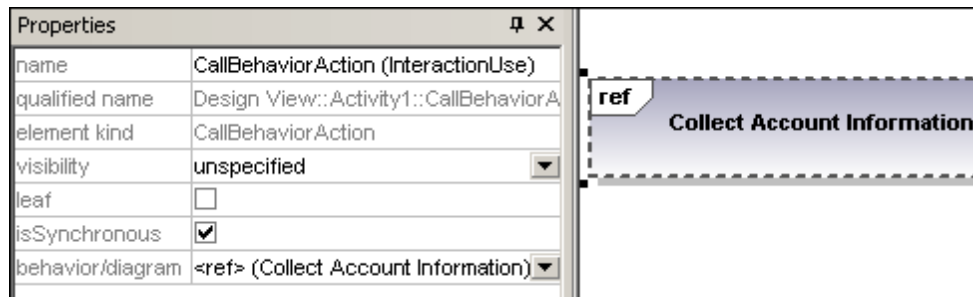
If you select **<ref> BankAPI**, then the Interaction element occurrence is displayed.



Inserting an Interaction element occurrence:

1. Click the CallBehaviorAction (InteractionUse) icon  in the icon bar, and click in the Interaction Overview diagram to insert it.

Collect Account Information is automatically inserted as a Interaction occurrence element, if you are using the Bank_Multilanguage.ump example file from the ... \UModelExamples folder. The first existing sequence diagram is selected per default.



- To change the Interaction element: double click the **behavior** combo box in the Properties tab.
A list of all the possible elements that can be inserted is presented.
- Select the occurrence you want to insert.
Note that all elements inserted using this method appear in the form shown in the screenshot above i.e. with "**ref**" in the frame's title space.



DecisionNode

Inserts a Decision Node which has a single incoming transition and multiple outgoing guarded transitions. Please see "[Creating a branch](#)" for more information.



MergeNode

Inserts a Merge Node which merges multiple alternate transitions defined by the Decision Node. The Merge Node does not synchronize concurrent processes, but selects one of the processes.



InitialNode

The beginning of the activity process. An interaction can have more than one initial node.



ActivityFinalNode

The end of the interaction process. An interaction can have more that one final node, all flows stop when the "first" final node is encountered.



ForkNode

Inserts a vertical Fork node.
Used to divide flows into multiple concurrent flows.



ForkNode (Horizontal)

Inserts a horizontal Fork node.
Used to divide flows into multiple concurrent flows.



JoinNode

Inserts a vertical Fork node.

A Join node synchronizes multiple flows defined by the Fork node.



Join Node (horizontal)

Inserts a horizontal Fork node.

A Join node synchronizes multiple flows defined by the Fork node.



AddDurationConstraint

A Duration defines a ValueSpecification that denotes a duration in time between a start and endpoint. A duration is often an expression representing the number of clock ticks, which may elapse during this duration.



ControlFlow

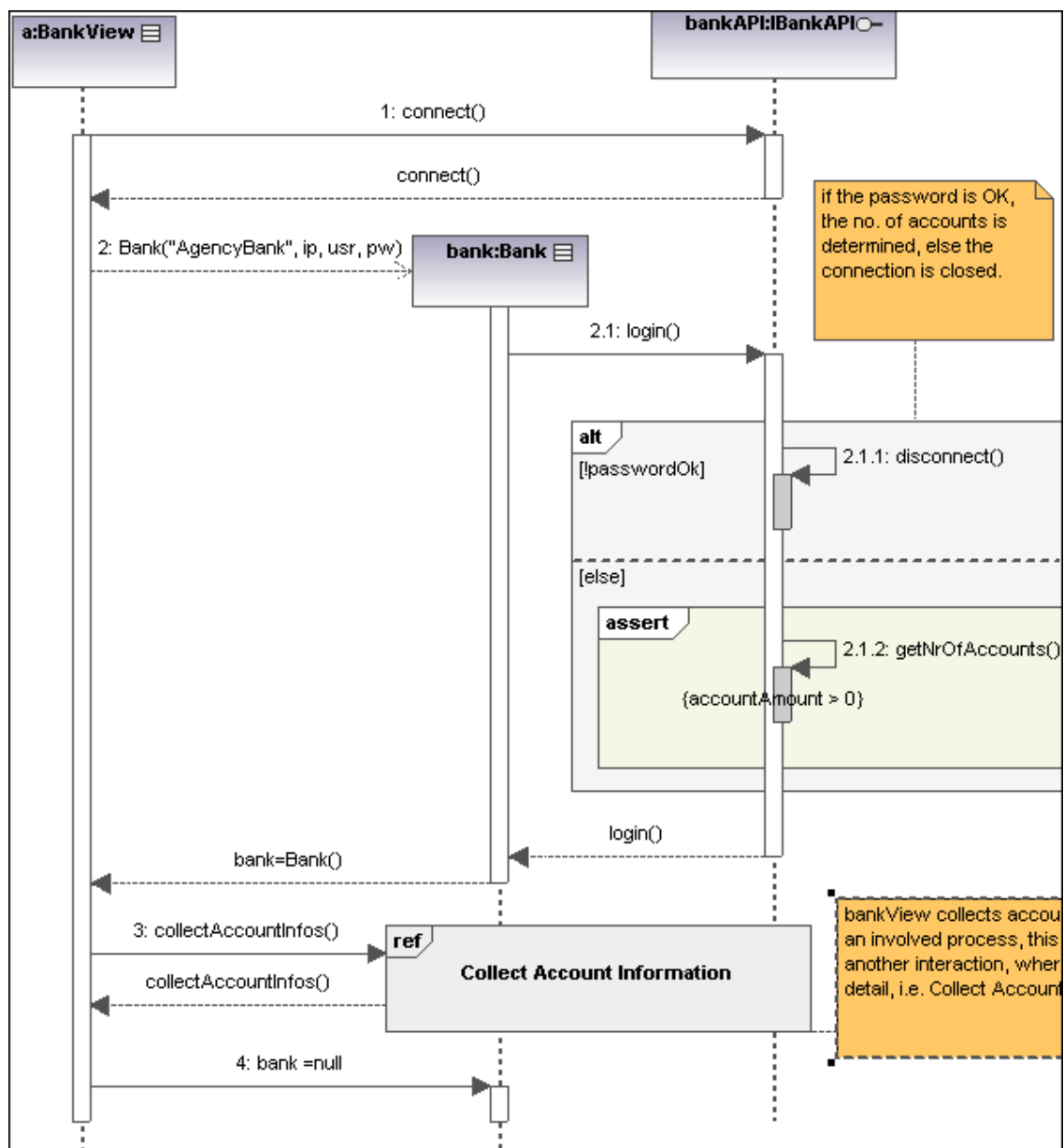
A Control Flow is an edge, i.e. an arrowed line, that connects two behaviours, and starts an interaction after the previous one has been completed.

10.1.7 Sequence Diagram

Altova web site:  [UML Sequence diagrams](https://www.altova.com/uml/sequence-diagrams)

UModel supports the standard Sequence diagram defined by UML, and allows easy manipulation of objects and messages to model use case scenarios. Please note that the sequence diagrams shown in the following sections are only available in the **Bank_Java.ump**, **Bank_CSharp.ump** and **Bank_MultiLanguage.ump** samples, in the ...**UModelExamples** folder supplied with UModel.

Note: It is now possible to generate a sequence diagram directly from source code, please see [Sequence Diagram diagram generation](#) , or through the UModel API, see [How to create Sequence Diagrams](#).



Inserting sequence diagram elements

A sequence diagram models runtime dynamic object interactions, using messages. Sequence diagrams are generally used to explain individual use case scenarios.

- **Lifelines** are the horizontally aligned boxes at the top of the diagram, together with a dashed vertical line representing the object's life during the interaction. Messages are shown as arrows between the lifelines of two or more objects.
- **Messages** are sent between sender and receiver objects, and are shown as labeled arrows. Messages can have a sequence number and various other optional attributes: argument list etc. Conditional, optional, and alternative messages are all supported. Please see [Combined Fragment](#) for more information.

Please see the following topics for more information:

[Lifeline](#)

[Combined Fragment](#)

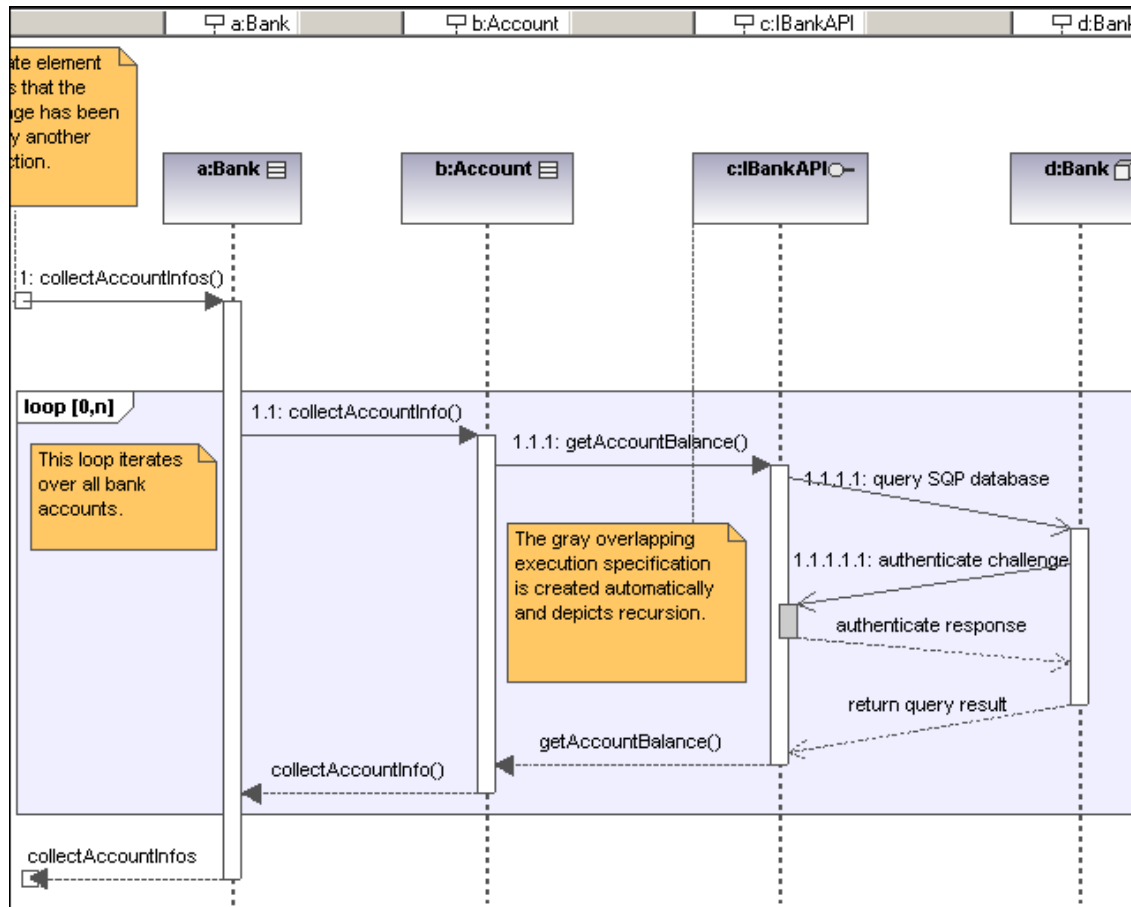
[Interaction Use](#)

[Gate](#)

[State Invariant](#)

[Messages](#)

Sequence diagram and other UModel elements, can be inserted into a sequence diagram using several methods.



Using the toolbar icons:

1. Click the specific sequence diagram icon in the Sequence Diagram toolbar.
2. Click in the Sequence diagram to insert the element. To insert multiple elements of the selected type, hold down the **Ctrl** key and click in the diagram window.

Dragging existing elements into the sequence diagram:

Most classifier types, as well as elements occurring in other sequence diagrams, can be inserted into an existing sequence diagram.

1. Locate the element you want to insert in the Model Tree tab (you can use the search function text box, or press **Ctrl+F** to search for any element).
2. Drag the element(s) into the sequence diagram.

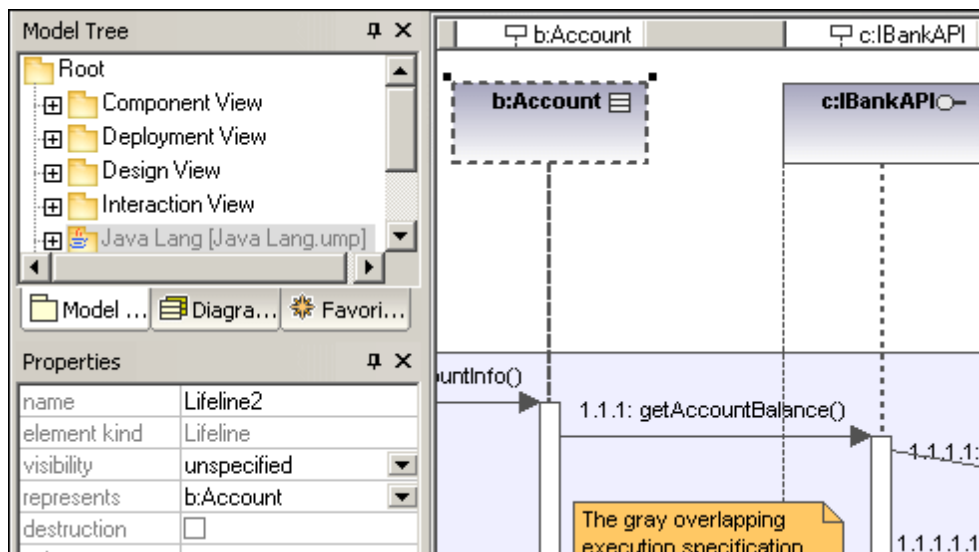
Lifeline**Lifeline**

The lifeline element is an individual participant in an interaction. UModel also allows you to insert other elements into the sequence diagram, e.g. classes and actors. Each of these elements appear as a new lifeline once they have been dragged into the diagram pane from the Model Tree tab.

The lifeline label appears in a bar at the top of the sequence diagram. Labels can be repositioned and resized in the bar, with changes taking immediate effect in the diagram tab. You can also redefine the label colors/gradient using the "Header Gradient" combo boxes in the Styles tab.

To create a **multiline** lifeline, press **Ctrl+Enter** to create a new line.

Most classifier types can be inserted into the sequence diagram. The "represents" field in the Properties tab displays the element type that is acting as the lifeline. Dragging **typed** properties onto a sequence diagram also creates a lifeline.

**Execution Specification (Object activation):**

An execution specification (activation) is displayed as a box (rectangle) on the object lifeline. An activation is the execution of a procedure and the time needed for any nested procedures to execute. Activation boxes are automatically created when a message is created between two lifelines.

A recursive, or self message (one that calls a different method in the same class) creates stacked activation boxes.

Displaying/hiding activation boxes:

1. Click the **Styles** tab and scroll to the bottom of the list.
The "**Show Execution Specifications**" combo box allows you to show/hide the activation boxes in the sequence diagram.

Lifeline attributes:

The **destruction** check box allows you to add a destruction marker, or stop, to the lifeline without having to use a destruction message.

The **selector** field allows you to enter an expression that specifies the particular part represented by the lifeline, if the ConnectableElement is multivalued, i.e. has a multiplicity greater than one.

Goto lifeline element

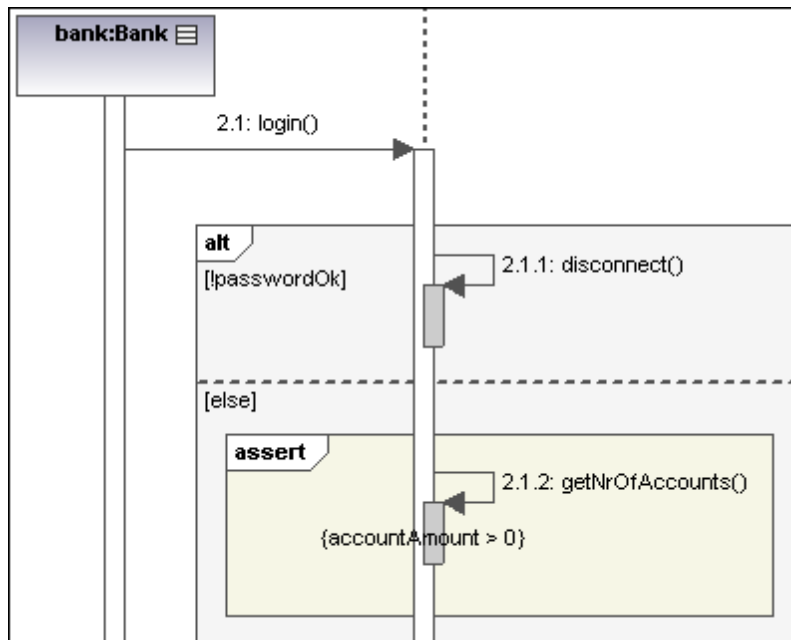
Right clicking a lifeline allows you to select Goto XXX, where XXX is the specific lifeline type that you clicked. The element will then be visible in the Model Tree window.

Combined Fragment

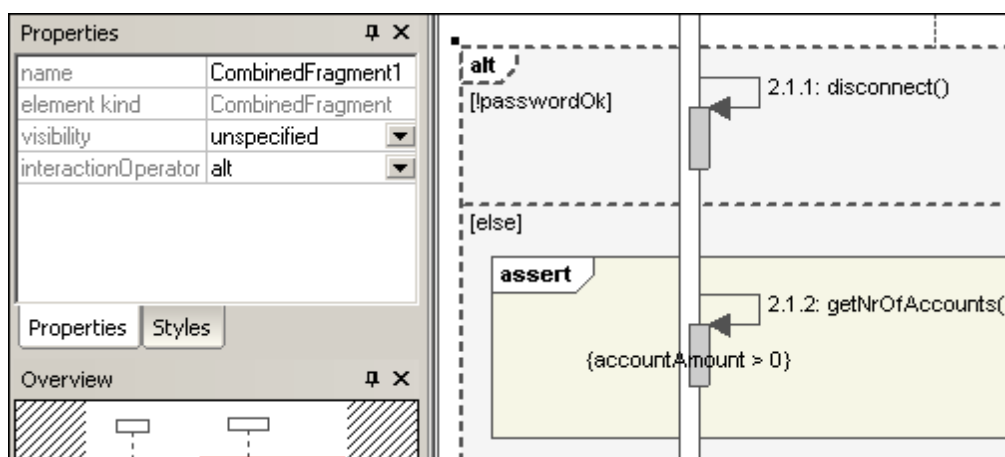


CombinedFragment

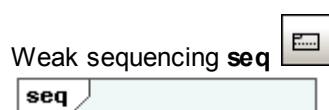
Combined fragments are subunits, or sections of an interaction. The **interaction operator** visible in the pentagon at top left, defines the specific kind of combined fragment. The constraint thus defines the specific fragment, e.g. loop fragment, alternative fragment etc. used in the interaction.



The combined fragment icons in the icon bar, allow you to insert a specific combined fragment: seq, alt or loop. Clicking the **interactionOperator** combo box, also allows you to define the specific interaction fragment.



InteractionOperators



The combined fragment represents weak sequencing between the behaviours of the operands.

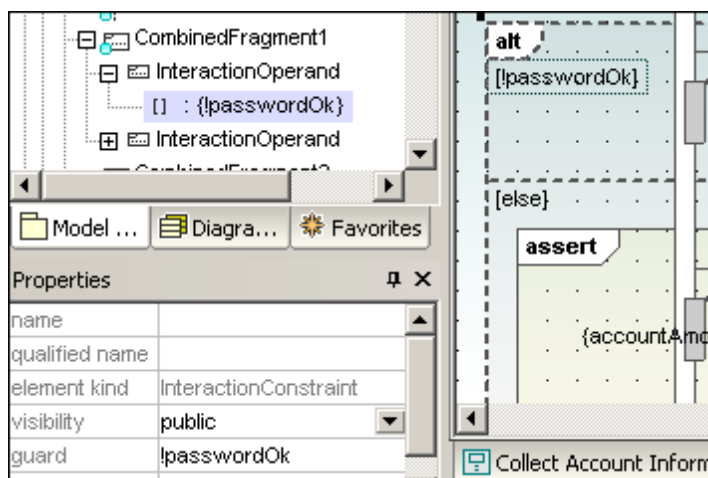


Alternatives alt

Only one of the defined operands will be chosen, the operand must have a guard expression that evaluates to true.



If one of the operands uses the guard "else", then this operand is executed if all other guards return false. The guard expression can be entered immediately upon insertion, will appear between the two square brackets.



The **InteractionConstraint** is actually the guard expression between the square brackets.

Option opt

Option represents a choice where either the sole operand is executed, or nothing happens.

Break `break`

The break operator is chosen when the guard is true, the rest of the enclosing fragment is ignored.

Parallel `par`

Indicates that the combined fragment represents a parallel merge of operands.

Strict sequencing `strict`

The combined fragment represents a strict sequencing between the behaviours of the operands.

Loop `loop`

The loop operand will be repeated by the number of times defined in the guard expression.



Having selected this operand, you can directly edit the expression (in the loop pentagon) by double clicking.

Critical Region `critical`

The combined fragment represents a critical region. The sequence(s) may not be interrupted/interleaved by any other processes.

Negative `neg`

Defines that the fragment is invalid, and all others are considered to be valid.

Assert `assert`

Designates the valid combined fragment, and its sequences. Often used in combination with consider, or ignore operands.

Ignore `ignore`

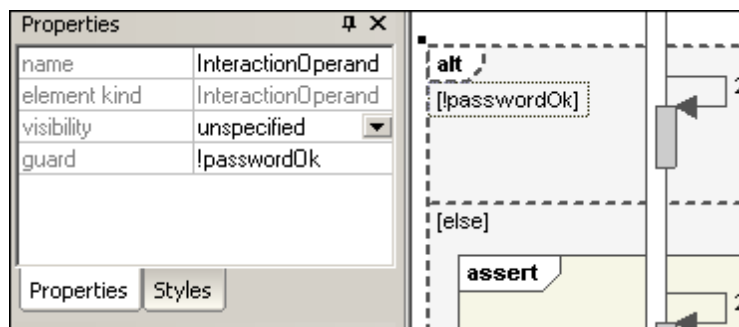
Defines which messages should be ignored in the interaction. Often used in combination with assert, or consider operands.

Consider `consider`

Defines which messages should be considered in the interaction.

Adding InteractionOperands to a combined fragment:

1. Right click the combined fragment and select **New | InteractionOperand**.
The text cursor is automatically set for you to enter the guard condition.
2. Enter the guard condition for the InteractionOperand e.g. **!passwordOK** and press Enter to confirm.



Use **Ctrl+Enter** to create a **multi-line** InteractionOperand.

3. Use the same method to add the second interaction operand with the guard condition "else".
Dashed lines separate the individual operands in the fragment.

Deleting InteractionOperands:

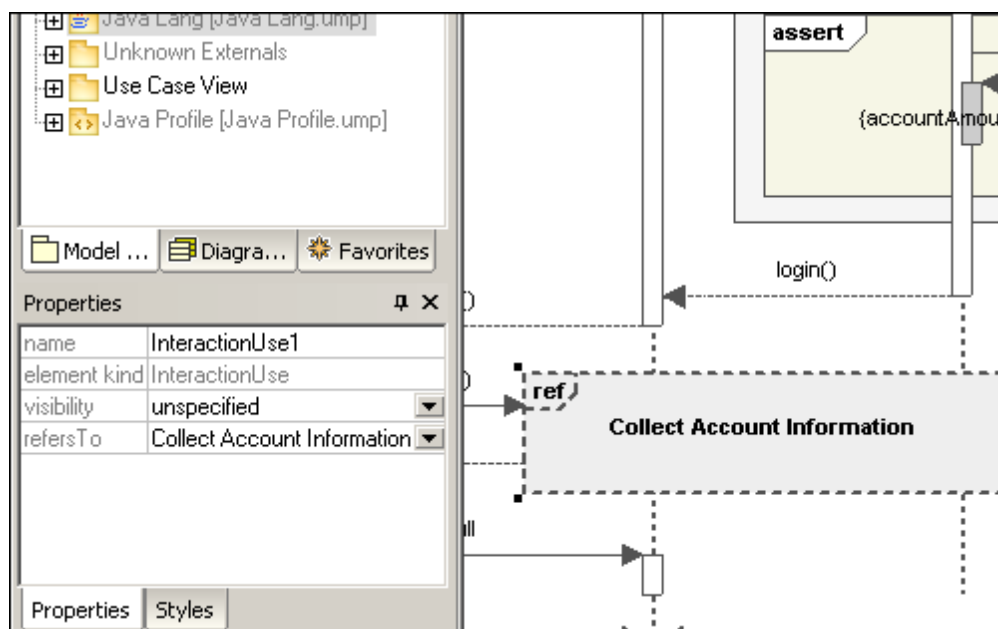
1. Double click the guard expression in the combined fragment element, of the diagram (not in the Properties tab).
2. Delete the guard expression completely, and press Enter to confirm.
The guard expression/interaction operand is removed and the combined fragment is automatically resized.

Interaction Use



InteractionUse

The InteractionUse element is a reference to an interaction element. This element allows you to share portions of an interaction between several other interactions.



Clicking the "refersTo" combo box, allows you to select the interaction that you want to refer to. The name of the interaction use you select, appears in the element.

Please note:

You can also drag an existing Interaction Use element from the Model Tree into the diagram tab.

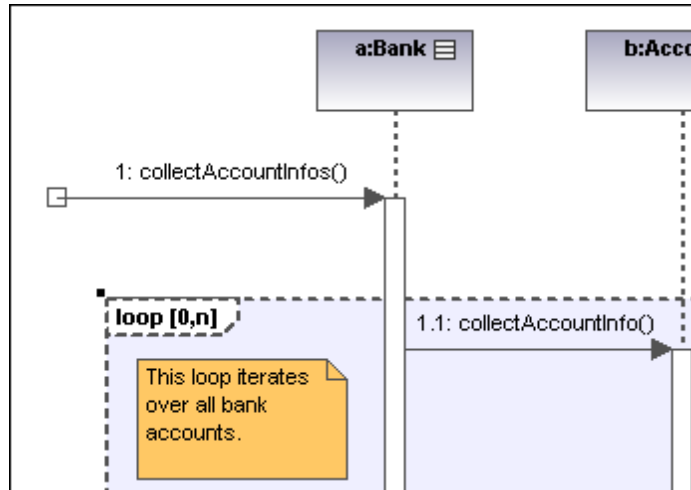
Gate



Gate

A gate is a connection point which allows messages to be transmitted into, and out of, interaction fragments. Gates are connected using messages.

1. Insert the gate element into the diagram.
 2. Create a new message and drag from the gate to a lifeline, or drag from a lifeline and drop onto a gate.
- This connects the two elements. The square representing the gate is now smaller.



State Invariant

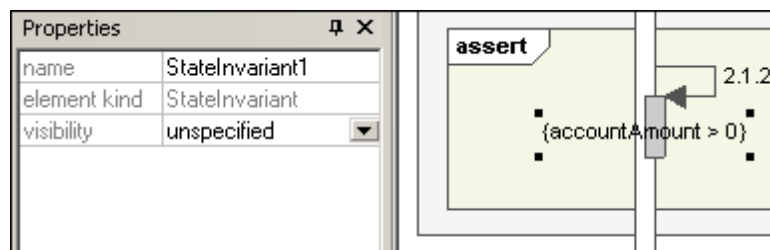


StateInvariant

A StateInvariant is a condition, or constraint applied to a lifeline. The condition must be fulfilled for the lifeline to exist.

To define a StateInvariant:

1. Click the State invariant icon, then click a lifeline, or an object activation to insert it.
2. Enter the condition/constraint you want to apply, e.g. `accountAmount > 0`, and press Enter to confirm.



Messages

Messages are sent between sender and receiver lifelines, and are shown as labeled arrows. Messages can have a sequence number and various other optional attributes: argument list etc. Messages are displayed from top to bottom, i.e. the vertical axis is the time component of the sequence diagram.

- A **call** is a synchronous, or asynchronous communication which invokes an operation that allows control to return to the sender object. A call arrow points to the **top** of the activation that the call initiates.
- Recursion, or calls to another operation of the same object, are shown by the stacking of activation boxes (Execution Specifications).

To insert a message:

1. Click the specific message icon in the Sequence Diagram toolbar.
 2. Click the lifeline, or activation box of the sender object.
 3. Drag and drop the message line onto the receiver objects lifeline or activation box.
Object lifelines are highlighted when the message can be dropped.
- The direction in which you drag the arrow defines the message direction. Reply messages can point in either direction.
 - Activation box(es) are automatically created, or adjusted in size, on the sender/receiver objects.
 - Depending on the message numbering settings you have enabled, the numbering sequence is updated.
 - Having clicked a message icon and holding down **Ctrl** key, allows you to insert multiple messages by repeatedly clicking and dragging in the diagram tab.

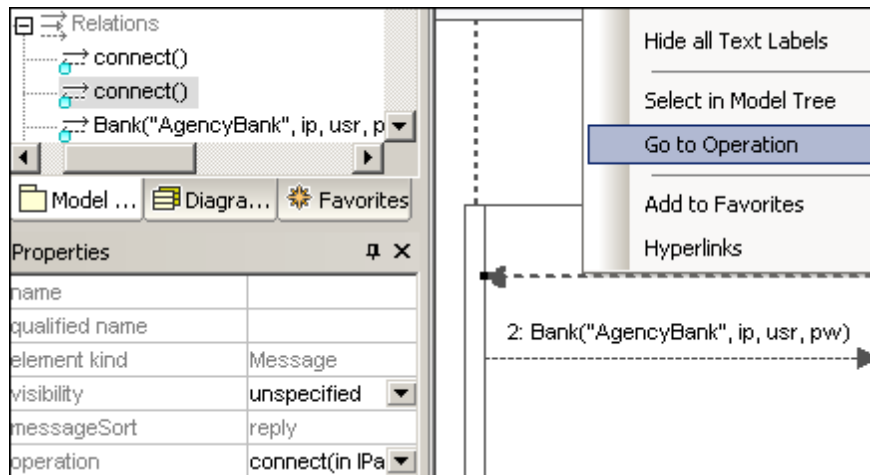
To delete a message:

1. Click the specific message to select it.
2. Press the Del. key to delete it from the model, or right click it and select "Delete from diagram".
The message numbering and activation boxes of the remaining objects are updated.

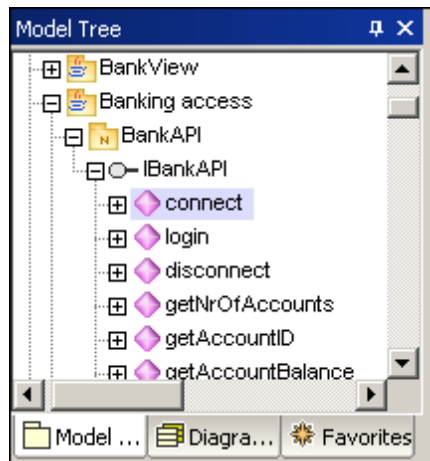
"Go to operation" for call messages:

The operations referenced by call messages can be found in sequence and communication diagrams.

1. Right click a call message and select "Go to Operation".

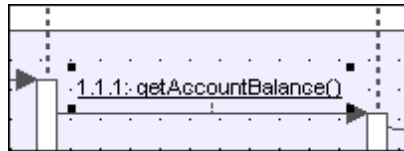


The display changes and the connect operation is displayed in the Model Tree tab.



Please note:

Static operation names are shown as underlined in sequence diagrams.



To position dependent messages:

1. Click the respective message and drag vertically to reposition it.
The default action when repositioning messages, is it to move all dependent messages related to the active one.

Using **Ctrl+Click**, allows you to select multiple messages.

To position messages individually:

1. Click the "**Toggle dependent message movement**" icon  to deselect it.
2. Click the message you want to move and drag to move it.
Only the selected message moves during dragging. You can position the message anywhere in the vertical axis between the object lifelines.

To automatically create reply messages:

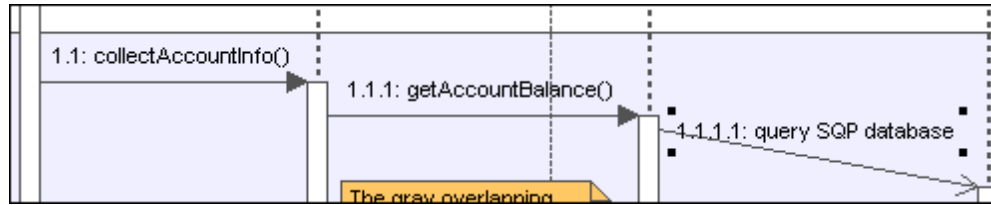
1. Click the "**Toggle automatic creation of replies for messages**" icon .
2. Create a new message between two lifelines.
A reply message is automatically inserted for you.

Message numbering:

UModel supports different methods of message numbering: nested, simple and none.

- **None**  removes all message numbering.
- **Simple**  assigns a numerical sequence to all messages from top to bottom i.e. in the order that they occur on the time axis.
- **Nested**  uses the decimal notation, which makes it easy to see the hierarchical structure of the messages in the diagram. The sequence is a dot-separated list of

sequence numbers followed by a colon and the message name.



To select the message numbering scheme:

There are two methods of selecting the numbering scheme:

- Click the respective icon in the icon bar.
- Use the Styles tab to select the scheme.

To select the numbering scheme using the Styles tab:

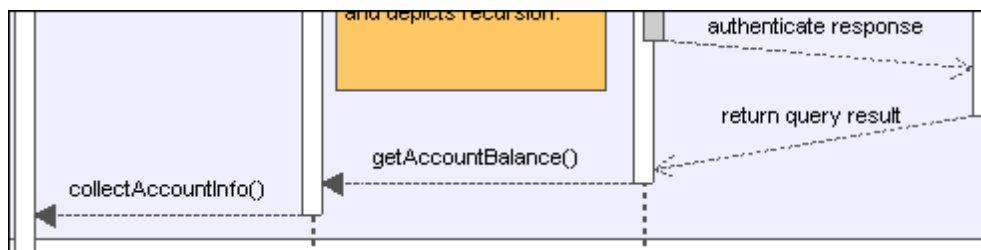
1. Click the Styles tab and scroll down to the **Message Numbering** field.
2. Click the combo box and select the numbering option you want to use.
The numbering option you select is immediately displayed in the sequence diagram.

Please note:

The numbering scheme might not always correctly number all messages, if ambiguous traces exist. If this happens, adding return messages will probably clear up any inconsistencies.

Message replies:

Message reply icons are available to create reply messages, and are displayed as dashed arrows.

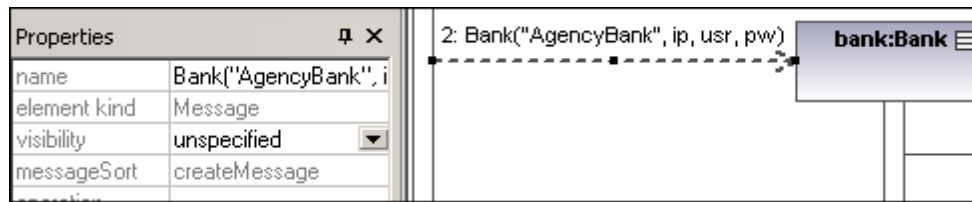


Reply messages are also generally implied by the bottom of the activation box when activation boxes are present. If activation boxes have been disabled (**Styles tab | Show Execution Specifics=false**), then reply arrows should be used for clarity.

Activating the  "toggle reply messages" icon, automatically creates syntactically correct reply messages when creating a call message between lifelines/activation boxes.

Creating objects with messages:

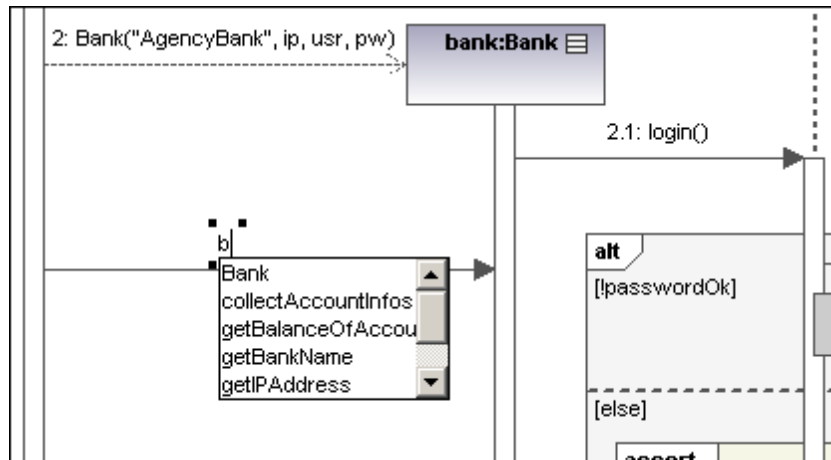
1. Messages can create new objects. This is achieved using the Message Creation icon .
2. Drag the message arrow to the lifeline of an existing object to create that object. This type of message ends in the middle of an object rectangle, and often repositions the object box vertically.



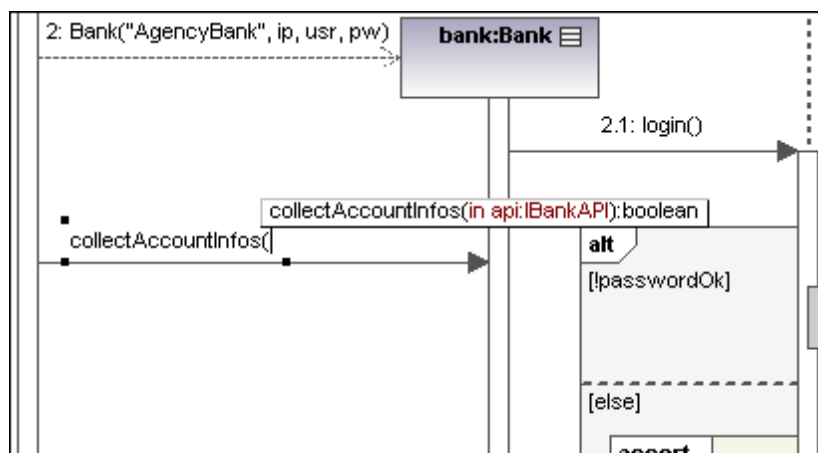
Sending messages to specific class methods/operations in sequence diagrams

Having inserted a class from the Model Tree into a sequence diagram, you can then create a message from a lifeline to a specific method of the receiver class (lifeline) using UModel's syntax help and autocompletion functions.

1. Create a message between two lifelines, the receiving object being a class lifeline (Bank)
As soon as you drop the message arrow, the message name is automatically highlighted.
2. Enter a character using the keyboard e.g. "b".
A pop-up window containing a list of the existing class methods is opened.



3. Select an operation from the list, and press Enter to confirm e.g. collectAccountInfos.
 4. Press the space bar and press Enter to select the parenthesis character that is automatically supplied.
- A syntax helper now appears, allowing you to enter the parameter correctly.



Creating operations in referenced classes

Activating the  "Toggle automatic creation of operations in target by typing operation names" icon, automatically creates the corresponding operation in the referenced class, when creating a message and entering a name e.g. myOperation().

Note: operations can only be created automatically when the lifeline references a class, interface...

Message icons:



Message (Call)



Message (Reply)



Message (Creation)



Message (Destruction)



Asynchronous Message (Call)



Asynchronous Message (Reply)



Asynchronous Message (Destruction)



Toggle dependent message movement



Toggle automatic creation of replies for messages



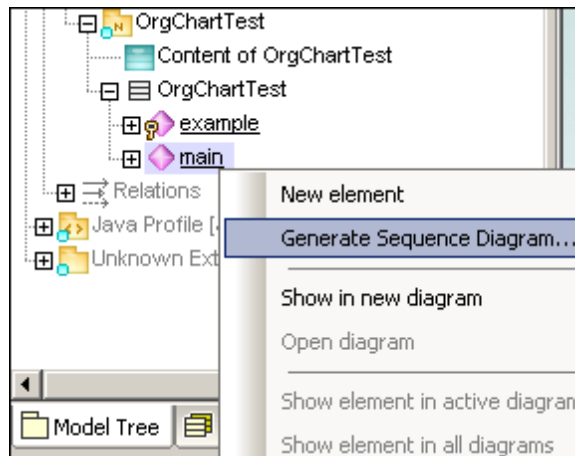
Toggle automatic creation of operations in target by typing operation names

Generating Sequence Diagrams from source code

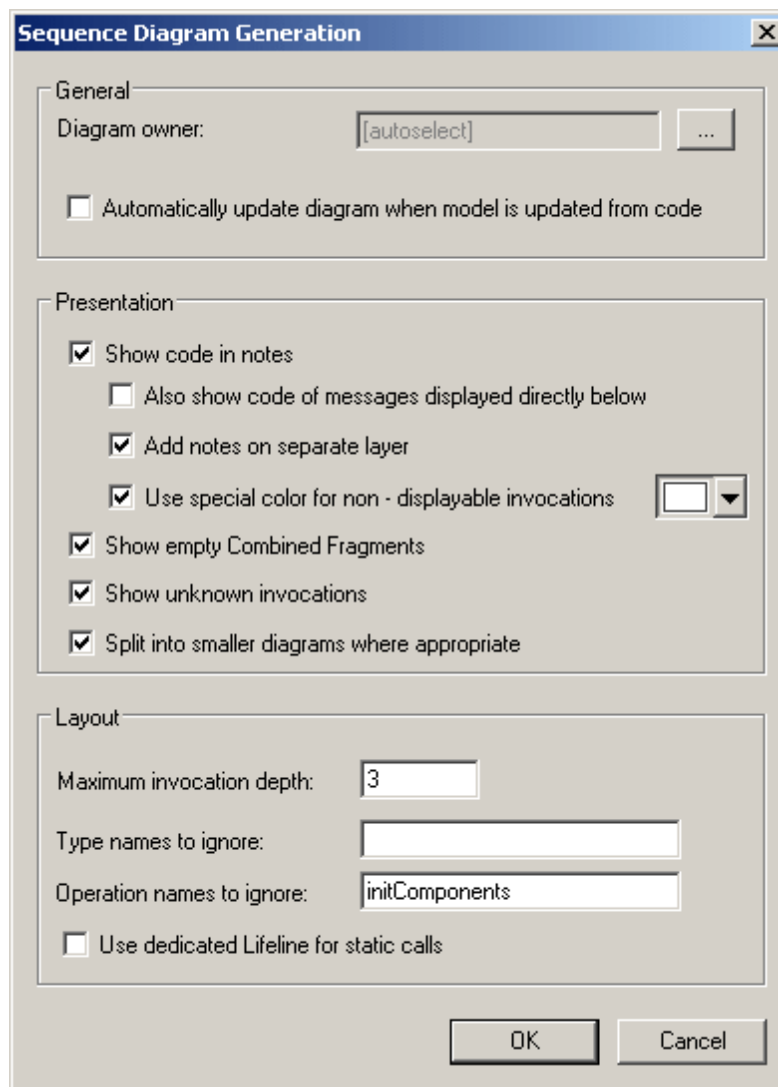
UModel is able to create sequence diagram models directly from code, or through the UModel API. Please see [How to create Sequence Diagrams](#) using the UModelAPI for more information.

The example below shows how to automatically create a sequence diagram by using an existing method. You can, of course, use the same approach on your own code. The method is available in the OrgChart package which was imported using the **Project | Import Source Directory** command in the tutorial. Please see [Round-trip engineering \(code - model - code\)](#) in the tutorial for more information on importing code.

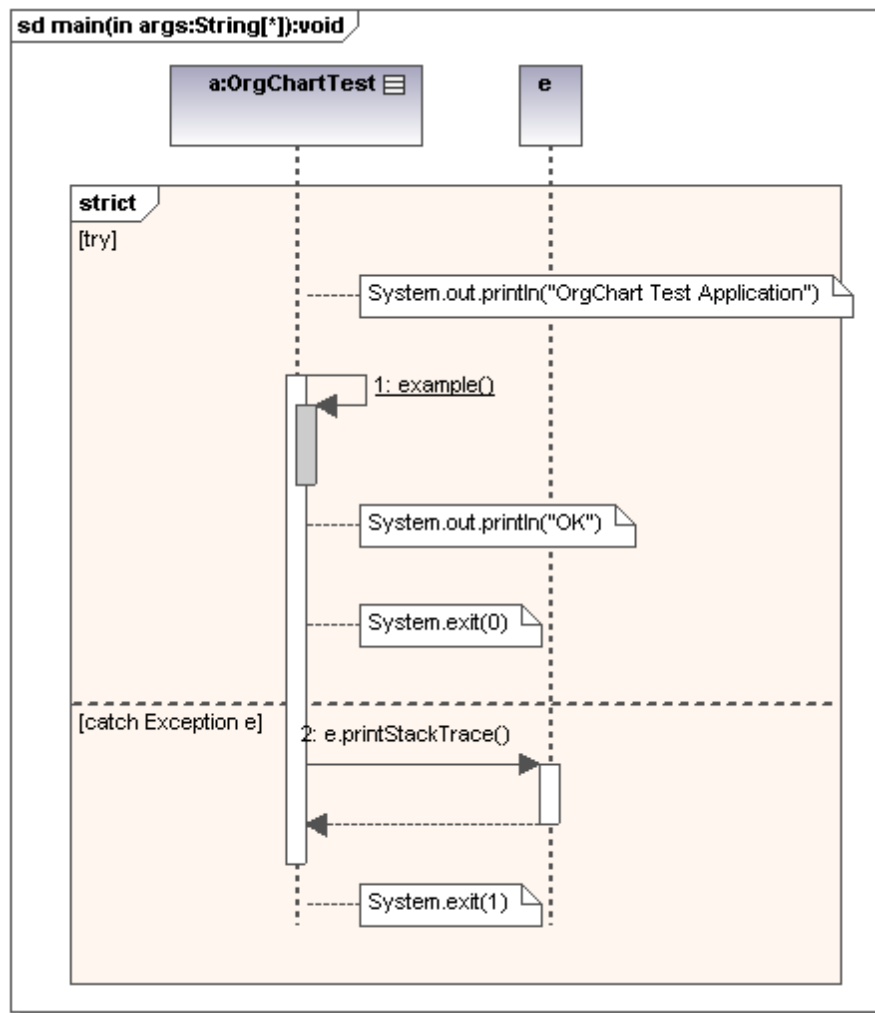
1. Having imported the code, right click the **main** method of the **OrgChartTest** class, in the Model Tree and select **Generate Sequence Diagram...**



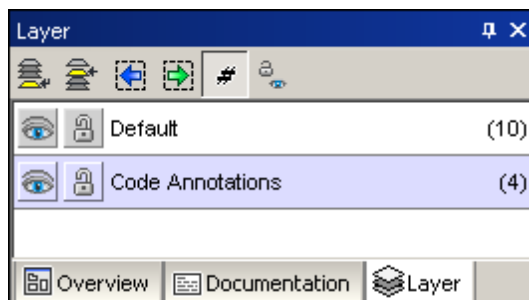
This opens the Sequence Diagram Generation dialog box in which you define the generation settings.



2. Select the presentation and layout options, then click OK to generate the diagram. The settings shown above produce the sequence diagram below.



The "Add notes on separate layer" option assigns the code notes to the Code Annotations layer. The code notes are shown in orange.



Notes:

- You can assign a special color to the non-displayable invocations.
- The maximum **invocation depth** defines the **recursion** depth to be used in the diagram.
- The "**Type names to ignore**" option, allows you to define a comma delimited list of types that should not appear in the sequence diagram when it is generated.
- The "**Operation names to ignore**" option allows you define a comma delimited list of operations that should not appear in the generated sequence diagram (initComponents

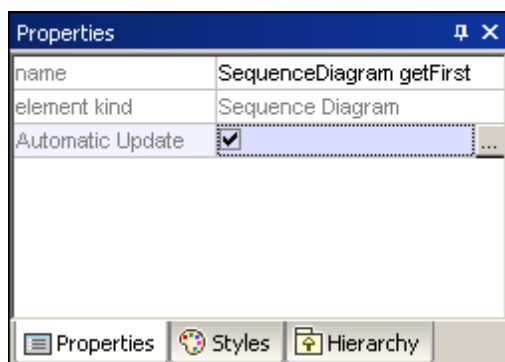
are automatically added as a default).

Adding the operation names to the list e.g. "InitComponents" causes the complete operation to be ignored. Prepending a + character to the operation in the list e.g.

+InitComponent, causes the operation calls to be shown in the diagram, but without their content.

- The "**Split into smaller diagrams where appropriate**" option automatically splits sequence diagrams into smaller sub-diagrams, and automatically generates hyperlinks between them for easy navigation.

Sequence diagrams are automatically updated when you update the complete UModel project. Sequence diagrams are not updated if you update classes, or individual files, externally. You can however change this setting by clicking on the diagram background and clicking the **Automatic Update** check box.



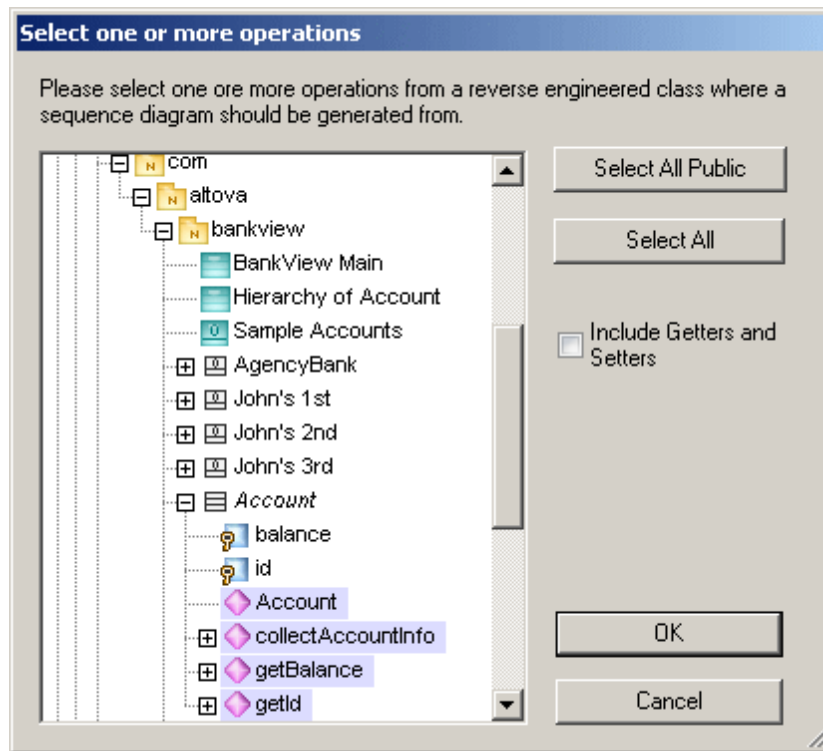
Clicking the  icon of the Automatic Update field, opens the Sequence Diagram Generation dialog box allowing you to change the sequence diagram generation settings.

Generate Sequence diagrams from properties

UModel is able to create multiple sequence diagram models from multiple operations, as well as sequence diagrams from Getter / Setter properties.

Creating multiple sequence diagrams from multiple operations:

1. Select the menu option **Project | Generate Sequence diagrams**.
This opens the **Select one or more operations** dialog box.

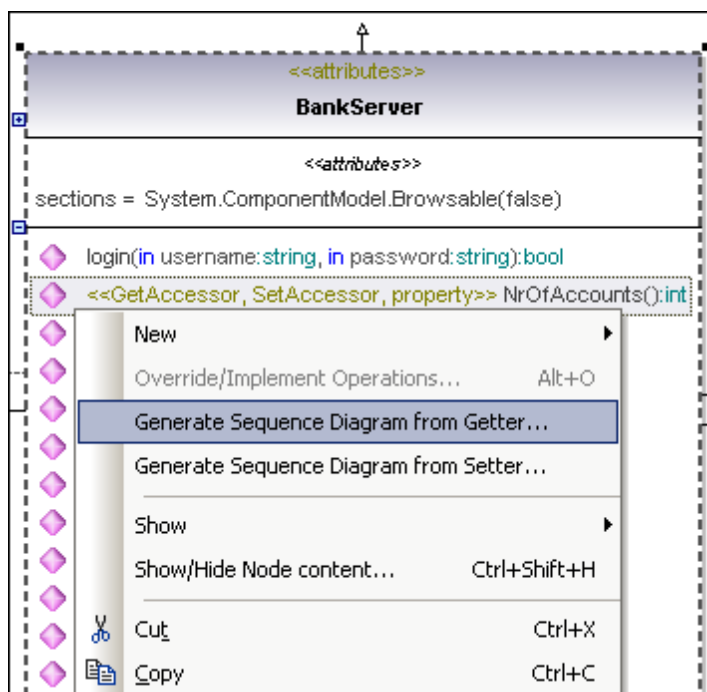


2. Select the operations that you want to generate a sequence diagram for and click OK. (Use the Select All Public and Select All buttons where necessary.) Clicking OK opens the Sequence Diagram Generation dialog box, in which you can further specify the sequence diagram generation options.
3. Click OK to generate the sequence diagrams. A sequence diagram is generated for each selected operation.

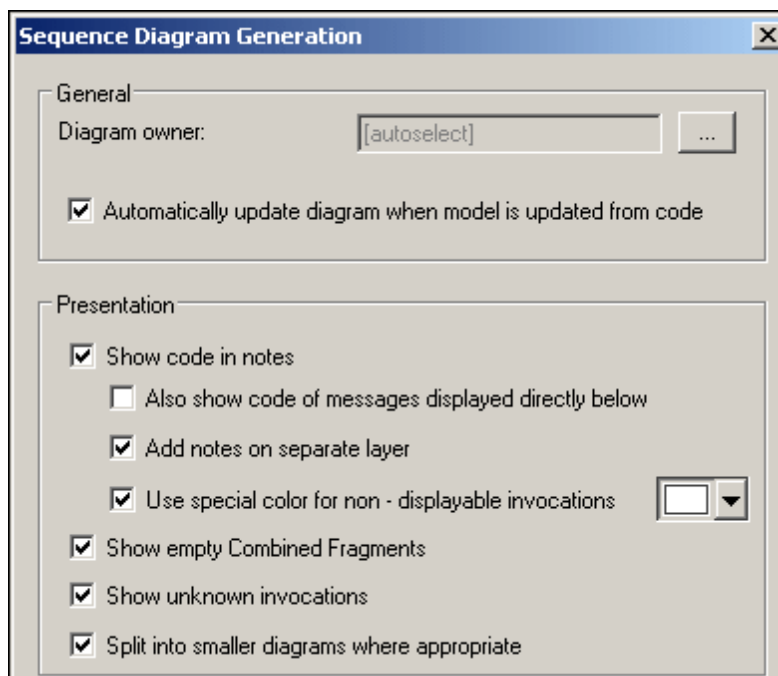
Note: you can now choose to include/exclude Getters and Setters, when generating the sequence diagram.

Creating a sequence diagram from getter/setter properties (C#, VB .NET)

1. Right click an Operation with a GetAccessor/SetAccessor stereotype.



2. Select the option from the context menu that you want to create a sequence diagram from (Getter/Setter). This opens the Sequence Diagram Generation dialog box in which you can further specify the sequence diagram presentation settings.



3. Click OK to generate the Sequence Diagram.

Generate code from sequence diagram

UModel can create code from a sequence diagram which is linked to at least one operation.

Code generation from sequence diagrams is available for:

- VB.NET, C# and Java
- UModel standalone, Eclipse, and Visual Studio editions
- All three UModel editions

Creating code from Sequence diagrams is possible by either:

- Starting from a reverse engineered operation, see [Generating Sequence Diagrams from source code](#),
- By creating a **new** sequence diagram from scratch, which is linked to an operation, by right-clicking the operation (in the Model Tree) and selecting [Create sequence diagram for code](#).

Notes:

When using a reverse engineered sequence diagram as basis, ensure that the option "Show code in notes" is selected when reverse engineering the code, so you do not lose any code when you start the forward-engineering process again.

This is due to the fact that UML is not able to display all the language features of VB.NET, Java and C# on the sequence diagram, and those code sections are therefore shown as code notes.

To add plain text as code when creating a sequence diagram:

1. Attach a note to a sequence diagram lifeline.
2. Type in the code which should be written into the final source code.
Click the "Is Code" check box (in the Properties pane) for that note, to make it accessible.

Please see [Adding code to sequence diagrams](#) for an example.

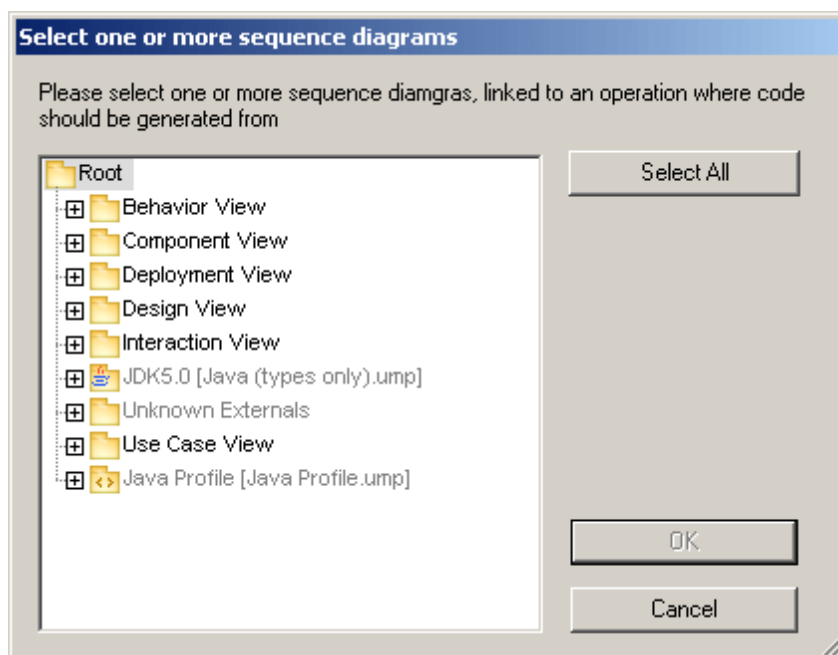
If a Sequence Diagram is to be used for code engineering automatically every time code engineering is started:

- Activate the "Use for forward engineering" check box in the property pane when the sequence diagram is selected.

Old code will always be lost when forward engineering code from a sequence diagram, because it will be overwritten with the new code.

Project menu:

1. Select the menu option **Project | Generate Code from Sequence Diagrams**.
You are now prompted to select the specific Sequence Diagram(s).

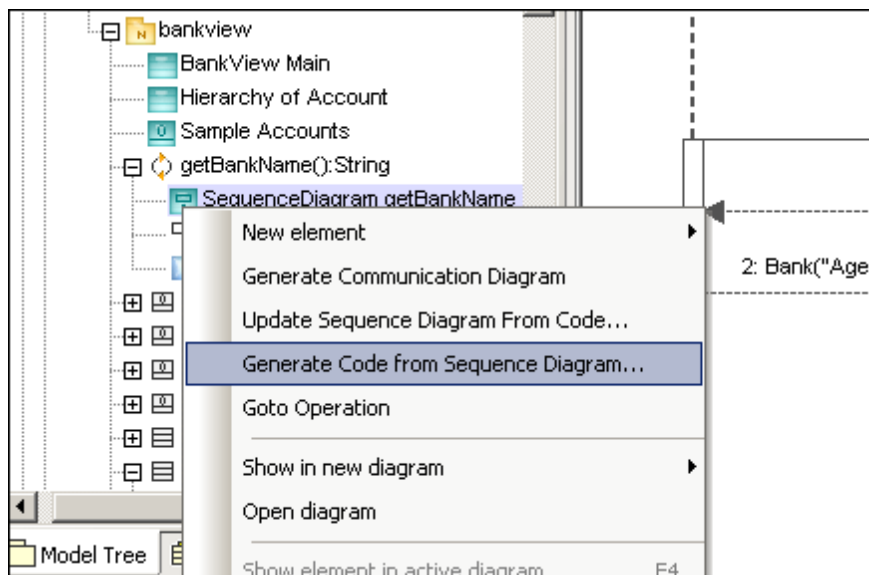


Clicking the "Select All" button selects all the Sequence Diagrams in the UModel project.

2. Click OK to generate the code.
The Messages window shows the status of the code generation process.

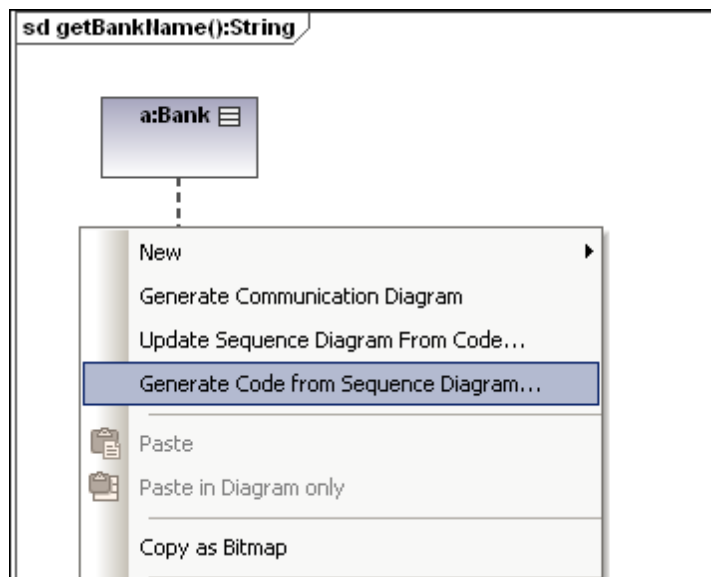
Model Tree:

- Right click a Sequence Diagram and select Generate Code from Sequence diagram.



Sequence Diagram containing code of an operation:

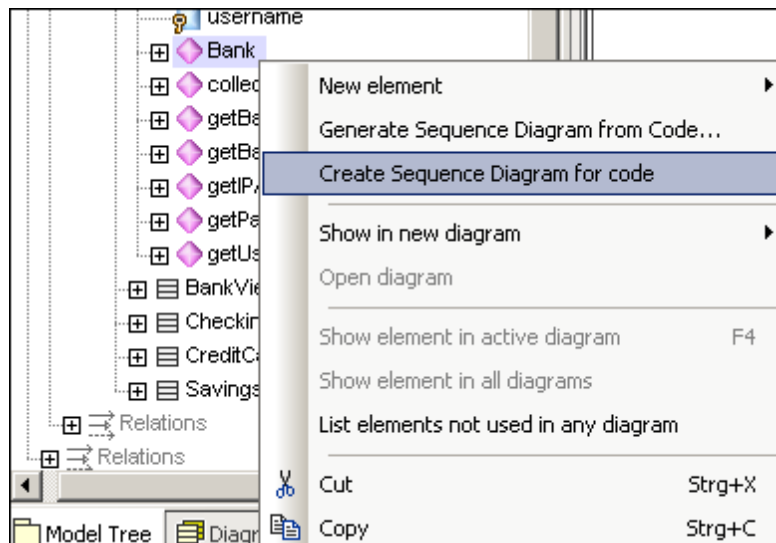
1. Click into the empty space of the Sequence Diagram, that contains code of an operation.
2. Select Generate Code from Sequence diagram.



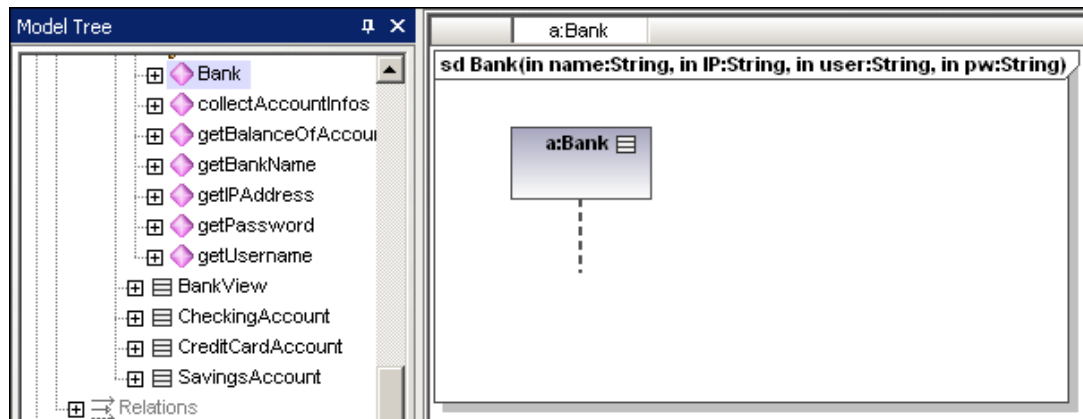
This command starts the forward-engineering process at this point.

To create a Sequence diagram for code (engineering):

1. In the Model Tree, right click an operation and select Create Sequence diagram for code.



You will then be prompted if you want to use the new diagram for forward engineering.



The result is a new Sequence Diagram containing the lifeline of that class.

Adding code to sequence diagrams

Program code can be generated from new, and reverse-engineered sequence diagrams, but only for a sequence diagram linked to the "main operation".

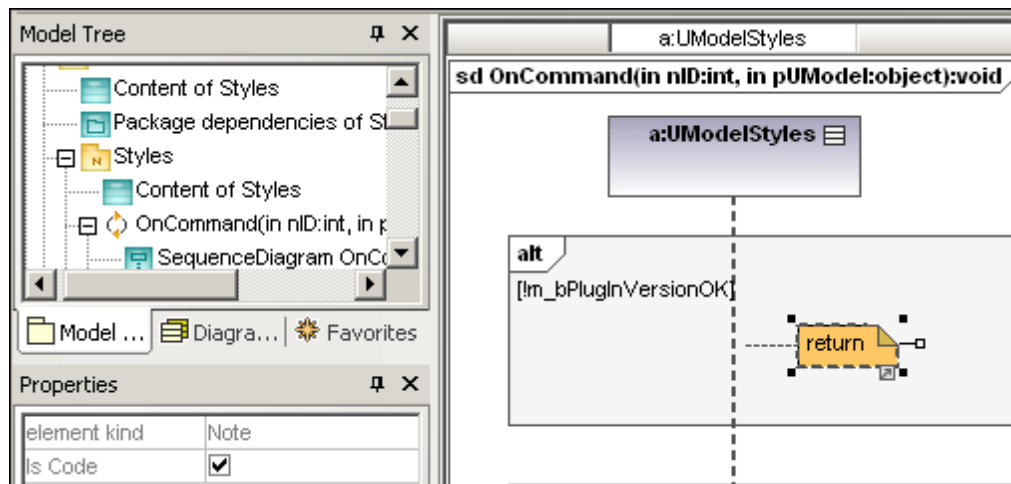
When reverse-engineering code, standard sequence diagram elements, e.g. CombinedFragments, are "mapped/assigned" to coding elements (e.g. "if" statements, loops, etc.).

For those programming statements that have no corresponding sequence diagram elements, e.g. "i = i+1", UModel makes use of "code" notes to add code to diagrams. These notes must then be linked to the lifeline.

Note that UModel does not check, or parse, these code fragments. It is up to you to make sure that the code fragments are correct and will compile.

To add code to a sequence diagram:

1. Click the Note icon  then click the model element where you want to insert it, e.g. CombinedFragment.
2. Enter the code fragment, e.g. return.
3. Click the Node Link handle of the inserted note and drop the cursor on the lifeline.
4. Activate the "Is Code" check box in the Properties tab to include this code fragment when generating code.



When selecting a note on a sequence diagram, which **can** be used for code generation, the property "is code" is available in the Properties window. Clicking the check box, allows you to switch between "ordinary" notes and code generation notes.

Ordinary notes:



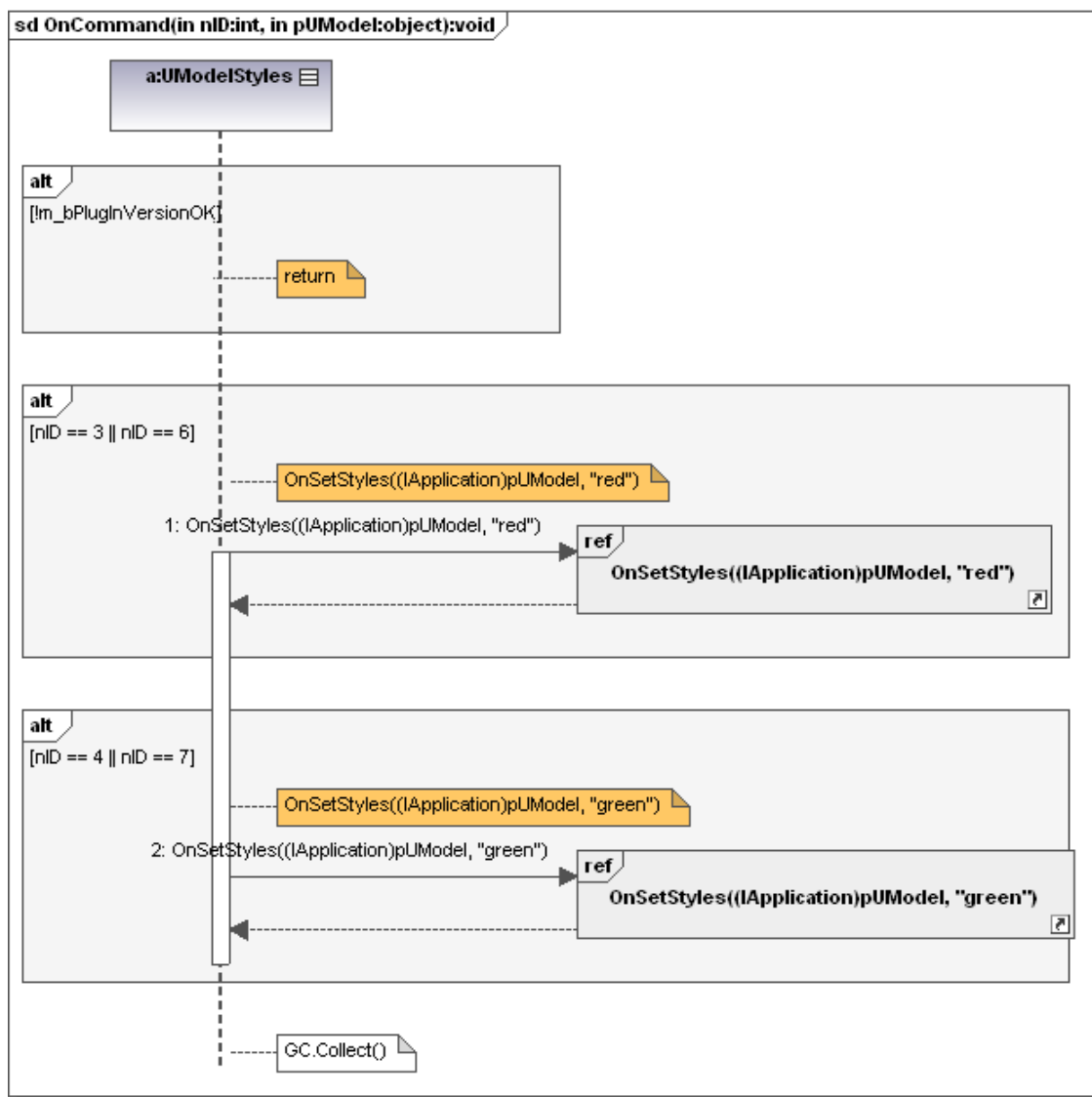
Code generation notes



- shown with a darker dog-ear

Code updates occur automatically on every forward engineering process if the "Use for forward engineering" check box is active. If changes were made to the sequence diagram, the code of the operation is always overwritten.

The sequence diagram shown below was generated by right clicking the **OnCommand** operation and selecting "Generate sequence diagram from code". The C# code of this example is available in the c:\Documents and Settings\<user>\My Documents\Altova\UModel2012\UModelExamples\IDEPlugin\Styles\ folder. Use the option **Project | Import Source Project**, to import the project.



The code shown below is generated from the sequence diagram.

```

Public void OnCommand(int nID, object pUModel)
{
    //Generated by UModel. This code will be overwritten when you re-run code
    generation.

    if (!m_bPlugINVersionOK)
    {
        return;
    }

    if (nID == 3 || nID == 6)
    {
        OnSetStyles((IApplication)pUModel, "red");
    }

    if (nID == 4 || nID == 7)
  
```

```
{  
    OnSetStyles((IApplication)pUModel, "green");  
}  
GC.Collect();  
}
```

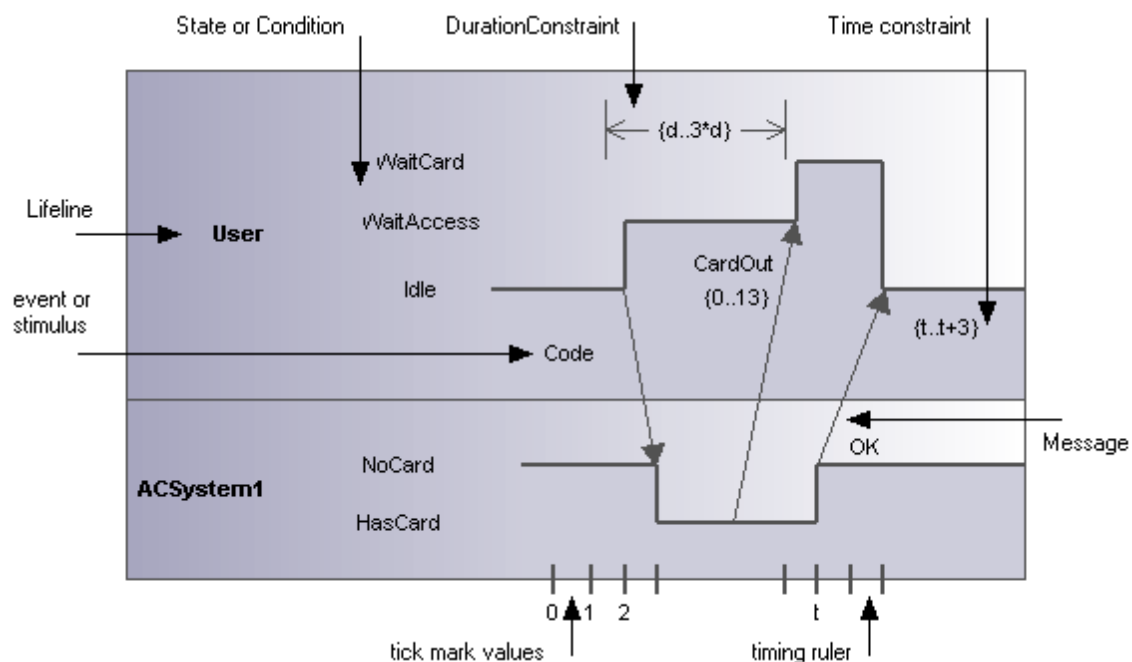
10.1.8 Timing Diagram

Altova web site:  [UML Timing diagrams](http://www.altova.com/UML/TimingDiagrams/)

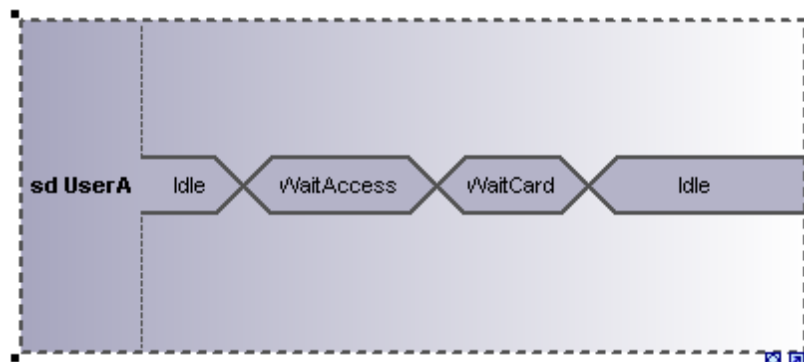
Timing diagrams depict the changes in state, or condition, of one or more interacting objects over a given period of time. States, or conditions, are displayed as timelines responding to message events, where a lifeline represents a Classifier Instance or Classifier Role.

A Timing diagram is a special form of a sequence diagram. The difference is that the axes are reversed i.e. time increases from left to right, and lifelines are shown in separate vertically stacked compartments.

Timing diagrams are generally used when designing embedded software or real-time systems.



There are two different types of timing diagram: one containing the State/Condition timeline as shown above, and the other, the General value lifeline, shown below.



Inserting Timing Diagram elements

Using the toolbar icons:

1. Click the specific timing icon in the Timing Diagram toolbar.



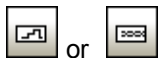
2. Click in the Timing Diagram to insert the element. To insert multiple elements of the selected type, hold down the **Ctrl** key and click in the diagram window.

Dragging existing elements into the timing machine diagram:

Elements occurring in other diagrams, e.g. classes, can be inserted into an Timing Diagram.

1. Locate the element you want to insert in the Model Tree tab (you can use the search function text box, or press **Ctrl+F** to search for any element).
2. Drag the element(s) into the state diagram.

Lifeline

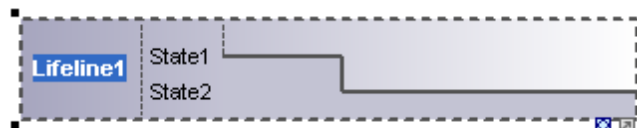


Lifeline

The lifeline element is an individual participant in an interaction, and is available in two different representations: State/Condition timeline or General Value lifeline. To create a **multiline** lifeline press **Ctrl+Enter** to create a new line.

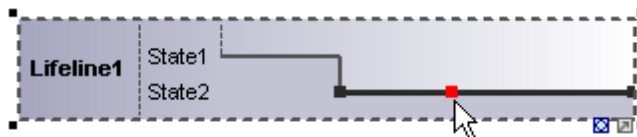
To insert a State Condition (StateInvariant) lifeline and define state changes:

1. Click the Lifeline (State/Condition) icon  in the title bar, then click in the Timing Diagram to insert it.



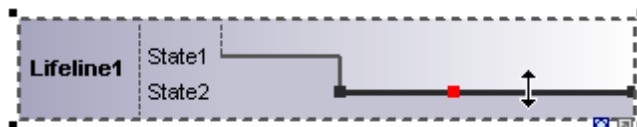
2. Enter the lifeline name to change it from the default name, Lifeline1, if necessary.
3. Place the mouse cursor over a section of one of the timelines and click left. This selects the line.
4. Move the mouse pointer to the position you want a state change to occur, and click again.

Note that you will actually see the double headed arrow when you do this.

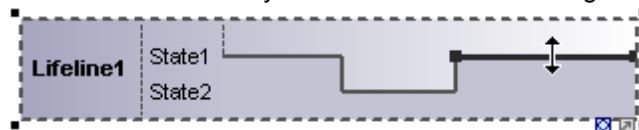


A red box appears at the click position and divides the line at this point.

5. Move the cursor to the right hand side of the line and drag the line upwards.



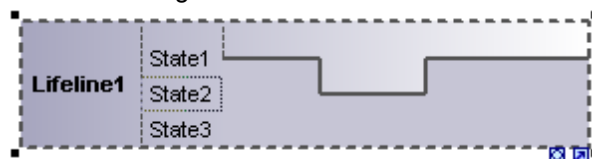
Note that lines can only be moved between existing states of the current lifeline.



Any number of state changes can be defined per lifeline. Once the red box appears on a line, clicking anywhere else in the diagram deletes it.

To add a new state to the lifeline:

1. Right click the lifeline and select **New | State/Condition (StateInvariant)**.
A new State e.g. State3 is added to the lifeline.



To move a state within a lifeline:

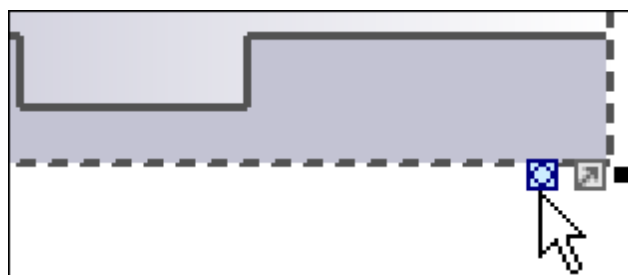
1. Click the state label that you want to move.
2. Drag it to a different position in the lifeline.

To delete a state from a lifeline:

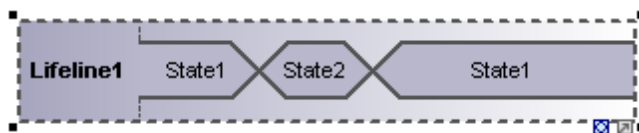
1. Click the state and press the Del. key, or alternatively, right click and select Delete.

To switch between timing diagram types:

1. Click the "toggle notation" icon at the bottom right of the lifeline.



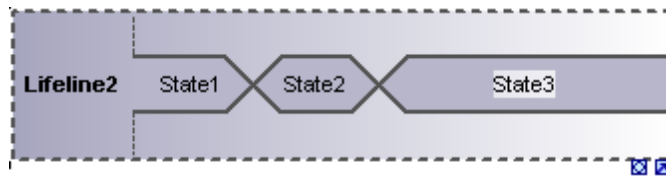
This changes the display to the General Value lifeline, the cross-over point represents a state/value change.



Please note that clicking the Lifeline (General Value) icon , inserts the lifeline as shown above. You can switch between the two representations at any time.

To add a new state to the General value lifeline:

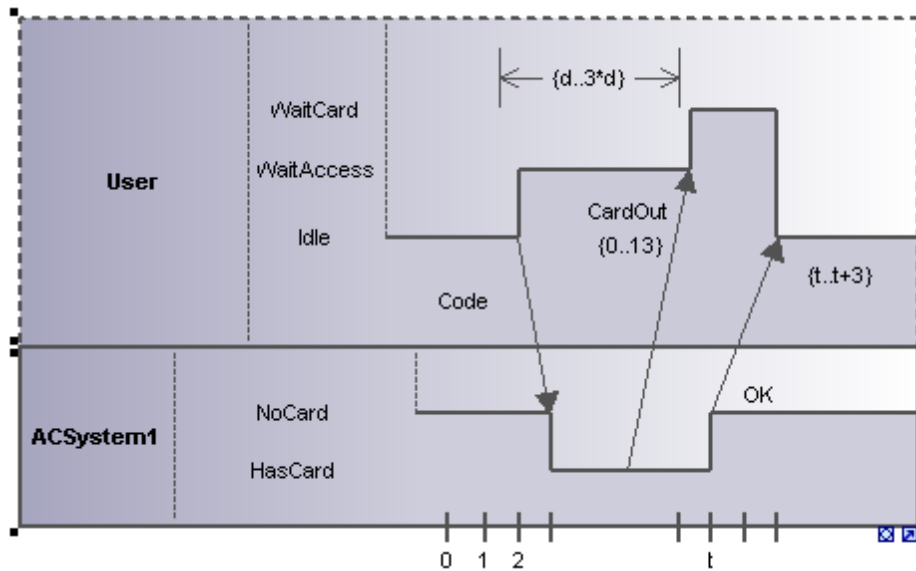
1. Right click the lifeline and select **New | State/Condition (StateInvariant)**.
2. Edit the new name e.g. State3, and press Enter to confirm.



A new State is added to the lifeline.

Grouping lifelines

Placing, or stacking lifelines, automatically positions them correctly and preserves any tick marks that might have been added. Messages can also be created between separate lifelines by dragging the respective message object.



Tick Mark

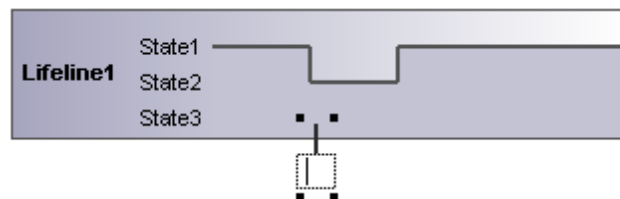


TickMark

The tick mark is used to insert the tick marks of a timing ruler scale onto a lifeline.

To insert a TickMark:

1. Click the tick mark icon and click on the lifeline to insert it.

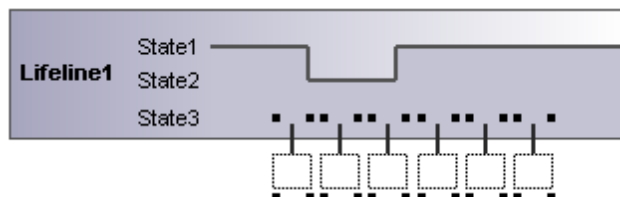


2. Insert multiple tick marks by holding down the **Ctrl** key and repeatedly clicking at different positions on the lifeline border.
3. Enter the tick mark label in the field provided for it.
Drag tick marks to reposition them on the lifeline.

To evenly space tick marks on a lifeline:

1. Use the marque, by dragging in the main window, to mark the individual tick marks.

2. Click the **Space Across** icon  in the icon bar.



Event/Stimulus

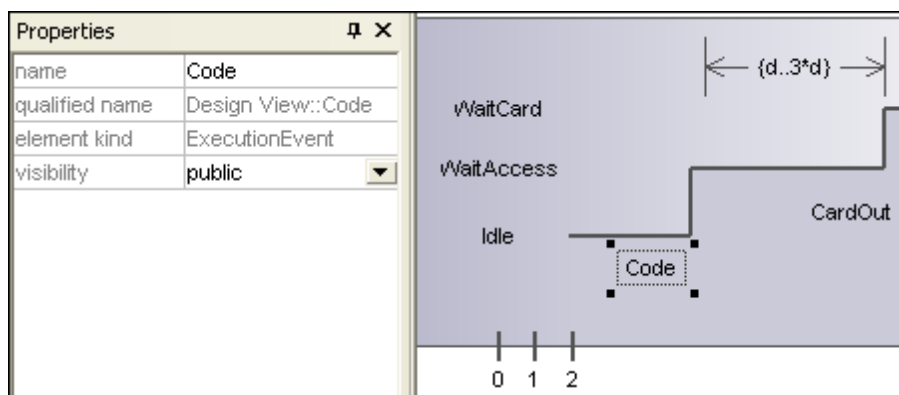


Event / Stimulus

The Event/Stimulus ExecutionEvent is used to show the change in state of an object caused by the respective event or stimulus. The received events are annotated to show the event causing the change in condition or state.

To insert an Event/Stimulus:

1. Click the Event/Stimulus icon, then click the specific position in the timeline where the state change takes place.



2. Enter a name for the event, in this example the event is "Code".
Note that the event properties are visible in the Properties tab.

DurationConstraint

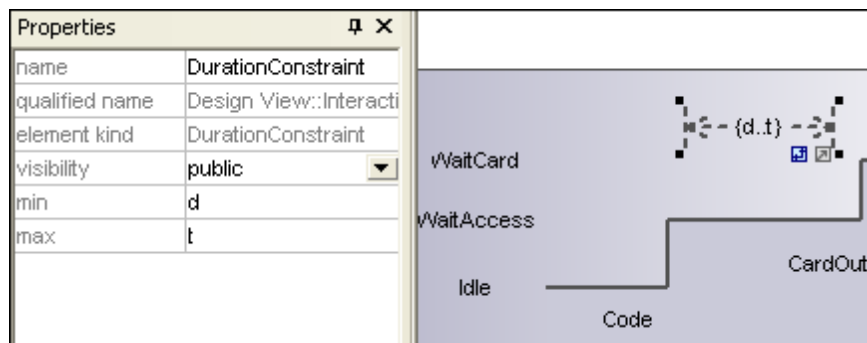


DurationConstraint

A Duration defines a ValueSpecification that denotes a duration in time between a start and endpoint. A duration is often an expression representing the number of clock ticks, which may elapse during this duration.

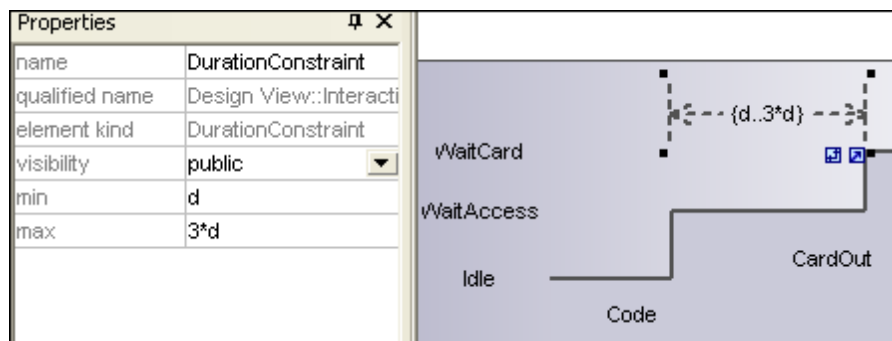
To insert an DurationConstraint:

1. Click the DurationConstraint icon, then click the specific position on the lifeline where the constraint is to be displayed.



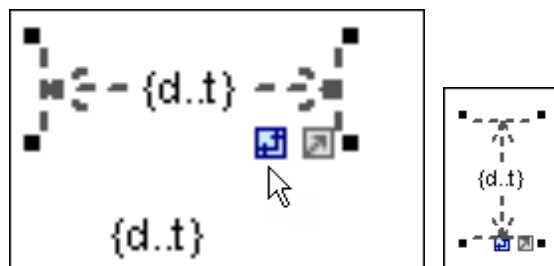
The default minimum and maximum values, "d..t", are automatically supplied. These values can be edited by double clicking the time constraint, or by editing the values in the Properties window.

2. Use the "handles" to resize the object if necessary.



Changing the orientation of the DurationConstraint:

1. Click the "Flip" icon to orient the constraint vertically.



TimeConstraint

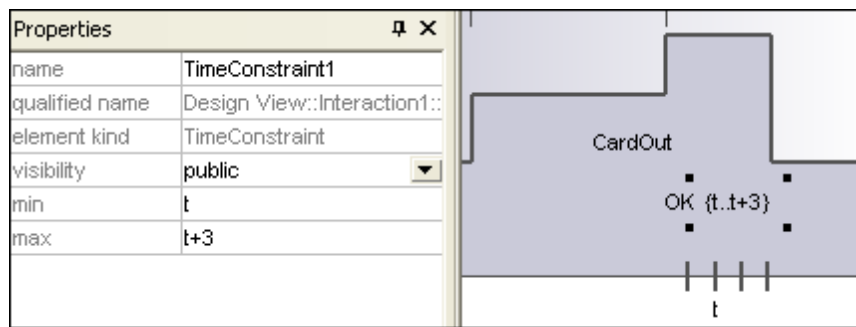


TimeConstraint

A TimeConstraint is generally shown as graphical association between a TimeInterval and the construct that it constrains. Typically this graphical association between an EventOccurrence and a TimeInterval.

To insert a TimeConstraint:

1. Click the TimeConstraint icon, then click the specific position on the lifeline where the constraint is to be displayed.



The default minimum and maximum values are automatically supplied, "d..t" respectively. These values can be edited by double clicking the time constraint, or by editing the values in the Properties window.

Message



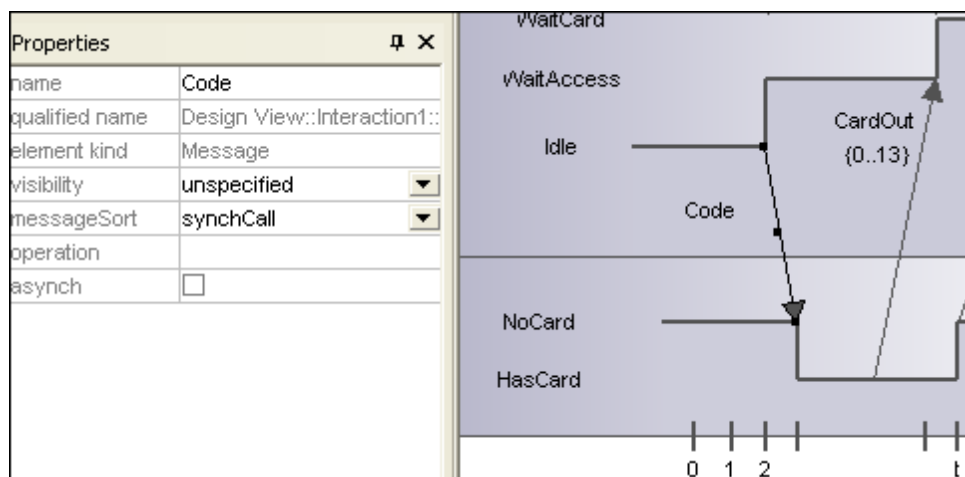
A Message is a modeling element that defines a specific kind of communication in an Interaction. A communication can be e.g. raising a signal, invoking an Operation, creating or destroying an Instance. The Message specifies the type of communication defined by the dispatching ExecutionSpecification, as well as the sender and the receiver.

Messages are sent between sender and receiver timelines, and are shown as labeled arrows.

To insert a message:

1. Click the specific message icon in the toolbar.
2. Click anywhere on the timeline sender object e.g. Idle.
3. Drag and drop the message line onto the receiver objects timeline e.g. NoCard.

Lifelines are highlighted when the message can be dropped.



- The direction in which you drag the arrow defines the message direction. Reply messages can point in either direction.
- Having clicked a message icon and holding down **Ctrl** key, allows you to insert multiple messages by repeatedly clicking and dragging in the diagram tab.

To delete a message:

1. Click the specific message to select it.

2. Press the **Del** key to delete it from the model, or right click it and select "Delete from diagram".

10.2 Structural Diagrams

These diagrams depict the structural elements that make up a system or function. Both the static, e.g. Class diagram, and dynamic, e.g. Object diagram, relationships are presented.

Structural Diagrams

[Class Diagram](#)[Component Diagram](#)[Composite Structure Diagram](#)[Deployment Diagram](#)[Object Diagram](#)[Package Diagram](#)[Profile Diagram](#)

10.2.1 Class Diagram

This section includes tasks and concepts applicable to Class Diagrams, as follows:

- [Customizing Class Diagrams](#)
- [Overriding Base Class Operations and Implementing Interface Operations](#)
- [Creating Getter and Setter Methods](#)
- [Ball and Socket Notation](#)
- [Adding Raised Exceptions to Methods of a Class](#)
- [Generating Class Diagrams](#)

For a basic introduction to Class Diagrams, see [Class Diagrams](#) in the tutorial section of this documentation.

Customizing Class Diagrams

Expanding / hiding class compartments in a UML diagram:

There are several methods of expanding the various compartments of class diagrams.

- Click on the **+** or **-** buttons of the currently active class to expand/collapse the specific compartment.
- Use the marquee (drag on the diagram background) to mark **multiple** classes, then click the expand/hide button. You can also use **Ctrl+Click** to select multiple classes.
- Press **Ctrl+A** to select **all classes**, then click the expand/collapse button, on one of the classes, to expand/collapse the respective compartments.

Expanding / collapsing class compartments in the Model Tree:

In the Model Tree classes are subelements of packages and you can affect either the packages or the classes.

Click the package / class you want to **expand** and:

Press the ***** key to expand the current package/class and all sub-elements

Press the **+** key to open the current package/class.

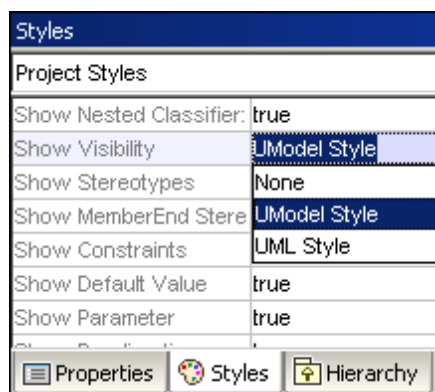
To **collapse** the packages/classes, press the **-** keyboard key.

Note that you can use the standard keyboard keys, or the numeric keypad keys to achieve this.

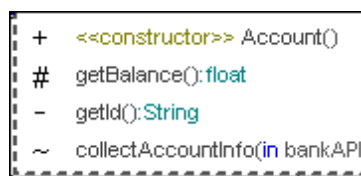
Changing the visibility type icons

Clicking the **visibility icon** to the left of an operation , or property , opens a drop-down list enabling you to change the visibility status. You can also change the type of visibility symbols that you want to see.

- Click a class in the diagram window, click the **Styles** tab and scroll down the list until you find the **Show Visibility** entry.



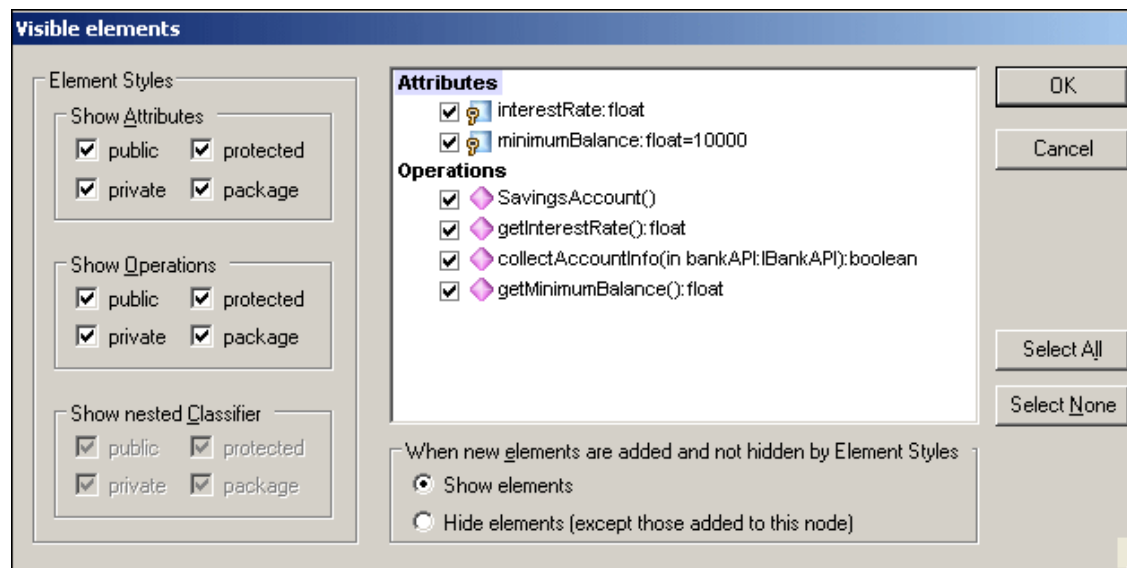
You can choose between the UModel type shown above, or the UML conformant symbols shown below.



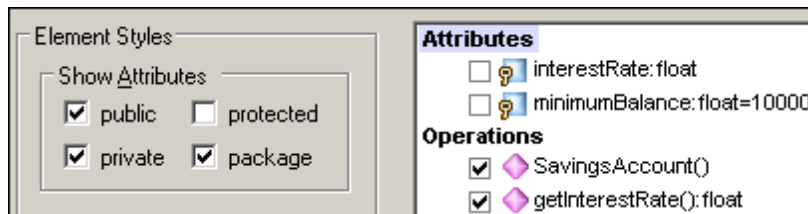
Showing / Hiding node content (class attributes, operations, slots)

UModel now allows you to individually display the attributes or operations of a class, as well as define which should be shown when adding them as new elements. Note that this it now possible to show/hide object slots i.e. InstanceSpecifications using the same method.

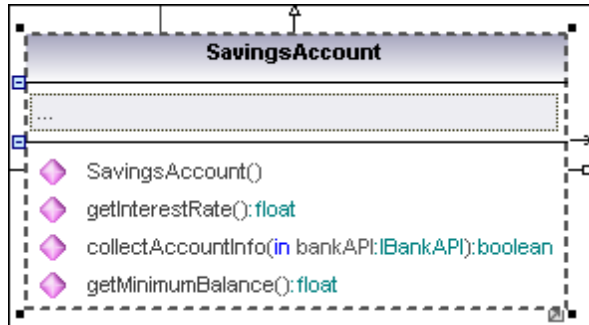
Right click a class, e.g. SavingsAccount, and select the menu option **Show/Hide Node content**.



Deselecting a **protected** check box in the **Attributes** group, deselects the protected attributes in the preview window.



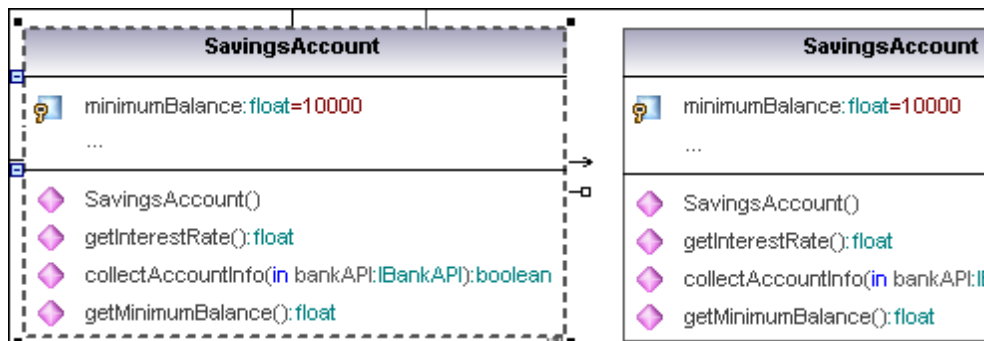
Having confirmed with OK, the protected attributes in the class are replaced with ellipsis "...". Double clicking the ellipsis opens the dialog box.



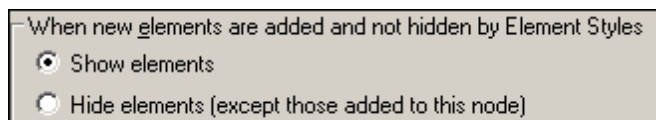
Note that individual attributes can be affected by only deselecting the check box in the preview window.

Showing / Hiding class attributes or operations - Element styles

UModel allows you to insert **multiple instances** of the same class on a **single** diagram, or even different diagrams. The visibility settings can be individually defined for each of these "views" to the class. The screenshot below shows two views to the same class i.e. SavingsAccount.



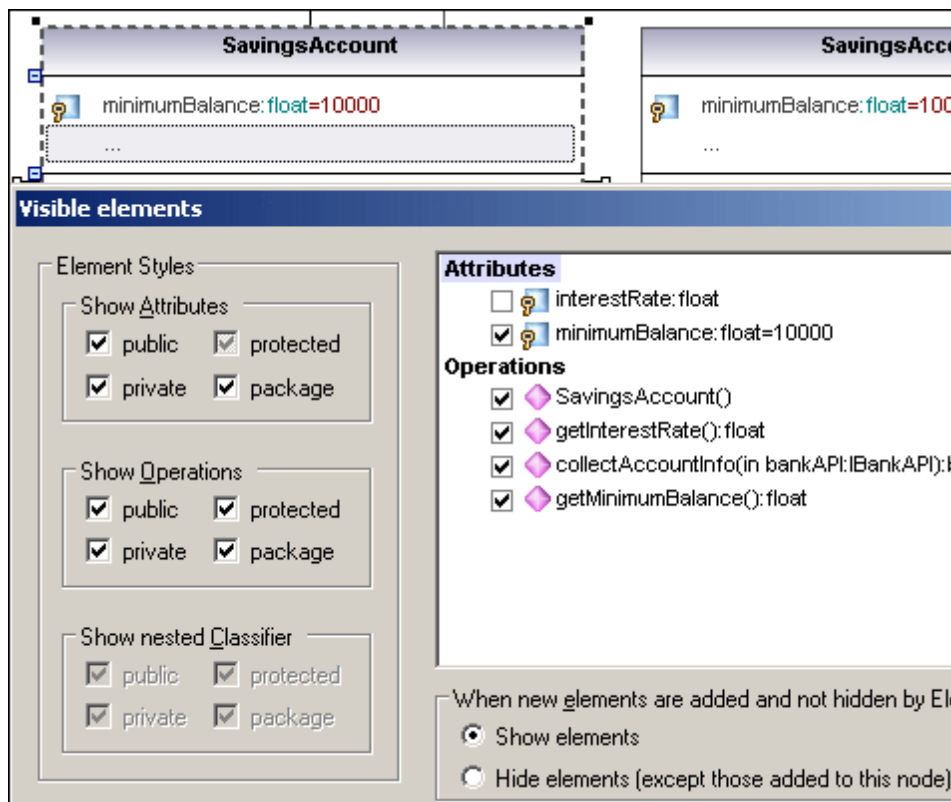
The "When new elements are added and not hidden by Element Styles" option allows you to define what will be made visible when new elements are added to the class. Elements can be added manually in the model diagram and in the Model Tree, or automatically during the code engineering process.



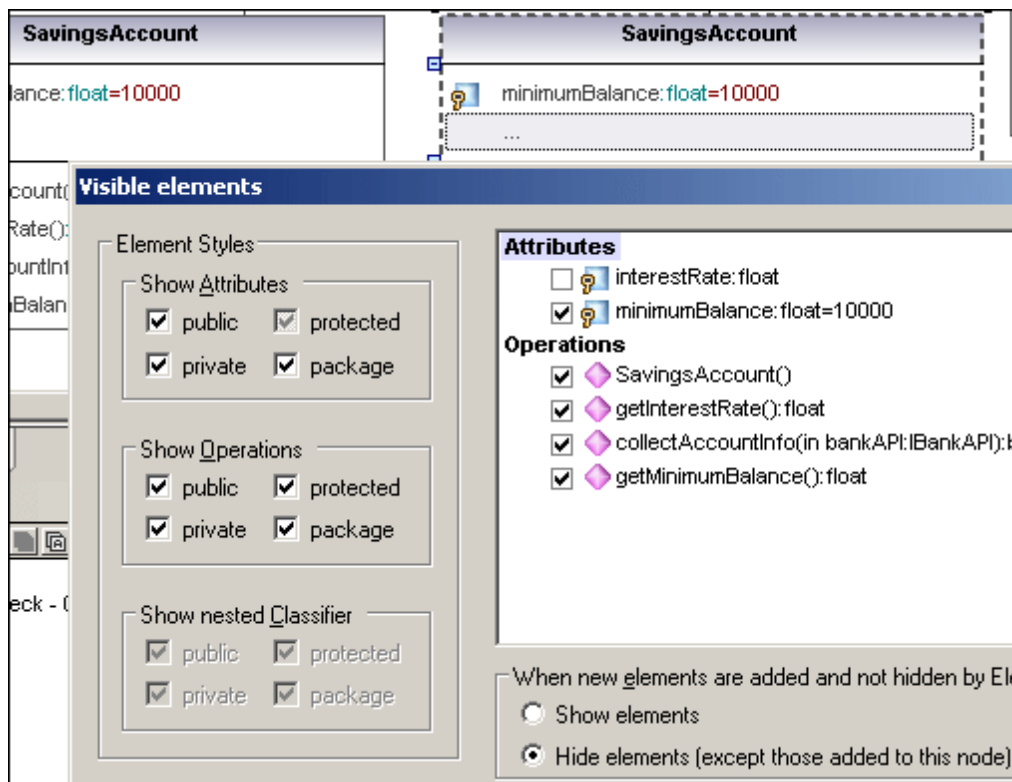
Show elements: displays all new elements that are added to any view of the class.

E.g. The interestRate:float attribute has been hidden in both "views" of SavingsAccount, leaving the minimumBalance attribute visible. The "Show elements" radio button is active for the left-hand class.

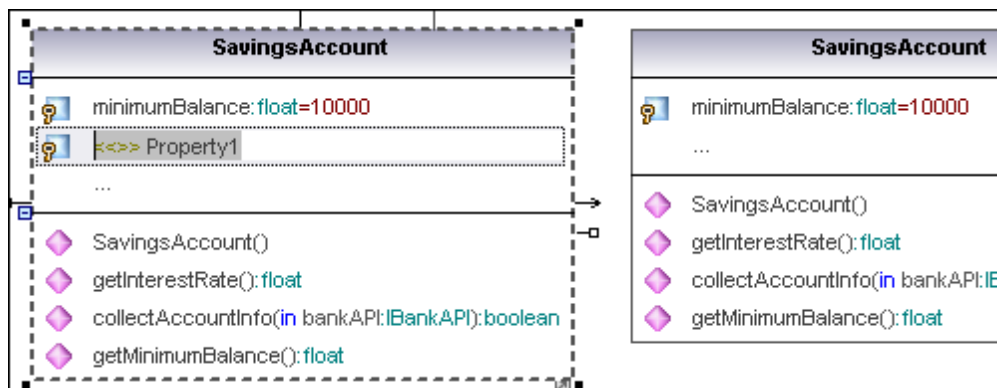
Double clicking the ellipsis "..." in the attribute compartment of the **left**-hand class shows that the "Show elements" radio button is active.



Double clicking the ellipsis "..." in the attribute compartment of the **right**-hand class shows that the "Hide elements (except those added to this node)" radio button is active.

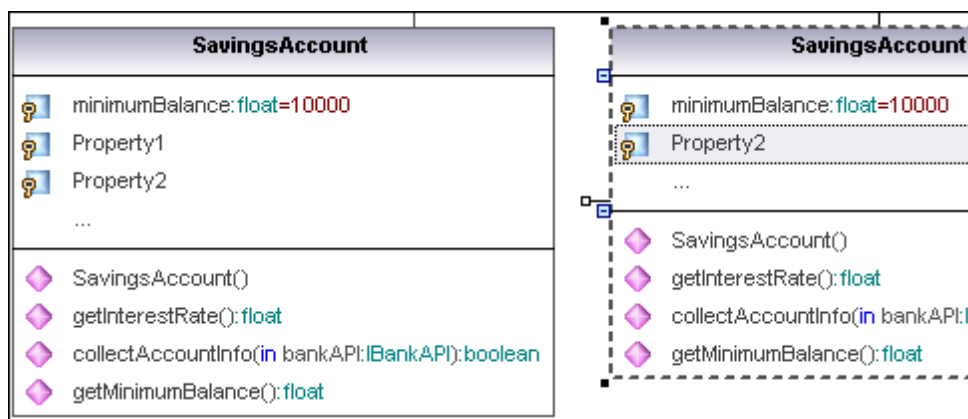


Clicking the **left**-hand class and pressing F7, (or clicking the class in the Model Tree and pressing F7) adds a new attribute (Property1) to the class.



The new element is only visible in the left-hand class, because "Show elements" is set as active. The right-hand class setting is "Hide elements...", so the new element is not shown there.

Clicking the **right**-hand class and pressing F7 adds a new attribute (Property2) to the class. This new attribute is now visible because the Hide elements... setting has the qualifier "**except those added to this node**", where "node" generically means this class, or modelling element.



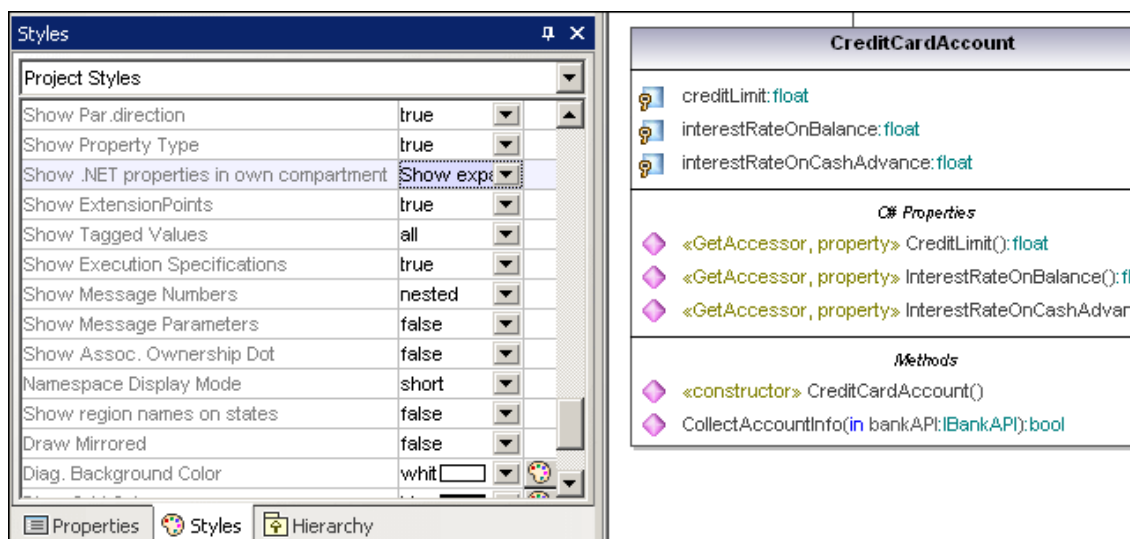
The `Property2` attribute is also visible in the left hand class, because the setting there is "Show elements"

Please note:

Tagged values of hidden elements are also hidden when you select the hide option.

Showing / Hiding VS .NET compartments:

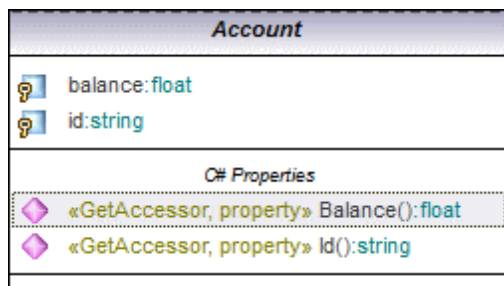
UModel can now display .NET properties in their own compartment, by selecting the "Show .NET properties in own compartment" option in the Styles tab.



Showing VS .NET properties as associations

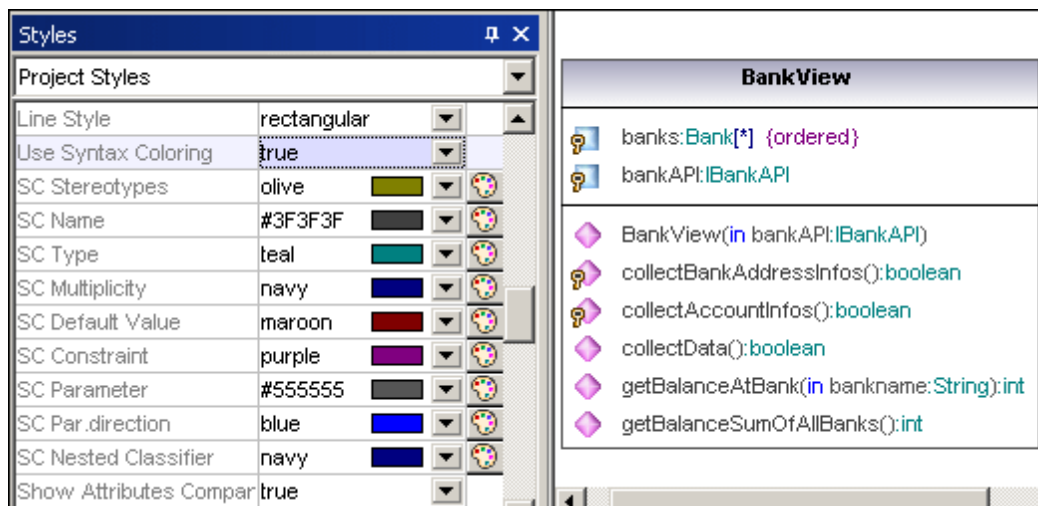
UModel can now display .NET properties as associations.

Right click a C# property as shown below, and select "Show | All .NET Properties as Associations" drag and drop the operation onto the diagram.



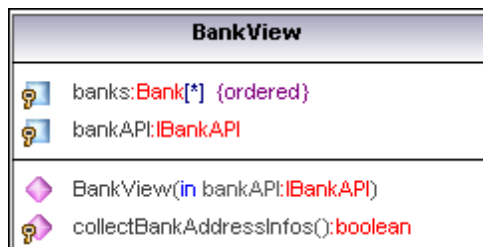
Changing the syntax coloring of operations/properties

UModel automatically enables syntax coloring, but lets you customize it to suit your needs. The default settings are shown below.



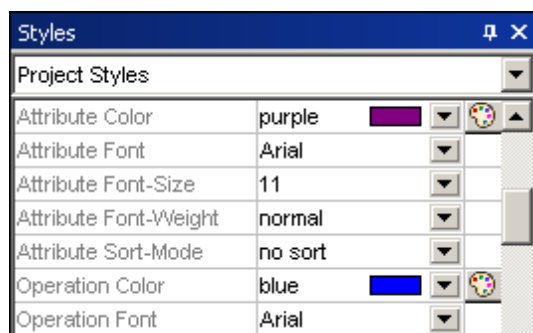
To change the default syntax coloring options (shown below):

1. Switch to the **Styles** tab and scroll the **SC** prefixed entries.
2. Change one of the SC color entries e.g. SC Type to red.



To disable syntax coloring:

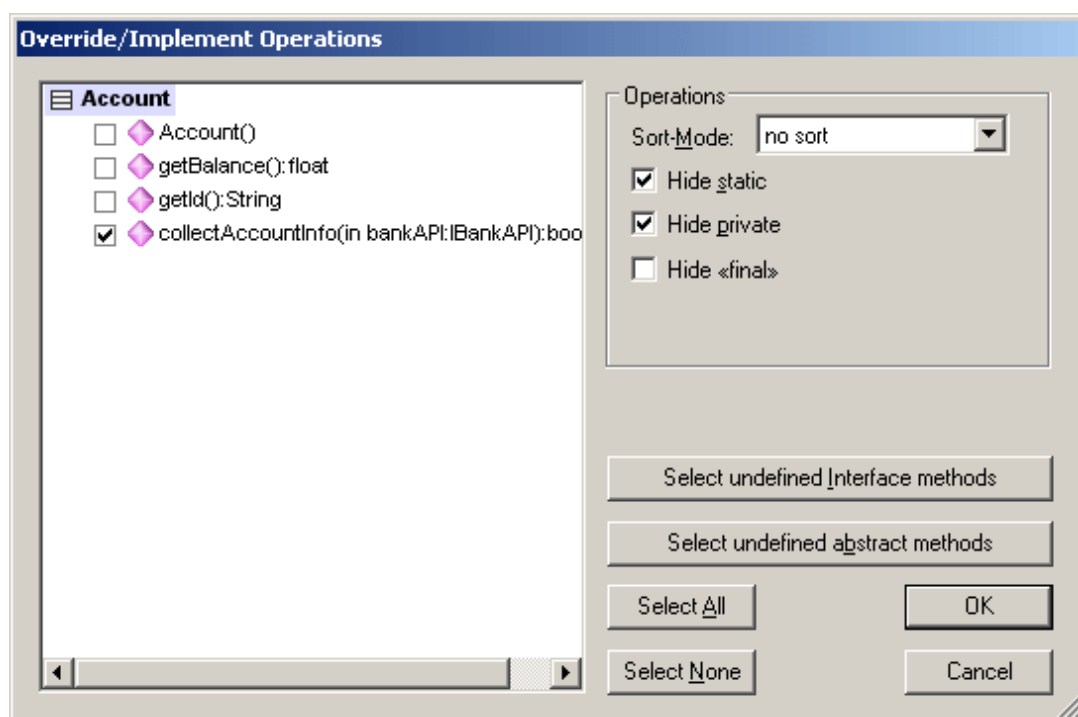
1. Switch to the **Styles** tab and change the **Use Syntax Coloring** entry to **false**.
2. Use the **Attribute Color**, or **Operation Color** entries in the **Styles** tab to customize these items in the class.



Overriding Base Class Operations and Implementing Interface Operations

UModel gives you the ability to override the base-class operations, or implement interface operations of a class. This can be done from the Model Tree, Favorites tab, or in Class diagrams.

1. Right click one of the derived classes in the class diagram, e.g. CheckingAccount, and select **Override/Implement Operations**. This opens the dialog box shown below.



2. Select the Operations that you want to override and confirm with OK.
The "Select undefined..." buttons select those method types in the window at left.

Please note:

When the dialog box is opened, operations of base classes and implemented interfaces that have the same signature as existing operations, are automatically checked (i.e. active).

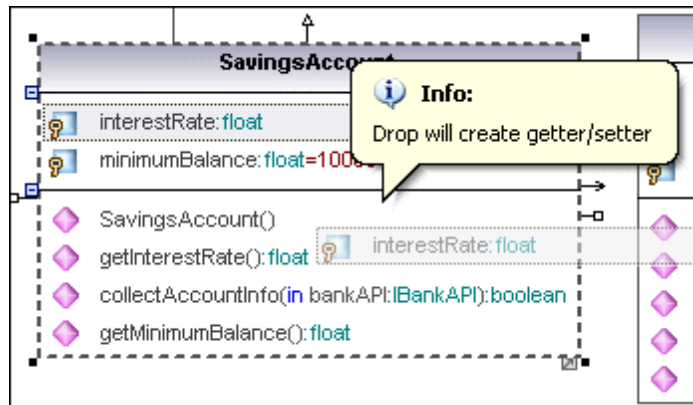
Creating Getter and Setter Methods

During the modeling process it is often necessary to create get/set methods for existing attributes. UModel supplies you with two separate methods to achieve this:

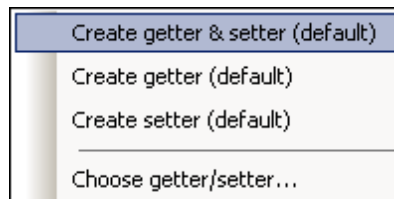
- Drag and drop an attribute into the operation compartment
- Use the context menu to open a dialog box allowing you to manage get/set methods

To create getter/setter methods using drag and drop:

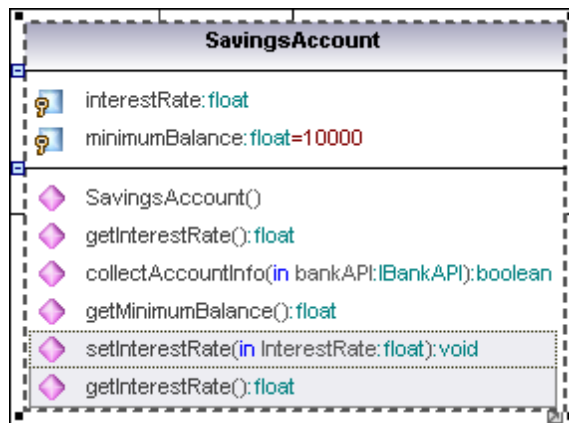
1. Drag an attribute from the Attribute compartment and drop it in the Operations compartment.



A pop-up menu appears at this point allowing you to decide what type of get/set method you want to create.

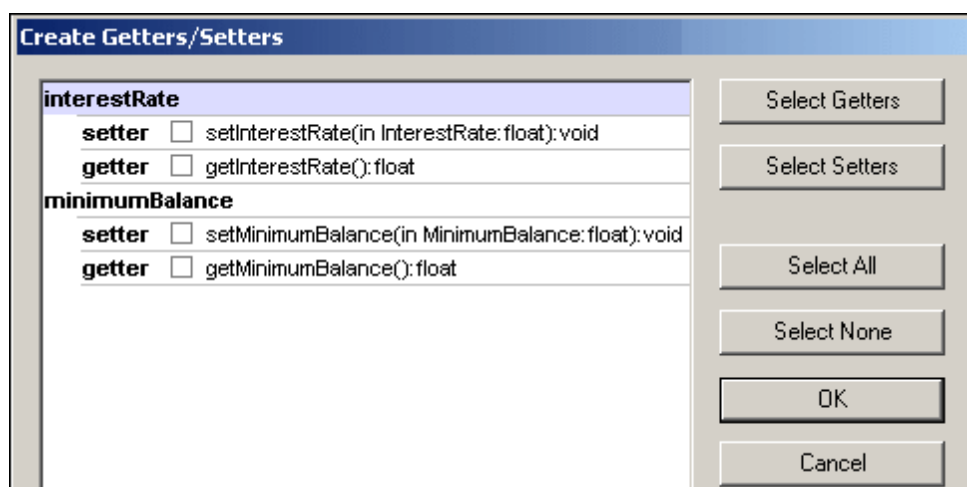


Selecting the first item creates a get and set method for `interestRate:float`.



To create getter/setter methods using the context menu:

1. Right-click the class title, e.g. **SavingsAccount**, and select the context menu option **Create Getter/Setter Operations**.



The Create Getters/Setters dialog box opens displaying all attributes available in the currently active class.

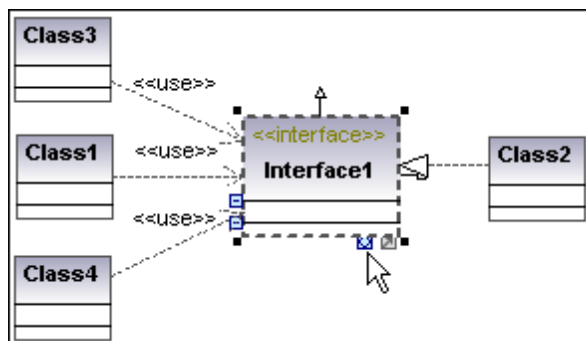
2. Use the buttons to select the items as a group, or click the getter/setter check boxes individually.

Please note:

You can also right-click a single attribute and use the same method to create an operation for it.

Ball and Socket Notation

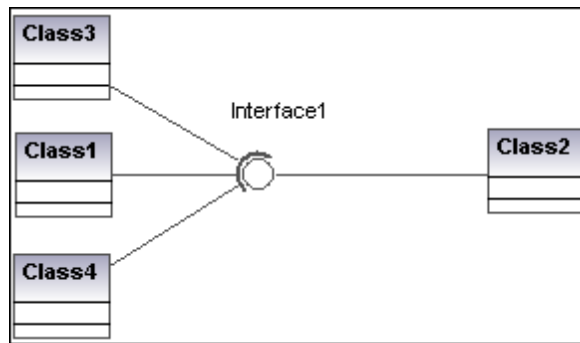
UModel supports the ball and socket notation of UML. Classes that require an interface display a "socket" and the interface name, while classes that implement an interface display the "ball".



In the shots shown above, Class2 realizes Interface1, which is used by classes 1, 3, and 4. The usage icons were used to create the usage relationship between the classes and the interface.

To switch between the standard and ball-and-socket view:

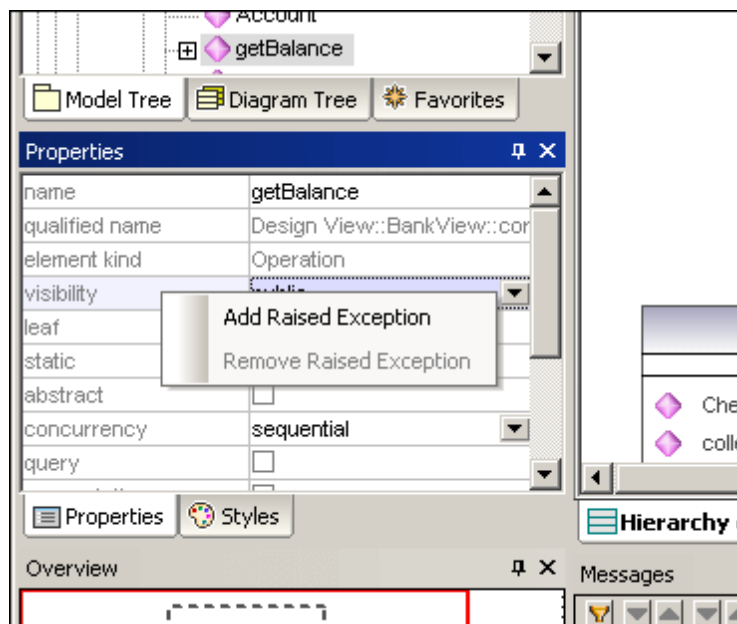
- Click the Toggle Interface notation icon at the base of the interface element.



Adding Raised Exceptions to Methods of a Class

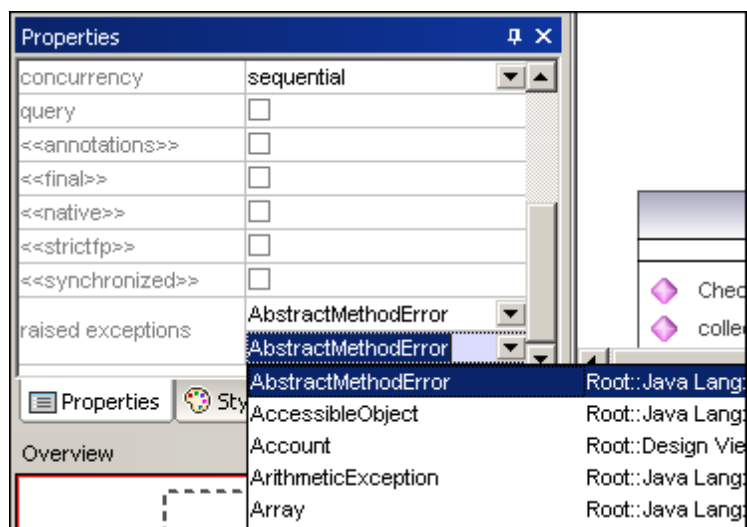
To add raised Exceptions to methods of a class:

1. Click the method of the class you want to add the raised exception to in the Model Tree window, e.g. getBalance of the Account class.
2. Right-click the Properties window and select **Add Raised Exception** from the pop-up menu.



This adds the **raised exceptions** field to the Properties window, and automatically selects the first entry in the list.

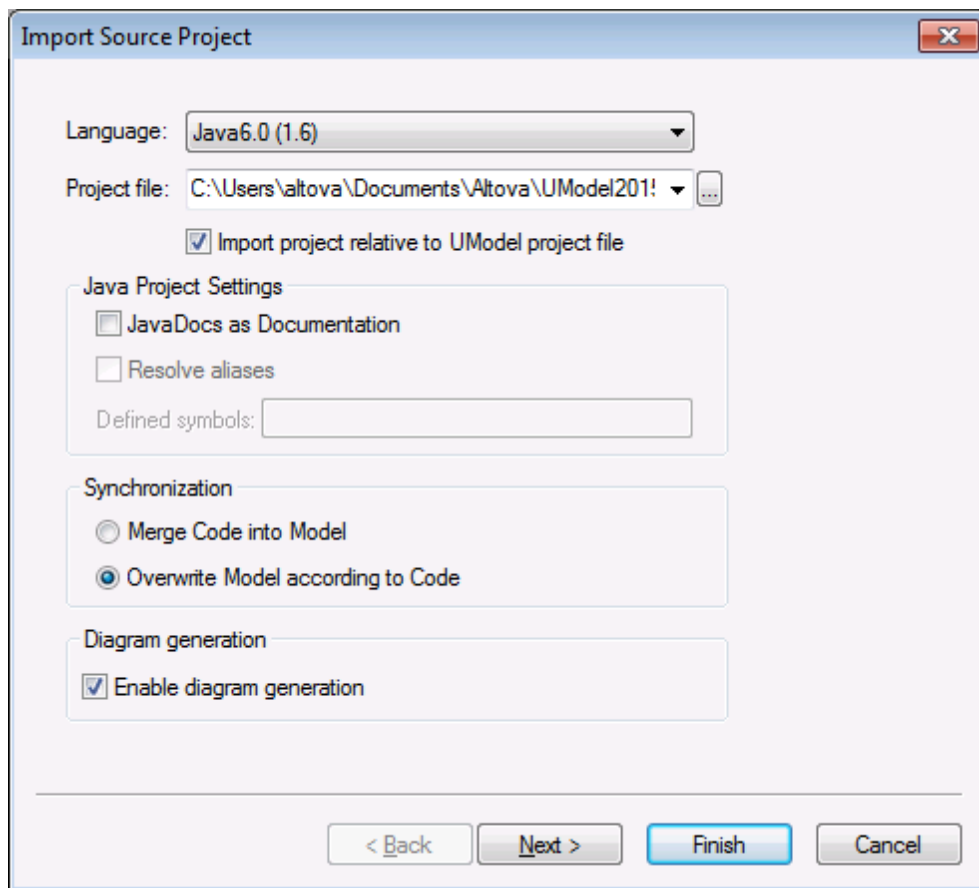
3. Select an entry from the list, or enter your own into the field.



Generating Class Diagrams

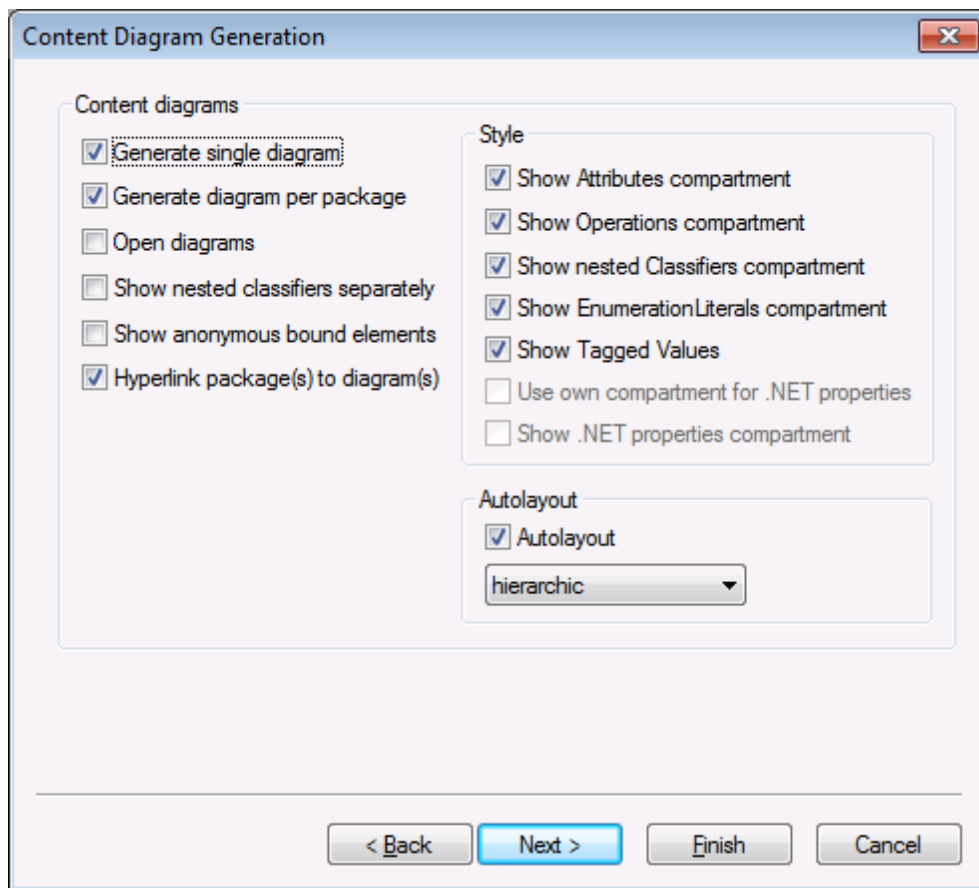
As an alternative to designing Class Diagrams directly in UModel, you can generate them automatically when importing source code or binaries into UModel projects (see [Importing source code into projects](#) and [Importing Java, C# and VB binaries](#)). When following the import wizard, make sure that:

- 1) The **Enable diagram generation** check box is selected on the "Import Source Project", "Import Binary Types", or "Import Source Directory" dialog box.



Import Source Project dialog box

2) The **Generate single diagram** and/or the **Generate diagram per package** options are selected on the "Content Diagram Generation" dialog box.



Content Diagram Generation dialog box

Once the import operation is finished, any generated class diagrams are available under "Class Diagrams" in the Diagram Tree.

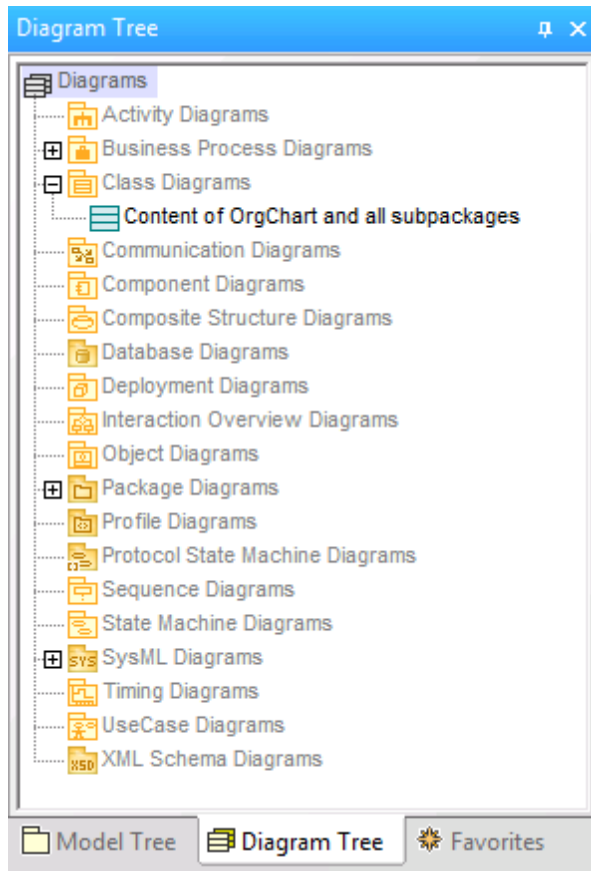
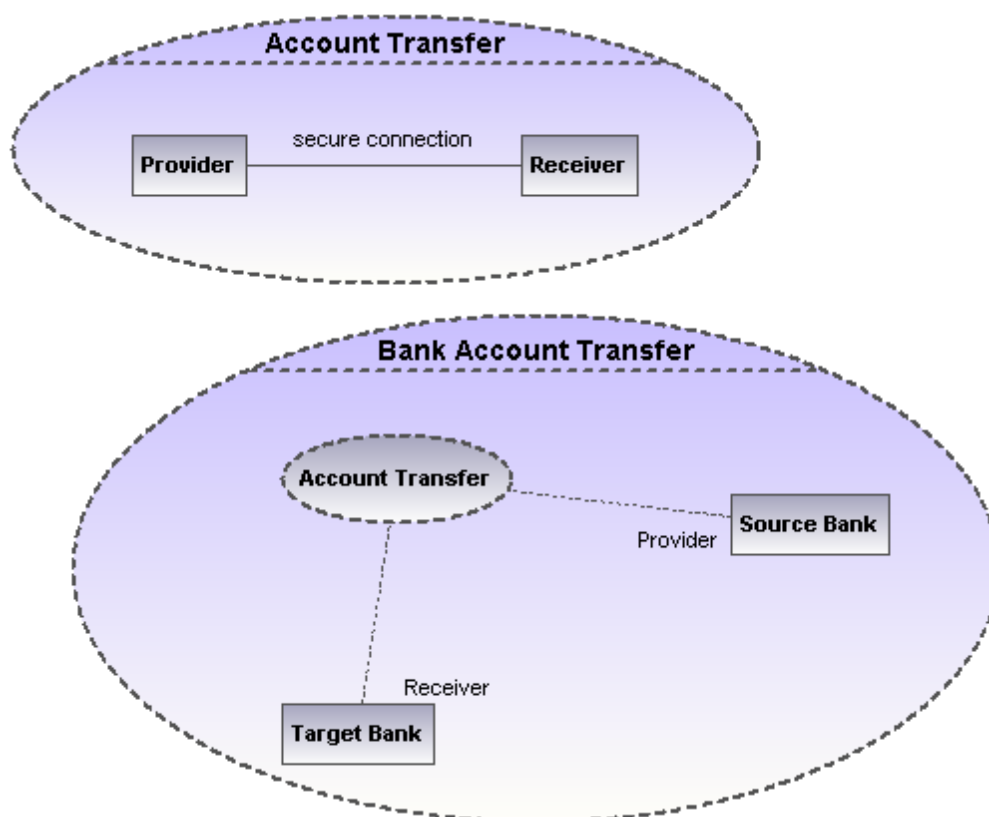


Diagram Tree

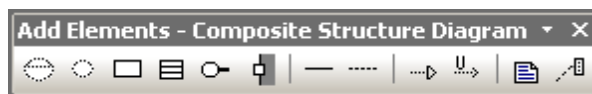
10.2.2 Composite Structure Diagram

Altova web site:  [UML Composite Structure diagrams](https://www.altova.com/uml-composite-structure-diagrams)

The Composite Structure Diagram has been added in UML 2.0 and is used to show the internal structure, including parts, ports and connectors, of a structured classifier, or collaboration.



Inserting Composite Structure Diagram elements



Using the toolbar icons:

1. Click the specific Composite Structure diagram icon in the toolbar.
2. Click in the Composite Structure diagram to insert the element. To insert multiple elements of the selected type, hold down the **Ctrl** key and click in the diagram window.

Dragging existing elements into the Composite Structure diagram:

Most elements occurring in other Composite Structure diagrams, can be inserted into an existing Composite Structure diagram.

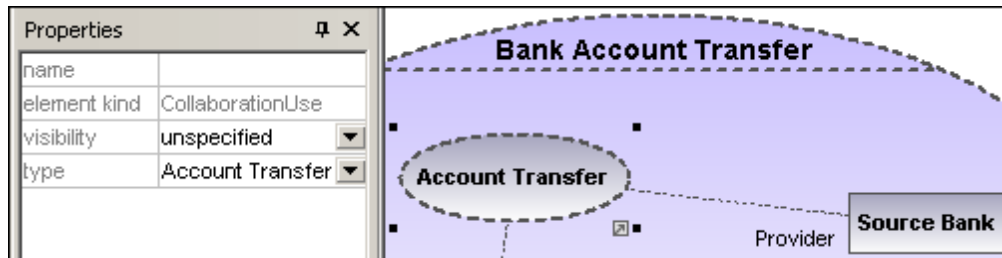
1. Locate the element you want to insert in the Model Tree tab (you can use the search function text box, or press Ctrl+F to search for any element).
2. Drag the element(s) into the Composite Structure diagram.

**Collaboration**

Inserts a collaboration element which is a kind of classifier/instance that communicates with other instances to produce the behavior of the system.

**CollaborationUse**

Inserts a Collaboration use element which represents one specific use of a collaboration involving specific classes or instances playing the role of the collaboration. A collaboration use is shown as a dashed ellipse containing the name of the occurrence, a colon, and the name of the collaboration type.



When creating dependencies between collaboration use elements, the "type" field must be filled to be able to create the role binding, and the target collaboration must have at least one part/role.

**Part (Property)**

Inserts a part element which represents a set of one or more instances that a containing classifier owns. A Part can be added to collaborations and classes.

**Port**

Inserts a port element which defines the interaction point between a classifier and its environment, and can be added on parts with a defined type.

**Class**

Inserts a Class element, which is the actual classifier that occurs in that particular use of the collaboration.

**Connector**

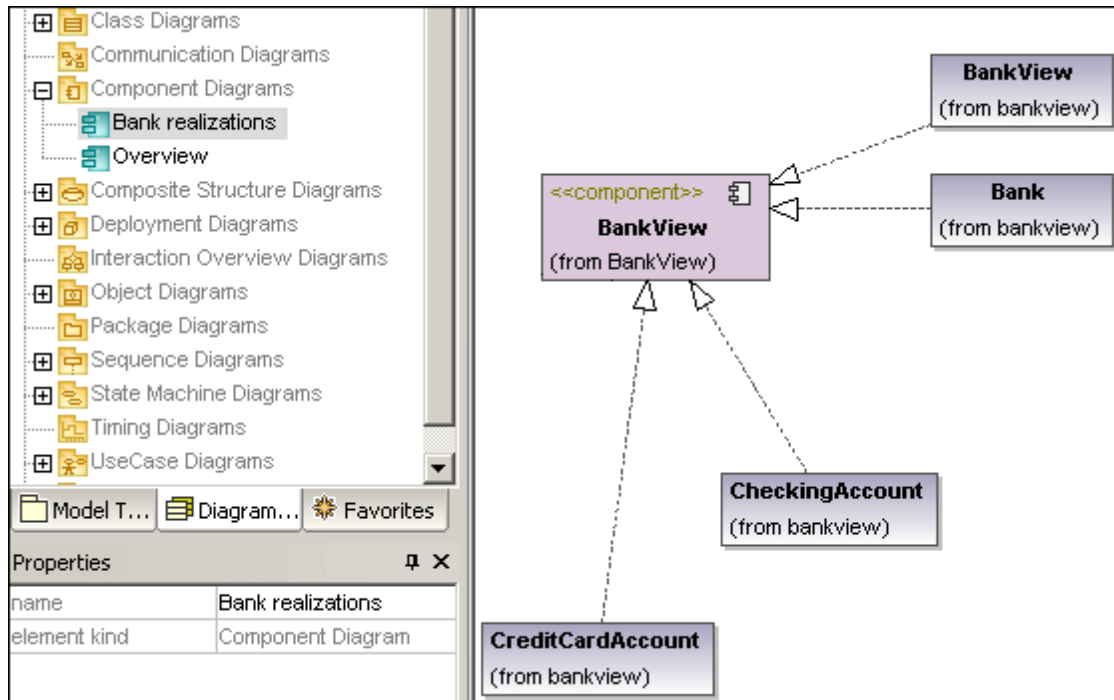
Inserts a Connector element which can be used to connect two or more instances of a part, or a port. The connector defines the relationship between the objects and identifies the communication between the roles.

**Dependency (Role Binding)**

Inserts the Dependency element, which indicates which connectable element of the classifier or operation, plays which role in the collaboration.

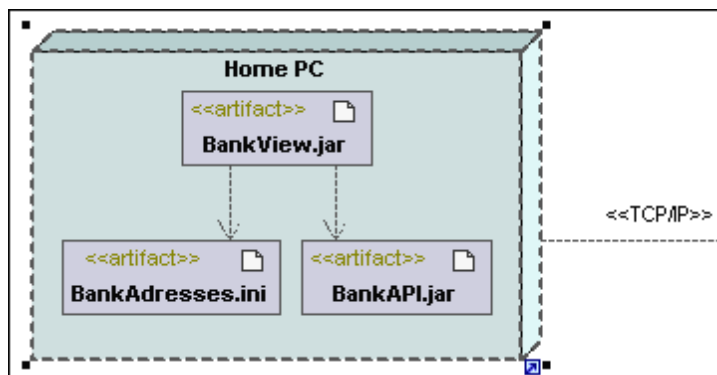
10.2.3 Component Diagram

Please see the [Component Diagrams](#) section in the tutorial for more information on how to add component elements to the diagram.



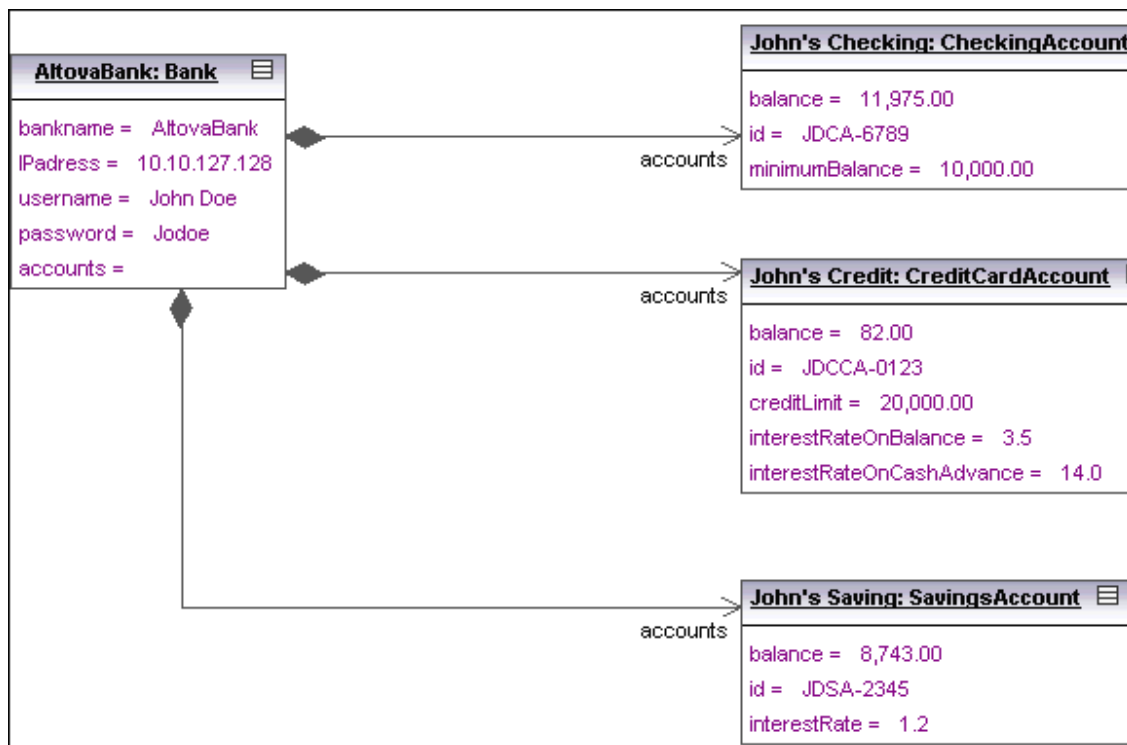
10.2.4 Deployment Diagram

Please see the [Deployment Diagrams](#) section in the tutorial for more information on how to add nodes and artifacts to the diagram.



10.2.5 Object Diagram

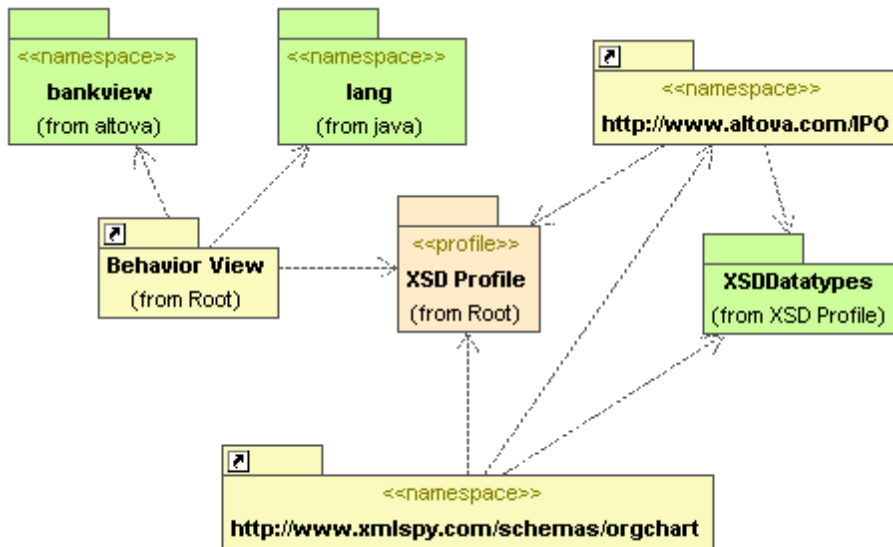
Please see the [Object Diagrams](#) section in the tutorial for more information on how to add new objects/instances to the diagram.



10.2.6 Package Diagram

Package diagrams display the organization of packages and their elements, as well as their corresponding namespaces. UModel additionally allows you to create a hyperlink and navigate to the respective package content.

Packages are depicted as folders and can be used on any of the UML diagrams, although they are mainly used on use-case and class diagrams.



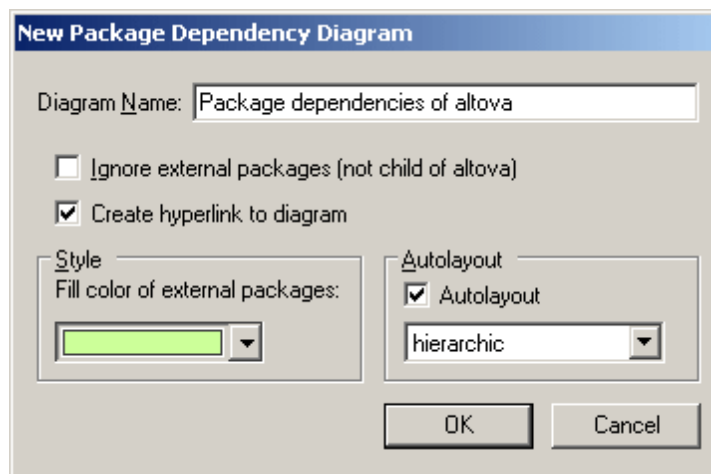
Automatic Package Dependency diagram generation

You can generate a package dependency diagram for any package that already exists in the Model Tree.

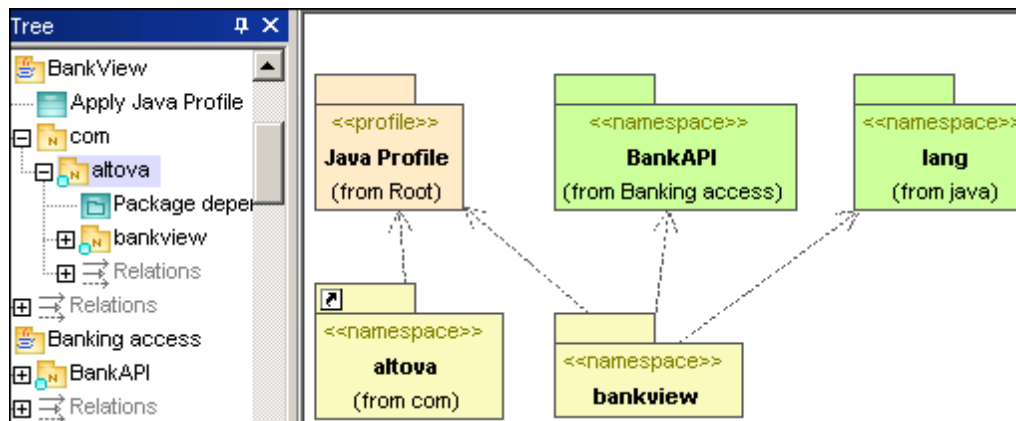
Dependency links between packages are created if there are any references between the modeling elements of those packages. E.g. Dependencies between classes, derived classes, or if attributes have types that are defined in a different package.

To generate a package dependency diagram:

1. Right click a package in the Model Tree, e.g. altova, and select **Show in new Diagram | Package Dependencies...**
This opens the New Package Dependency Diagram dialog box.



2. Select the specific options you need and click OK to confirm.



A new diagram is generated and displays the package dependencies of the altova package.

Inserting Package Diagram elements

Using the toolbar icons:

1. Click the specific icon in the Package Diagram toolbar.



2. Click in the diagram to insert the element. To insert multiple elements of the selected type, hold down the **Ctrl** key and click in the diagram window.

Dragging existing elements into the Package Diagram:

Elements occurring in other diagrams, e.g. other packages, can be inserted into a Package diagram.

1. Locate the element you want to insert in the Model Tree tab (you can use the search function text box, or press **Ctrl+F** to search for any element).
2. Drag the element(s) into the diagram.

**Package**

Inserts the package element into the diagram. Packages are used to group elements and also to provide a namespace for the grouped elements. Being a namespace, a package can import individual elements of other packages, or all elements of other packages. Packages can also be merged with other packages.

**Profile**

Inserts the Profile element, which is a specific type of package that can be applied to other packages.

The Profiles package is used to extend the UML meta model. The primary extension construct is the Stereotype, which is itself part of the profile. Profiles must always be related to a reference meta model such as UML, they cannot exist on their own.

**Dependency**

Inserts the Dependency element, which indicates a supplier/client relationship between modeling elements, in this case packages, or profiles.

**PackageImport**

Inserts an <<import>> relationship which shows that the elements of the included package will be imported into the including package. The namespace of the including package gains access to the included namespace; the namespace of the included package is not affected.

Note: elements defined as "private" within a package, cannot be merged or imported.

**PackageMerge**

Inserts a <<merge>> relationship which shows that the elements of the merged (source) package will be imported into the merging (target) package, including any imported contents the merged (source) package.

If the same element exists in the target package then these elements' definitions will be expanded by those from the target package. Updated or added elements are indicated by a generalization relationship back to the source package.

Note: elements defined as "private" within a package, cannot be merged or imported.

**ProfileApplication**

Inserts a Profile Application which shows which profiles have been applied to a package. This is a type of package import that states that a Profile is applied to a Package.

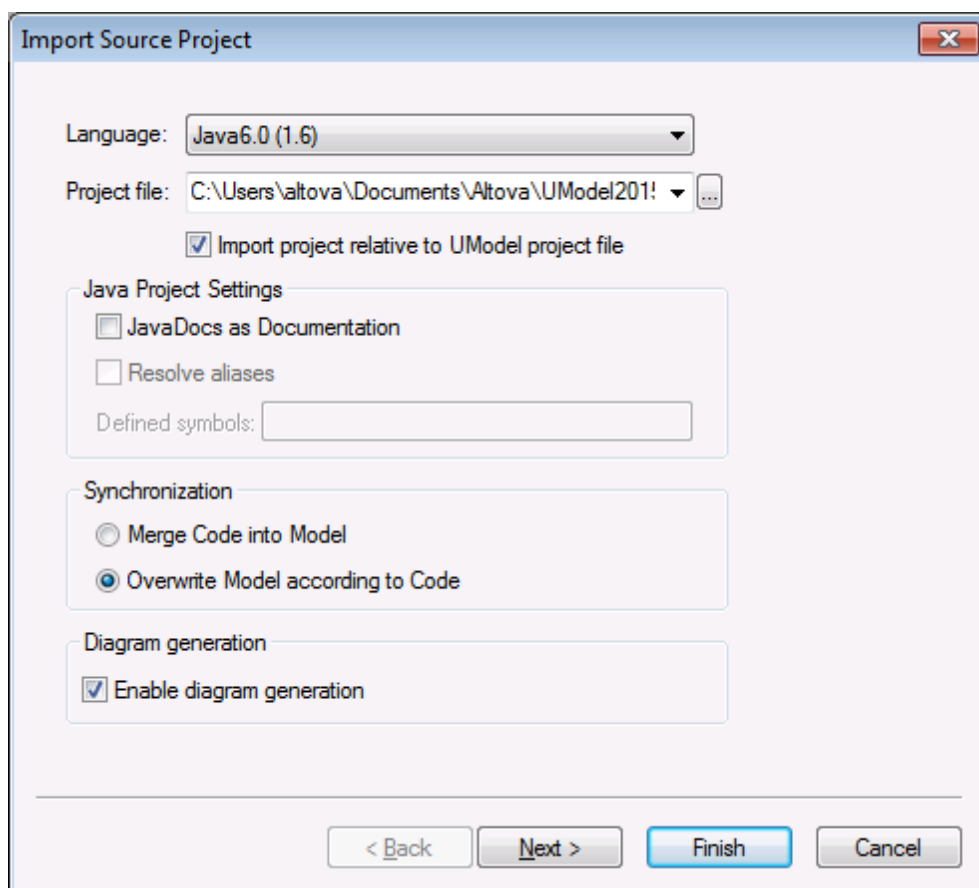
The Profile extends the package it has been applied to. Applying a profile, using the ProfileApplication icon, means that all stereotypes that are part of it, are also available to the package.

Profile names are shown as dashed arrows from the package to the applied profile, along with the <<apply>> keyword.

Generating Package Diagrams While Importing Code or Binaries

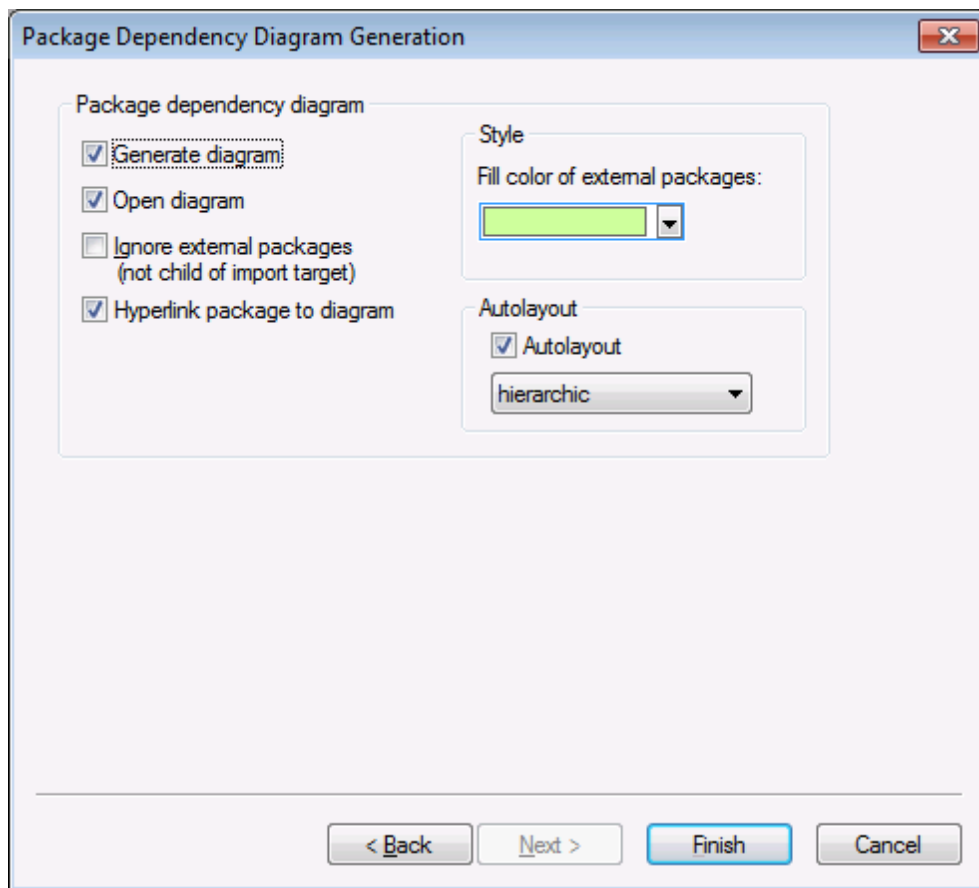
You can instruct UModel to generate package diagrams when importing source code or binaries into the UModel project (see [Importing source code into projects](#) and [Importing Java, C# and VB binaries](#)). When following the import wizard, make sure that:

- 1) The **Enable diagram generation** check box is selected on the "Import Source Project", "Import Binary Types", or "Import Source Directory" dialog box.



Import Source Project dialog box

- 2) The **Generate diagram** option is selected on the "Package Dependency Diagram Generation" dialog box.



Package Dependency Diagram Generation dialog box

Once the import operation is finished, any generated package diagrams are available under "Package Diagrams" in the Diagram Tree.

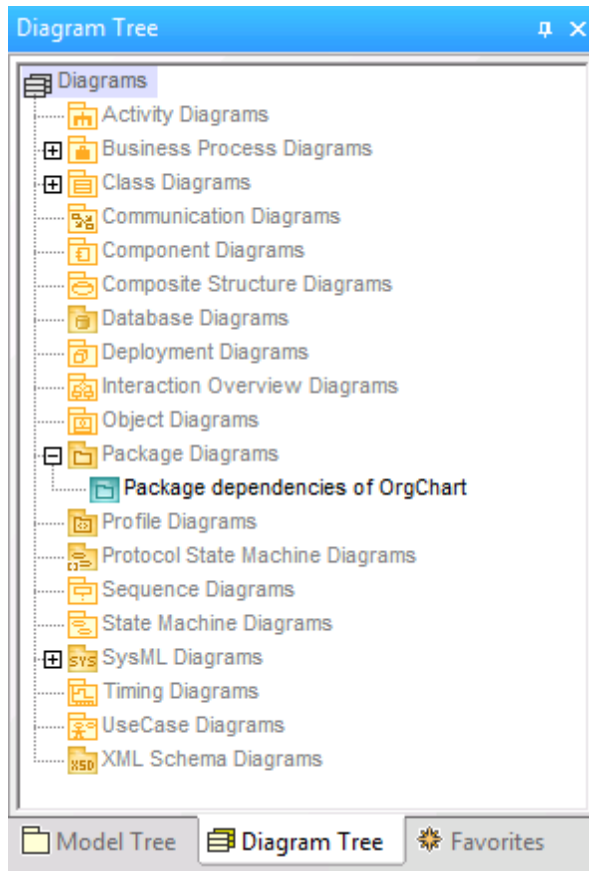


Diagram Tree

10.2.7 Profile Diagram and stereotypes

Altova web site:  [UML profile diagrams](http://www.altova.com/uml/profile-diagrams)

The UModel Profile diagram allows you to define custom stereotypes, tagged values and constraints in a dedicated diagram.

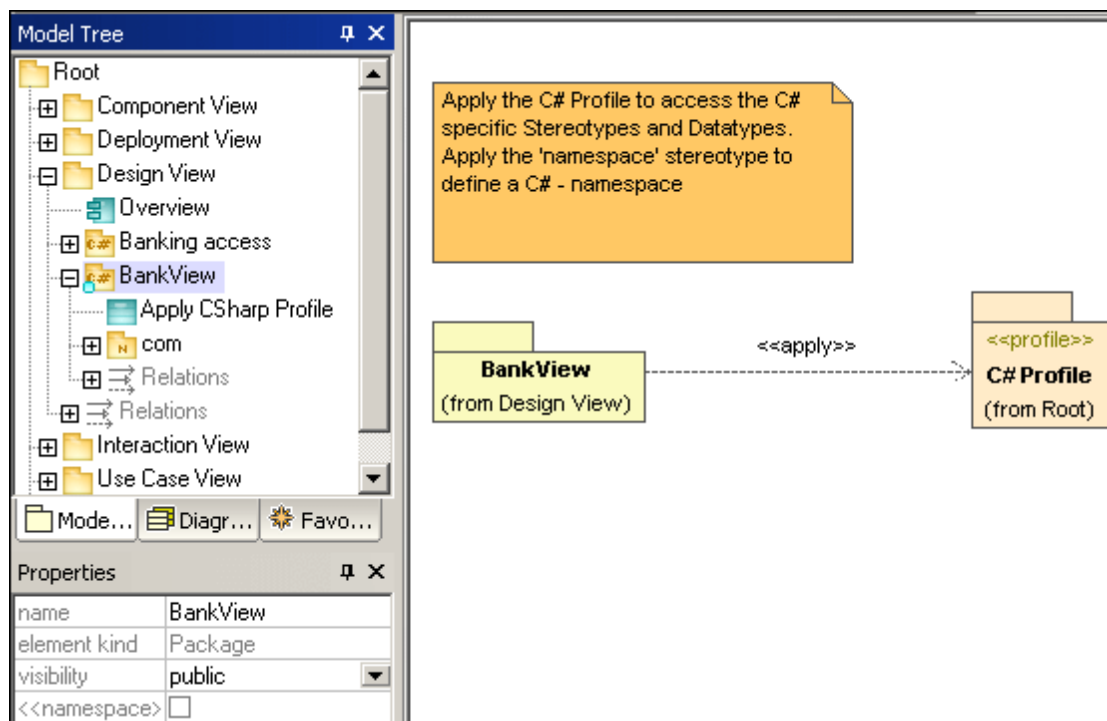
Profiles and stereotypes are used to extend the UML meta model. The primary extension construct is the Stereotype, which is itself part of the profile. Profiles must always be related to a reference meta model such as UML, they cannot exist on their own. UModel now supports the Profile Diagram in which you can define your own stereotypes.

The Java Profile.ump (C# Profile.ump, or VB Profile.ump) file need to be applied when creating new UModel projects using the menu item **Project | Include Subproject**. **This profile supplies the Java datatypes and stereotypes, and is essential when creating code for round-trip engineering.**

The following sections describe:

- [Adding Stereotypes and defining tagged values](#)
- [Stereotypes and enumerations](#)
- [User-defined stereotype styles](#)

The **Bank_CSharp.ump** sample file (in the ...**UModelExamples** folder) shows how this is done. The C# profile has been applied to the BankView package.



- Profiles are specific types of packages, that are applied to other packages.
- Stereotypes are specific metaclasses, that extend standard classes.
- "Tagged values" are values of stereotype attributes.

A ProfileApplication shows which profiles have been applied to a package, and is a type of package import that states that a Profile is applied to a Package. The Profile extends the

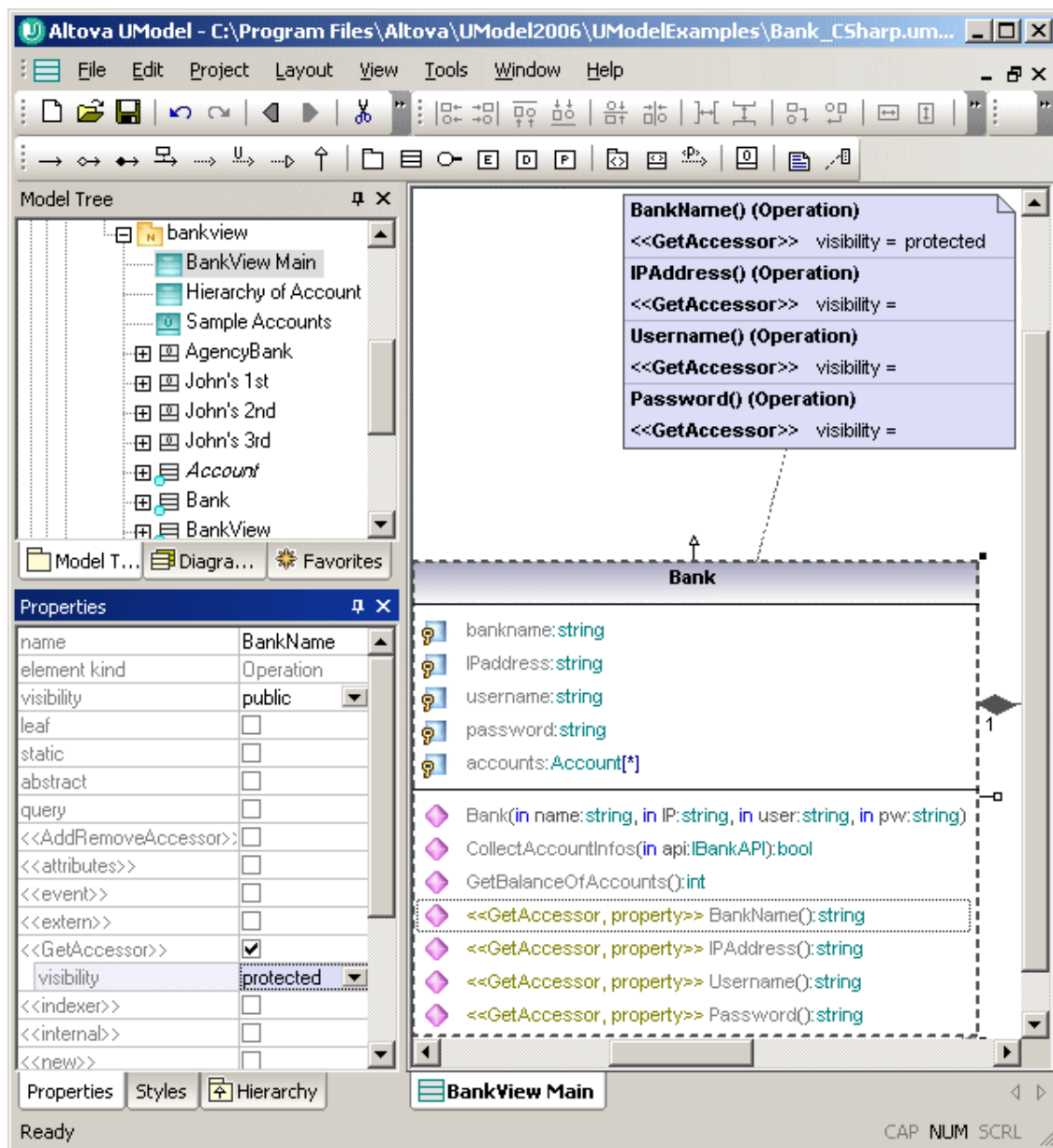
package it has been applied to. Applying a profile, using the ProfileApplication icon , means that all stereotypes that are part of it, are also available to the package.

ProfileApplications are shown as dashed arrows from the package to the applied profile, along with the <<apply>> keyword.

Stereotypes:

A stereotype defines how an existing metaclass may be extended. It is a kind of class that extends Classes through Extensions. Stereotypes can only be created in Profiles. Stereotypes are displayed as classes, in class diagrams, with the addition of the keyword <<stereotype>> added above the name of the class.

- Stereotypes may have **properties**, which are called "**tag definitions**"
- When the stereotype is applied to a model element, the property **values** are called "**tagged values**"
- When stereotypes containing properties are applied, the tagged values are automatically displayed in a comment element (shown below). Please see [Tagged values](#) for more info on how to customize the tagged values view
- Stereotypes have their own [style family](#)
- If the attribute is of type "enumeration", then a pop-up menu allows you to select from the predefined values. You can also enter/select the specific value in the Properties tab e.g. <<GetAccessor>> visibility = public, protected etc.

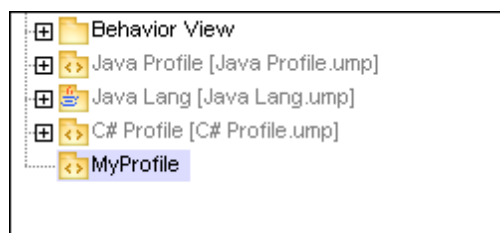


Adding Stereotypes and defining tagged values

This section uses the Bank_MultiLanguage.ump file available in the ...\\UModelExamples folder.

Creating a Profile diagram and a stereotype

1. Create a new profile in the Model Tree view, e.g. right click the Root package and select **New element | Profile** and name it "MyProfile".



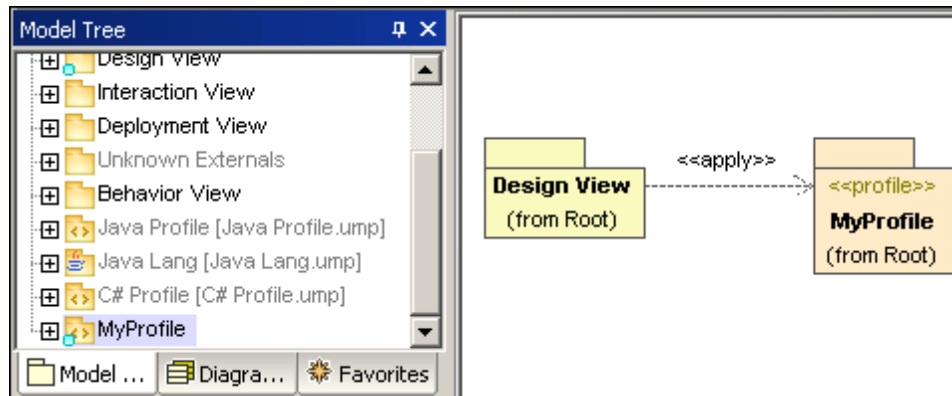
- Right click the **MyProfile** package in the Model Tree and select **New Diagram | Profile diagram**.

This adds the Profile diagram to the selected package.

- Drag the newly created profile "MyProfile", from the Model Tree into the new Profile diagram.

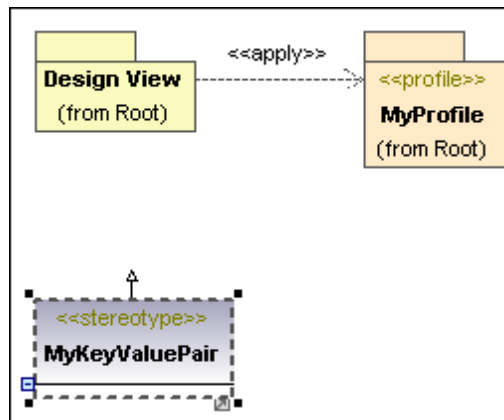
- Drag the **DesignView** package into the Profile diagram as well.

- Click the ProfileApplication icon  in the icon bar, select the DesignView package and drag the connector onto the MyProfile package.

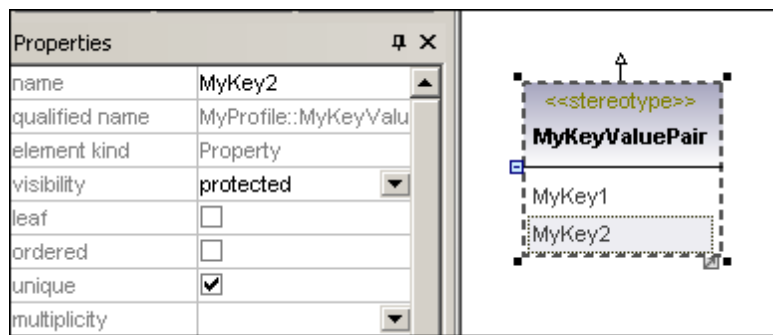


This allows the stereotypes defined in this profile (MyProfile) to be used in the DesignView package, or any of its subpackages.

- Click the stereotype icon  in the icon bar and insert a stereotype "class", e.g. MyKeyValuePair.



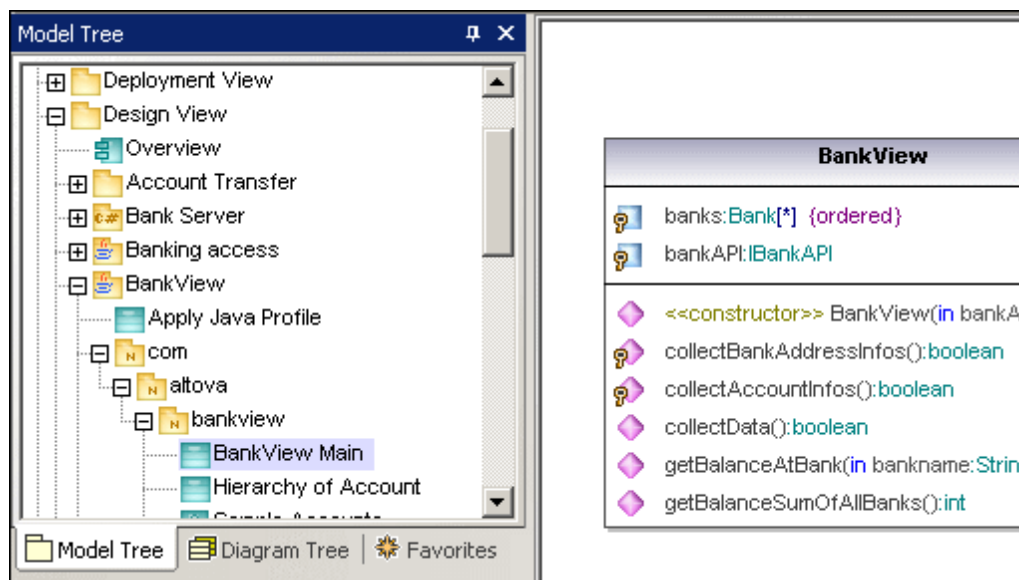
- Press F7 to add an attribute to the stereotype e.g. **MyKey1**. Do the same thing to add **MyKey2**.



This concludes the **definition** of the stereotype for the moment. We can now use/assign the stereotype when adding an attribute to a class which is part of the BankView package.

Using / assigning stereotypes

1. Double click the **BankView Main** class diagram icon in the Model Tree.



This opens the class diagram and displays the associations between the various classes. We now want to add an attribute to the BankView class, and assign/use the previously defined stereotype.

2. Click the **BankView** class and press F7 to add an attribute.
3. Use the scroll bar of the Properties tab to scroll to the bottom of the list. Notice that the **MyKeyValuePair** stereotype is available in the list box.

Properties	
derivedUnion	☐
default	...
aggregation	none
memberEndKind	n/a
<<annotationTypeElement>>	☐
<<annotations>>	☐
<<final>>	☐
<<transient>>	☐
<<volatile>>	☐
<<MyKeyValuePair>>	☒

- Click the **MyKeyValuePair** check box to activate/apply it. The two tagged values **MyKey1** and **MyKey2**, are now shown under the Stereotype entry.
- Double click in the respective fields and enter some values.

Properties	
<<annotations>>	☐
<<annotationTypeElement>>	☐
<<final>>	☐
<<MyKeyValuePair>>	☒
MyKey1	20
MyKey2	30
<<transient>>	☐
<<volatile>>	☐

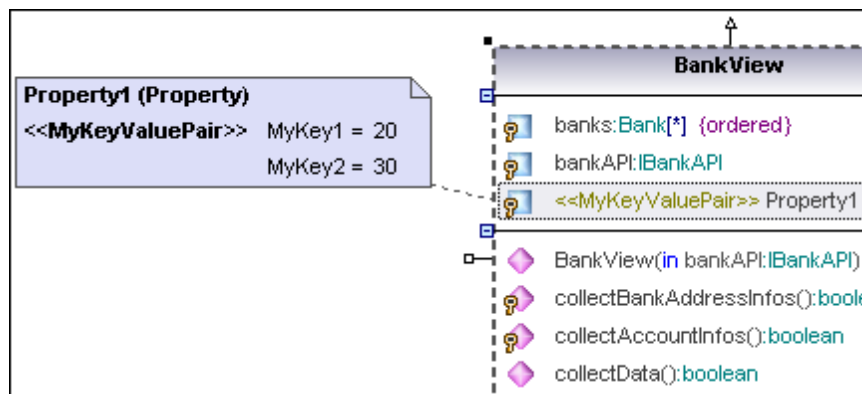
BankView	
banks:Bank[*] {ordered}	
bankAPI:IBankAPI	
<<MyKeyValuePair>> Property1	
BankView(in bankAPI:IBankAPI)	
collectBankAddressInfos():boolean	
collectAccountInfos():boolean	
collectData():boolean	

Displaying tagged values in a diagram

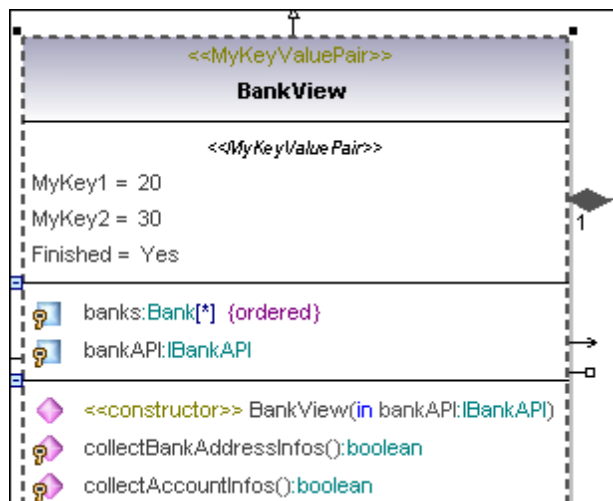
- Click the **Styles** tab, scroll down to the **Show Tagged Values** entry and select **all**.

Show Parameter	true	
Show Par.direction	true	
Show ExtensionPoints	true	
Show Tagged Values	all	
Show Execution Specific	true	
Show Message Number	nested	
Show Assoc. Ownershi	true	

The diagram tab now displays the tagged values in the Note element. Double clicking a value in the note element allows you to edit it directly.



Show tagged Values: **in compartment**



Displays the tagged values in a separate class **compartment**.

Please note:

When hiding attributes or operations using the ["Show / Hide node content"](#) context menu option, tagged values are also shown/hidden together with the modeling element.

Association (member) ends can display stereotypes by setting the **Show MemberEnd stereotypes** option to "true" in the Styles tab.

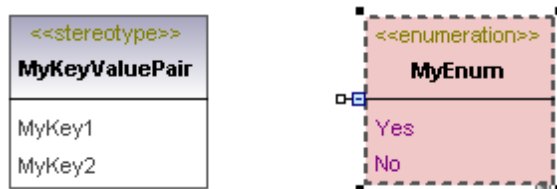
For more information on displaying tagged values see [Displaying tagged values](#).

Stereotypes and enumerations

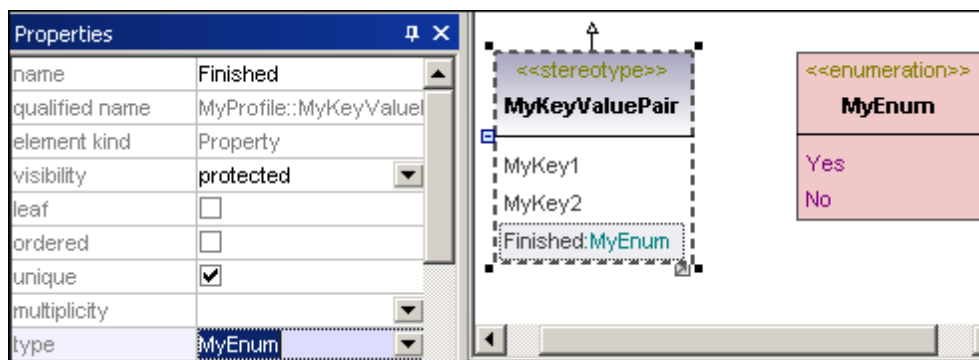
UModel has an efficient method of selecting enumerated values of stereotypes.

Click the diagram tab containing the stereotype definition (i.e. the [previously added](#) Profile diagram):

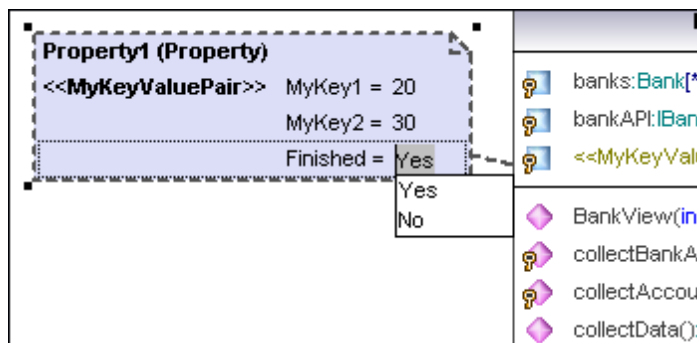
1. Click the Enumeration icon  in the icon bar to insert an enumeration into the diagram (containing the previously defined stereotype e.g. ProfileDiagram1).
2. Add **Enumeration Literals** to the enumeration by pressing SHIFT+F7, or use the context menu, e.g. the Enumeration Literals - **Yes**, and **No**.



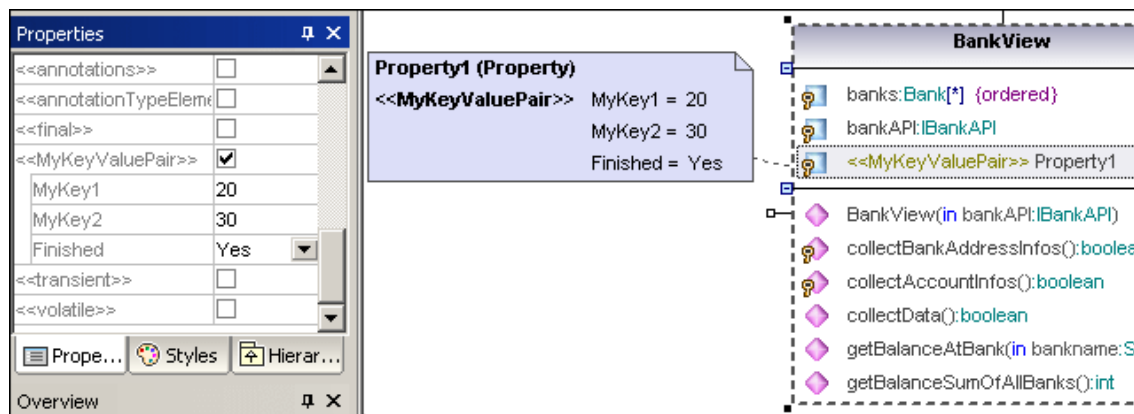
- Click the stereotype "class" and press F7 to add a new attribute/property, e.g. **Finished**.
- Select the **"MyEnum"** type from the Properties tab.



- Switch back to the **BankView Main** class diagram.
- The Property Finished, is now shown as a tagged value in the note element.

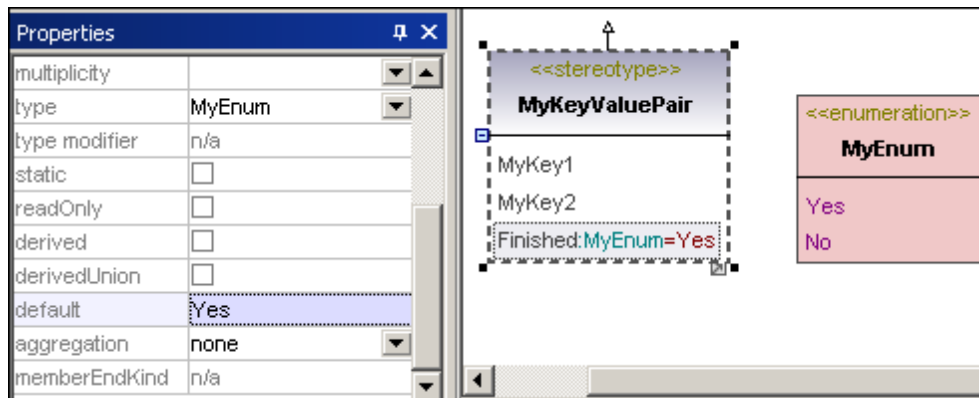


Double-clicking the Finished tagged value, presents the predefined enumeration values in a pop-up list. Click one of the enumerations to select it.



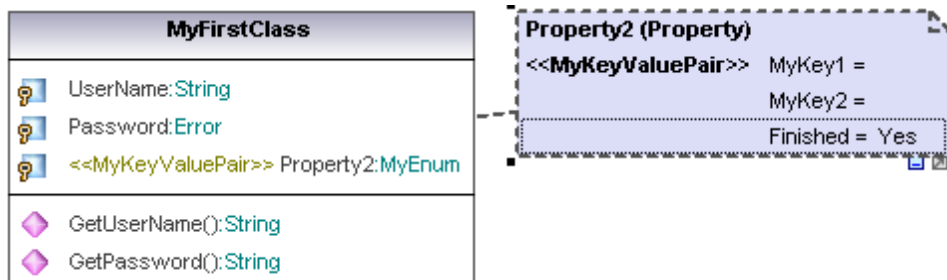
Enumeration default values

UModel allows you to define **default** tagged values. When adding an attribute to the stereotype, double click in the default field and enter one of the existing enumerations as the default value.



In this case, the default value "Yes" is entered.

When a property is added to a class, and the MyEnum type is selected, the default value is automatically inserted as the tagged value i.e. Finished = Yes.

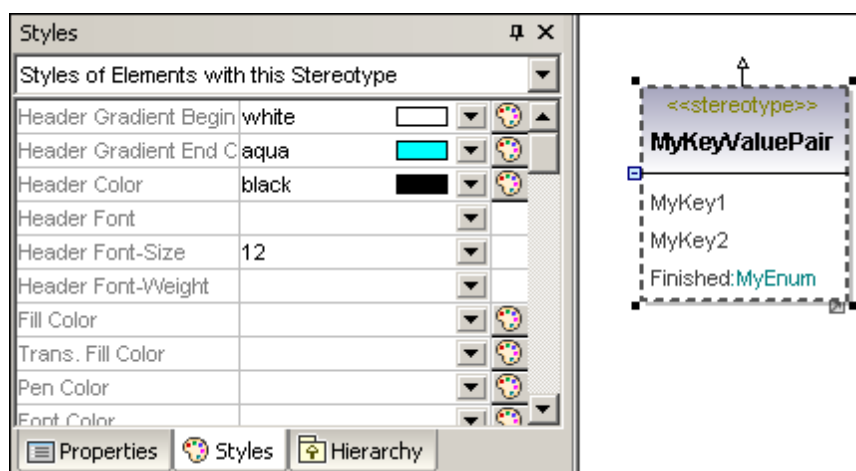


User-defined stereotype styles

It is now possible to create user-defined styles for individual stereotypes. This means that you can have specific fonts, colors etc. that are applied to those classes which are of that type of stereotype.

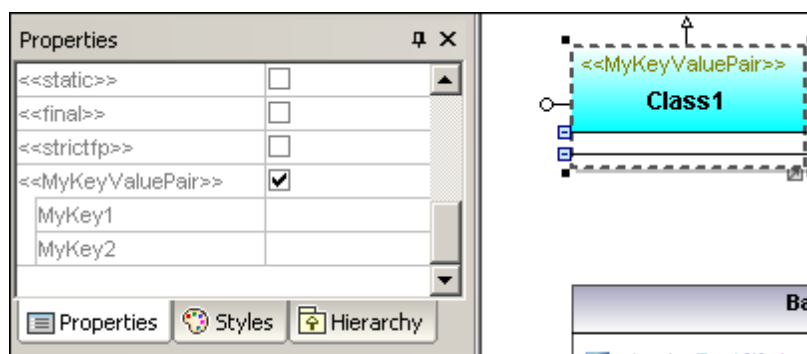
To create user-defined stereotype styles:

1. Click a previously defined stereotype e.g. [MyKeyValuePair](#) in the Class diagram.
2. Activate the Styles tab, then select **Styles of elements with this Stereotype** from the combo box.
3. Define the styles of this stereotype using the options in the Styles tab, e.g. Header Gradient End Color = aqua.



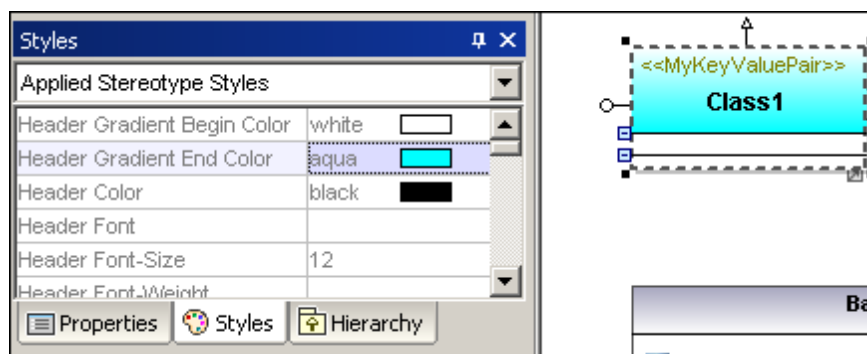
Clicking the stereotype class automatically displays the stereotype styles in the Styles tab.

4. Switch to a different Class Diagram and insert a new class.
5. Click the Class Header/Title and activate the MyKeyValuePair stereotype check box.



The new class now has the styles that were assigned to the stereotype i.e. an aqua gradient. Note that the stereotype styles are **not** applied if the stereotype is applied to an **property** or **operation** within the class.

6. Click the new stereotype class in the diagram then click the Styles tab.
7. Select the "**Applied Stereotype Styles**" entry in the Styles combo box.



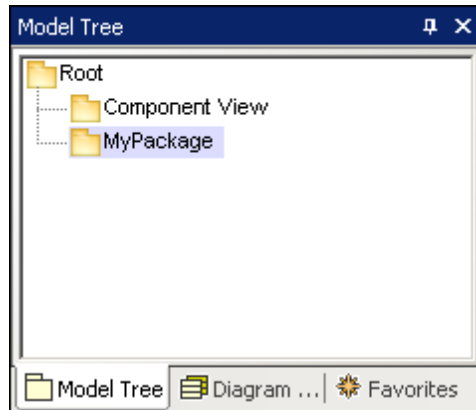
You can now preview the style settings defined for this stereotype in the Styles window. Note that you cannot change the style settings here. This must be done in the class diagram in which the stereotype was defined.

Custom stereotype icons - assigning

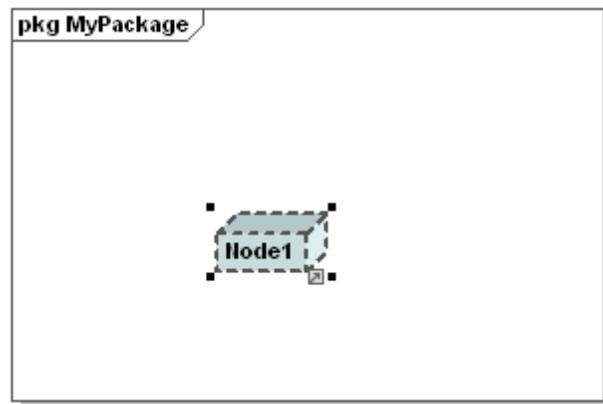
UModel allows you to create custom stereotype icons for custom stereotypes.

To create a custom stereotype icon:

1. Right click the Root package and select New element | Package.
2. Name the package MyPackage.

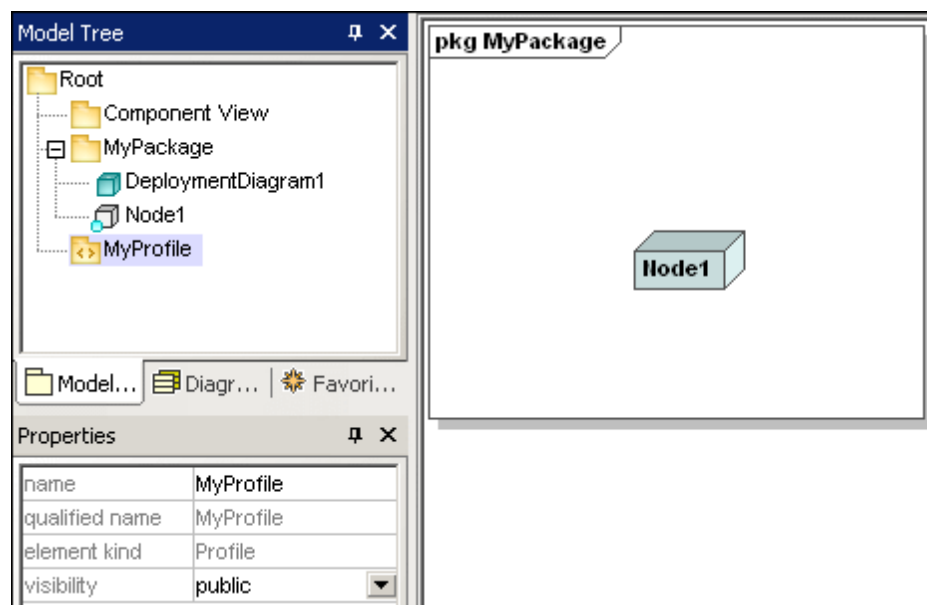


3. Right click MyPackage and select New Diagram | Deployment diagram.
This creates a deployment diagram in MyPackage.
4. Right click in MyPackage and select New | Node.



This adds the node object to the diagram.

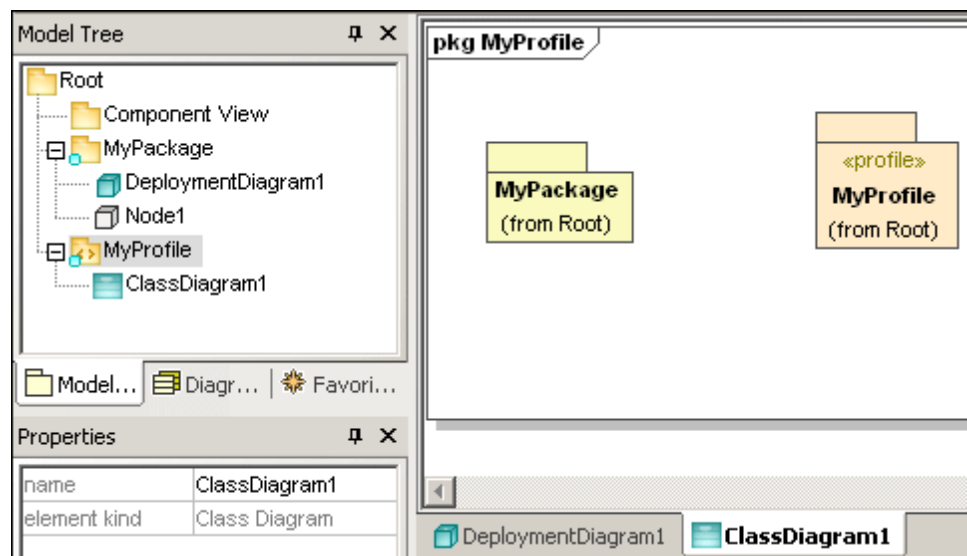
5. In the Model Tree, right click the Root package and select New element | Profile.
6. Name the profile "MyProfile".



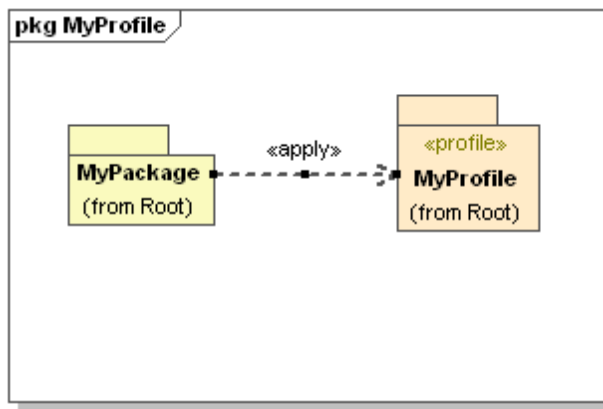
7. Right click the **MyProfile** package and select **New Diagram | Class diagram**.

Selecting the stereotype icon

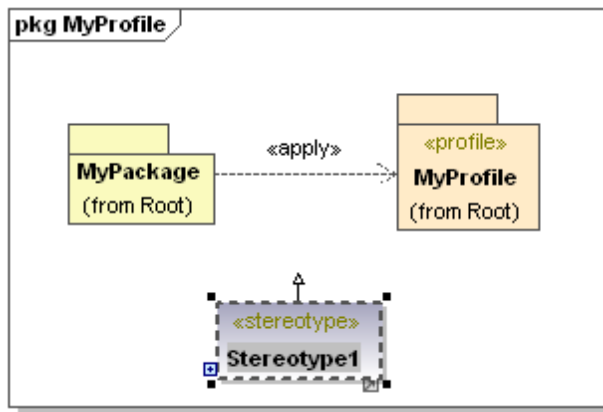
1. Drag the **MyPackage** and **MyProfile** folders from the **Model Tree** into the **Class diagram**.



2. Click the Profile-Application  icon in the tool bar, and drag the connector from **MyPackage** to **MyProfile**.

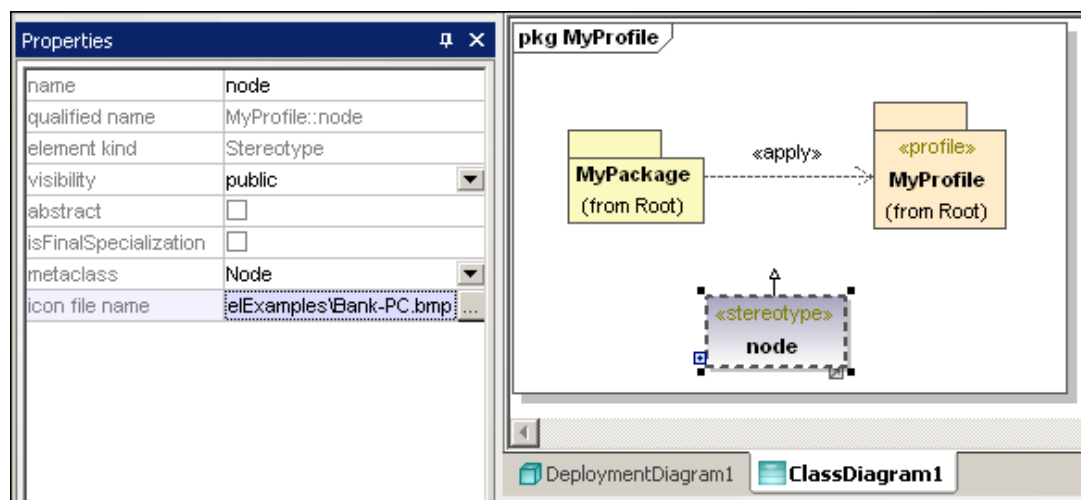


3. Click the stereotype icon  in the toolbar and click in the MyProfile package to insert it.

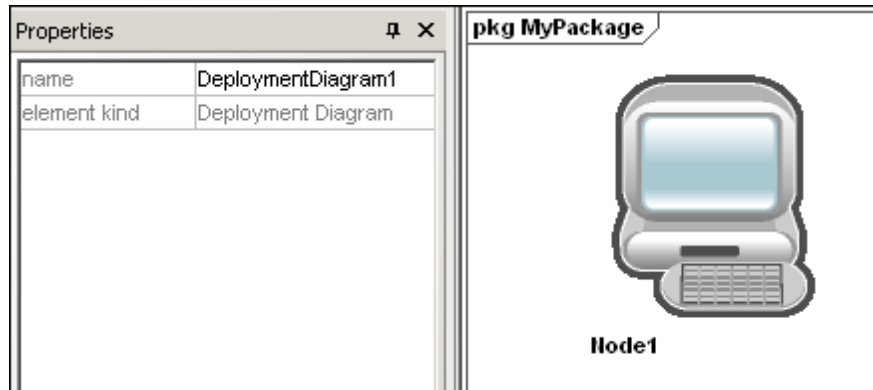


While the stereotype element is active, make the following changes in the Properties tab:

4. Change the name of the element from Stereotype1 to "node".
5. Change the metaclass property value from Element to "Node".
6. Enter a path for the image that you want to use as the stereotype icon in the "icon file name" field, or click the Browse button to select a path, e.g. c:\Documents and Settings \Me\My Documents\Altova\UModel2011\UModelExamples\Bank-PC.bmp.



7. Switch back to the Deployment diagram by clicking the DeploymentDiagram1 tab.
8. Click the Node1 element in the diagram.
9. Using the Properties window, activate <node> check box.



The representation of the stereotype icon has now changed to that of the selected bitmap.

Please note:

When displaying the custom icons the RGB color 82,82,82 is the background transparent color.

10.3 Additional Diagrams

UModel now supports the import and generation of W3C XML schemas as well as their forward and reverse-engineering in the code-engineering process.

UModel now supports the Business Processing Modeling Notation Standard version 1.0 which was adopted as an OMG standard in February 2006.



[XML Schema](#)



[Business Process Modeling Notation](#)



[SysML diagrams](#)



[Database diagrams](#)

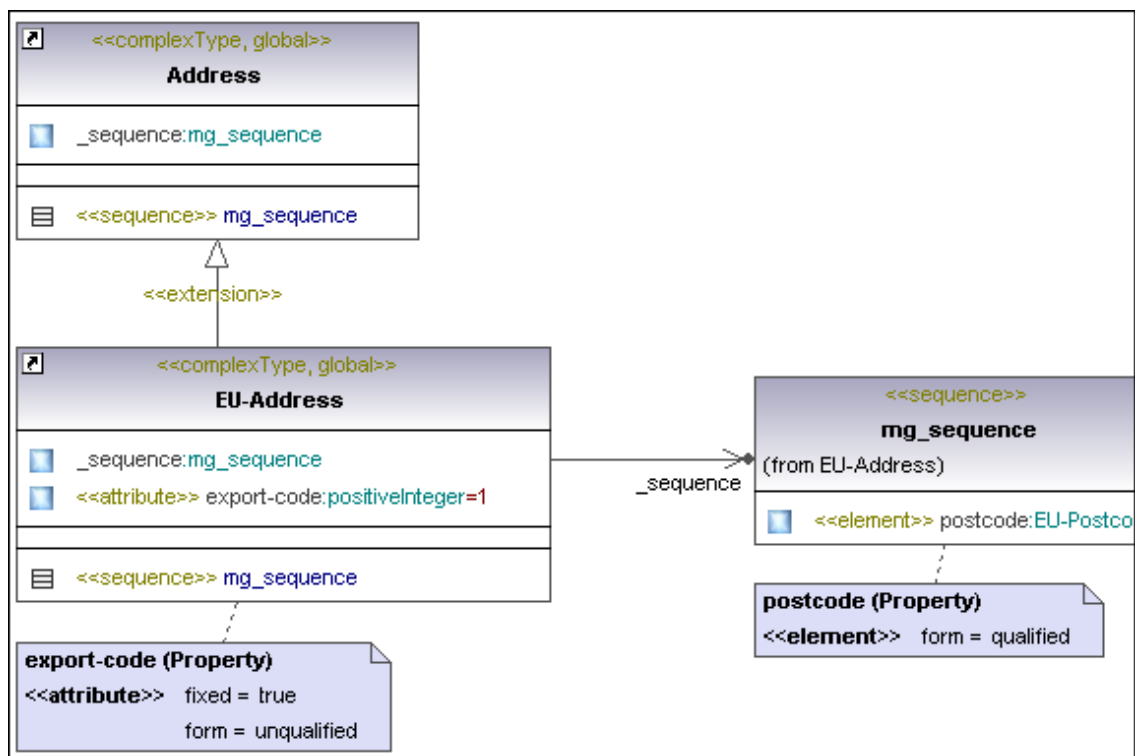
10.3.1 XML Schema Diagrams

Altova web site:  [XML Schemas in UML](#)

XML Schema diagrams display schema components in UML notation. Global elements i.e. elements, simpleTypes, complexTypes are shown as classes, or datatypes, with attributes in the attributes compartment.

There are no operations in the Operation compartment. The Tagged Value note modeling element is used to display the schema details.

To see how the UML elements and XML schema elements/attributes are mapped, navigate to [XML Schema to/from UModel elements](#).



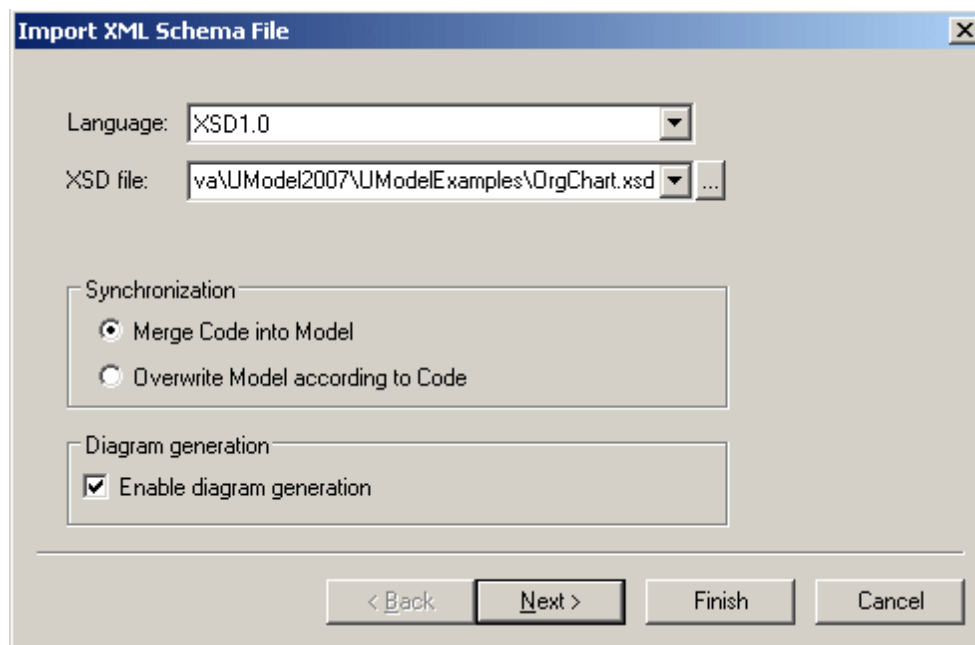
Please note:

Invalid XML Schemas cannot be imported into UModel. XML Schemas are not validated when importing, or creating them in UModel. XML Schemas are also not taken into account during the project syntax check. A well-formed check is however performed when importing an XML schema.

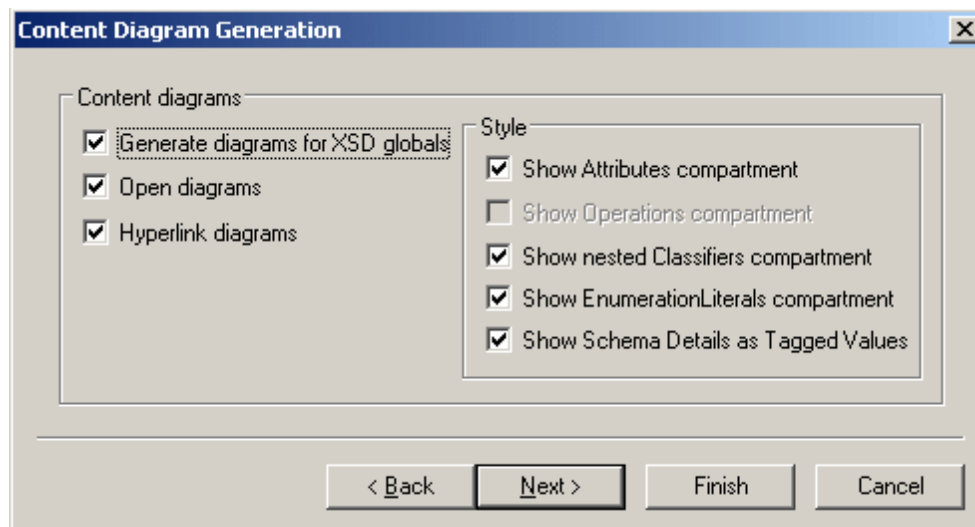
Importing XML Schema(s)

To import a single XML Schema:

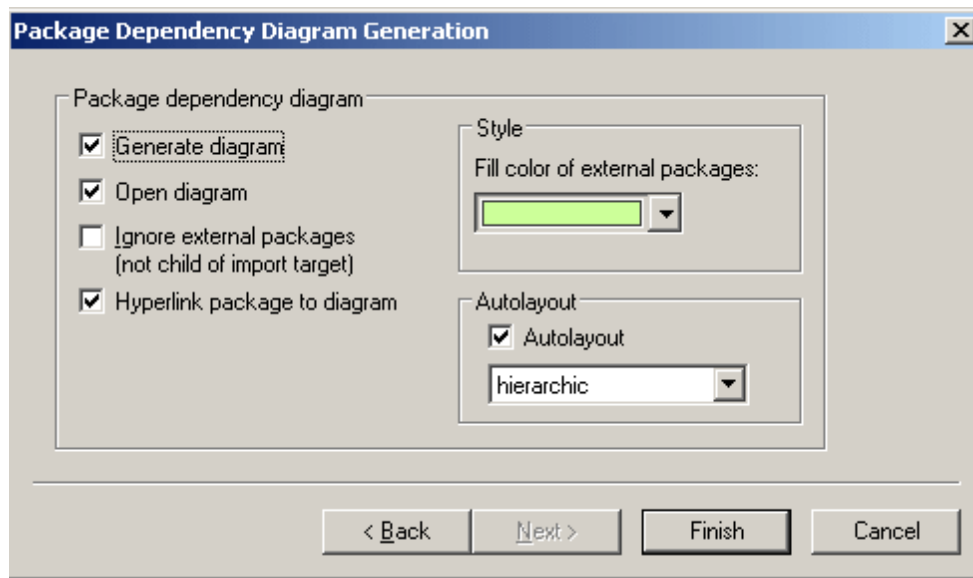
1. Select the menu option **Project | Import XML Schema file**.



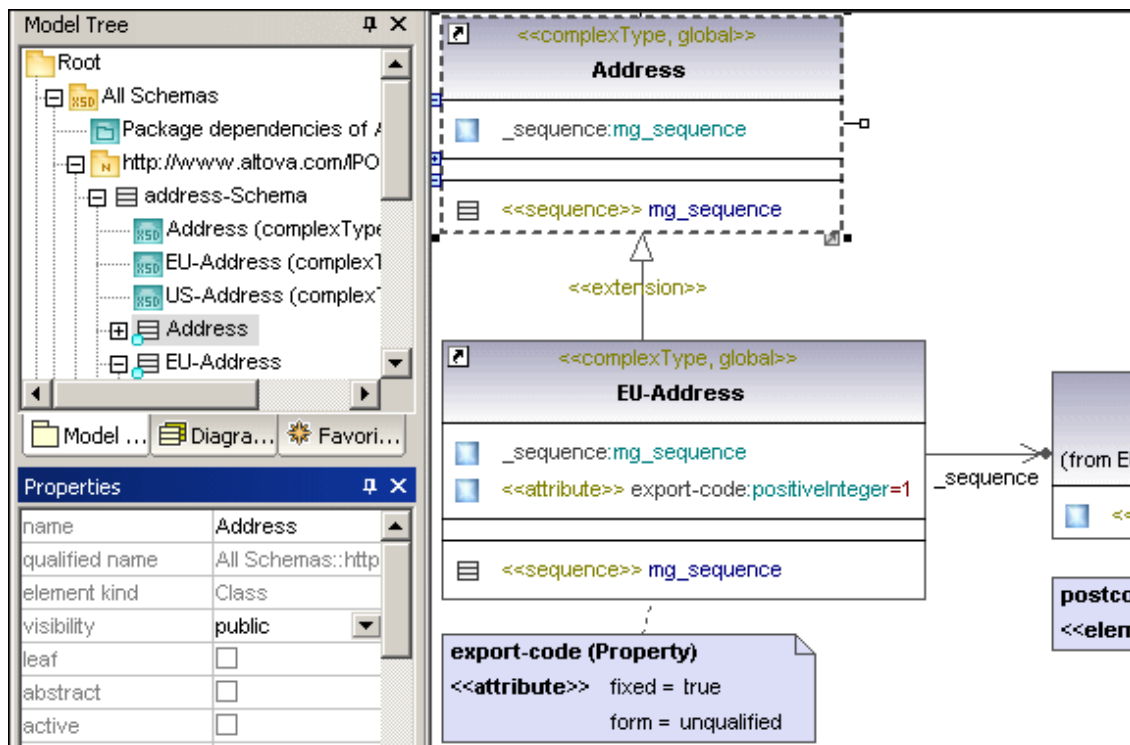
2. Make sure that the **Enable diagram generation** check box is active and click Next, to continue.



3. Define the Content diagram options in the group of that name. The first option creates a separate diagram for each schema global element.
4. Select the compartments that are to appear in the class diagrams in the Style group. The "Show schema details as tagged values" option displays the schema details in the Tagged Value note modeling element.
5. Click Next to define the Package dependency diagram.

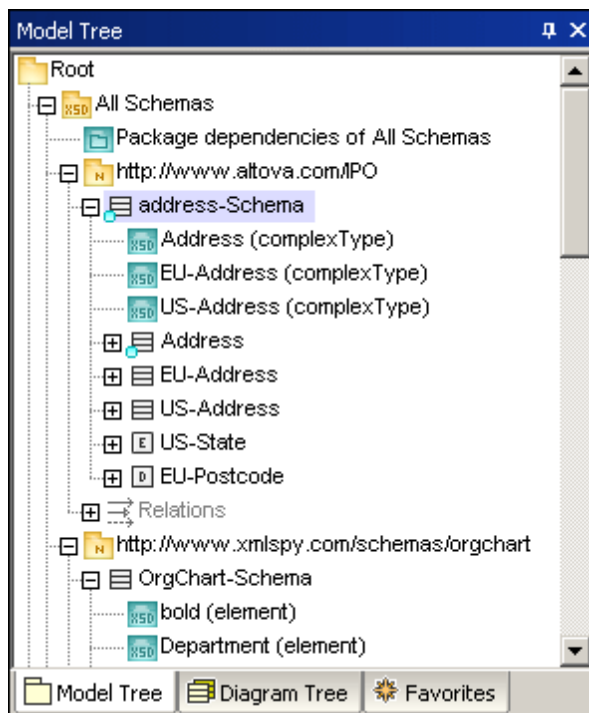


6. Click Finish to start the XML Schema import.
The schema(s) are imported into UModel and all diagrams are available as tabs. The screenshot below shows the content of the EU-Address (complexType) diagram.



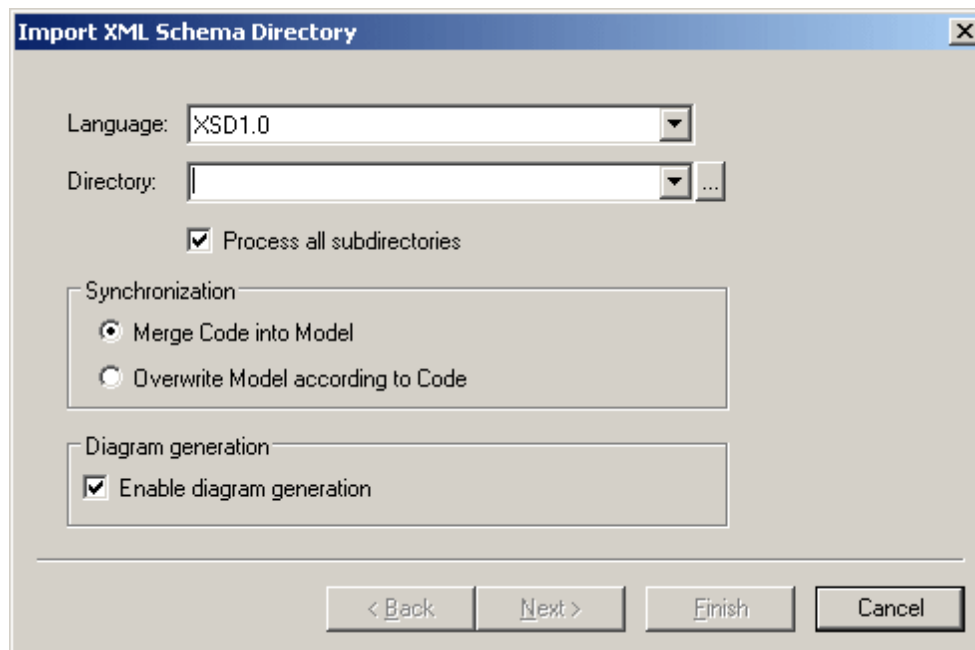
Please note:

A new package called All Schemas was created and set as the XSD Namespace Root. All XSD globals generate an XML Schema diagram, with the diagrams under the respective namespace packages.



To import multiple XML Schemas:

1. Select the menu option **Project | Import XML Schema directory**.



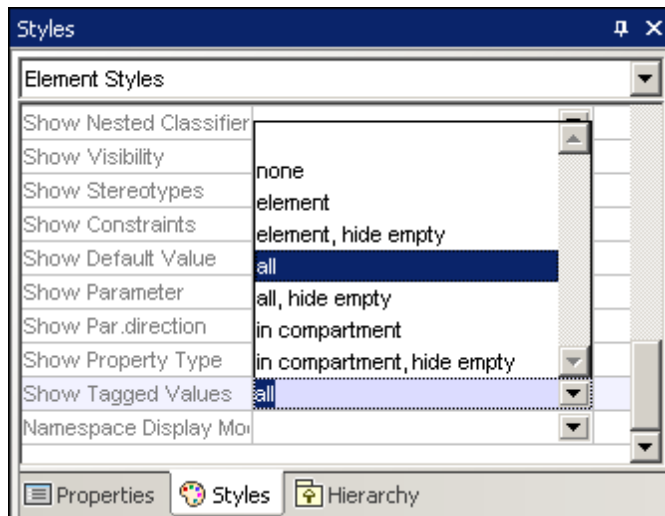
2. Activate the "Process all subdirectories" if you want to import Schemas from all subdirectories.
The rest of the import process follows the sequence of importing a single XML schema described above.

Please note:

If an XML schema includes or imports other schemas, then these schemas will be automatically imported as well.

Displaying tagged values - schema details

Schema details displayed as tagged values in the Tagged Value note element, can be configured using the **Show Tagged Values** in the Styles tab, or by clicking the "Toggle compact mode" icon at the bottom right of the Tagged Value **note**. This switches between the two states "all" and "all, hide empty", etc. which are shown below.

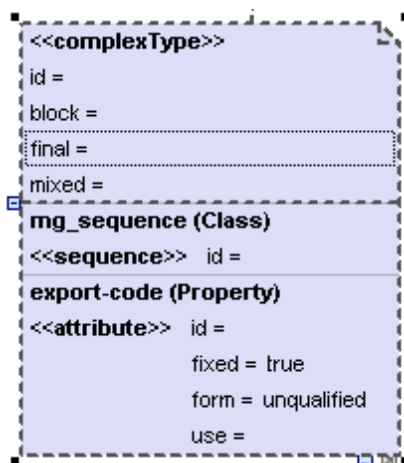


Note:

Selecting the tagged values from the **Styles** tab sets the tagged value settings project wide. Right clicking a **class** and selecting Tagged values | all etc., only affects the individually marked classes.

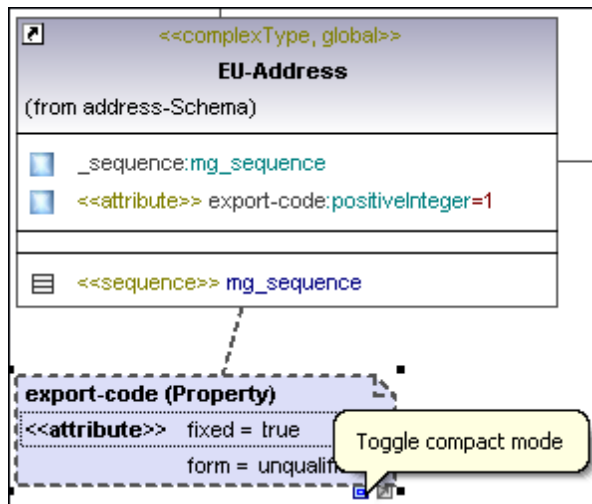
Show tagged values: **all**

Displays the tagged values of the class as well as those of the owned attributes, operations etc.



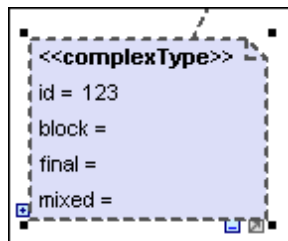
Show tagged values: **all, hide empty**

Displays only those tagged values where a value exists e.g. fixed=true.



Show tagged values: **element**

Displays the tagged values of the class but **not** those of the owned attributes, operations etc.

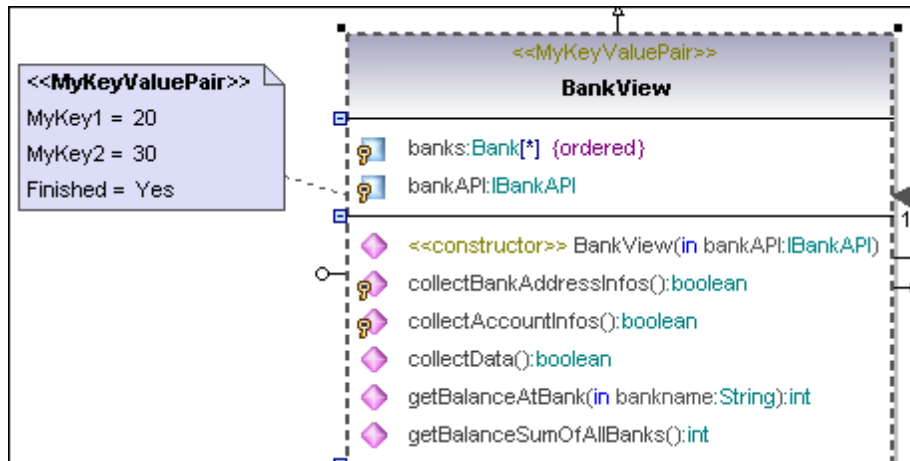


Show tagged values: **element, hide empty**

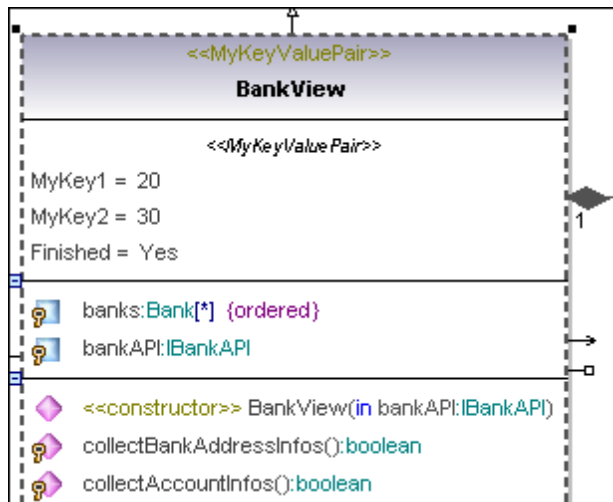
Displays only those tagged element values of a class, without the owned attributes, where a value exists e.g. `id=123`



Show tagged values: **in compartment**



Displays the tagged values in a separate class compartment.



Show tagged values: **in compartment, hide empty**

Displays only those tagged element values of a class, without the owned attributes, where a value exists.

XML Schema annotation:

When importing XML schemas, please note that only the first annotation of a complex- or simpleType is displayed in the Documentation window.

Inserting XML Schema elements



Using the toolbar icons:

1. Click the specific XML Schema diagram icon in the toolbar.
2. Click in the XML Schema diagram to insert the element.
Note that holding down **Ctrl** key and clicking in the diagram tab, allows you to insert multiple elements of the type you selected.

Dragging existing elements into the XML Schema diagram:

Elements occurring in other diagrams can be inserted into an existing XML Schema diagram.

1. Locate the element you want to insert in the Model Tree tab (you can use the search function text box, or press **Ctrl+F** to search for any element).
2. Drag the element(s) into the XML Schema diagram.

Note: you can also use the Copy and "Paste in diagram only" commands to insert elements.

Please note:

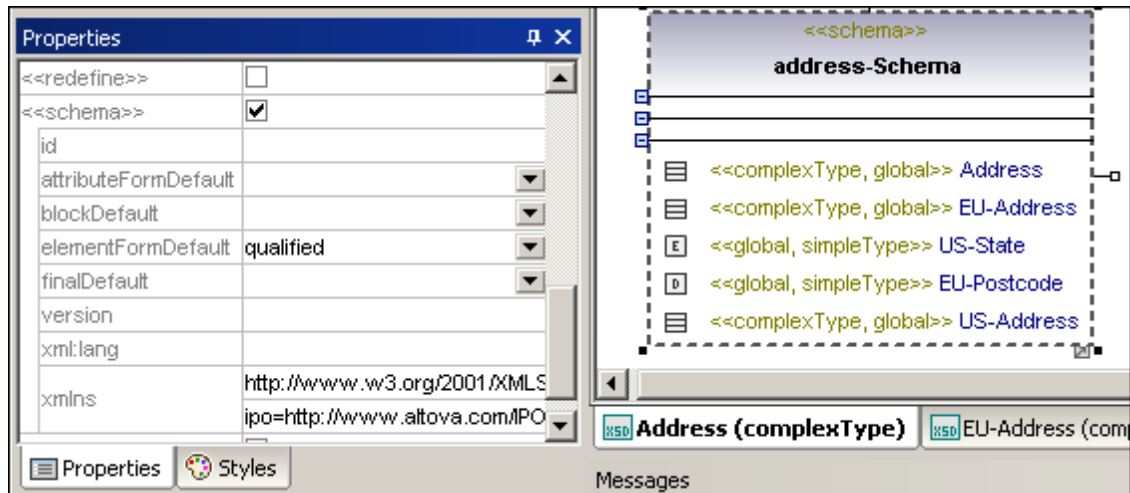
Either method described above, applies to any of the XML Schema elements that can be inserted onto the diagram.

**XSD Target Namespace**

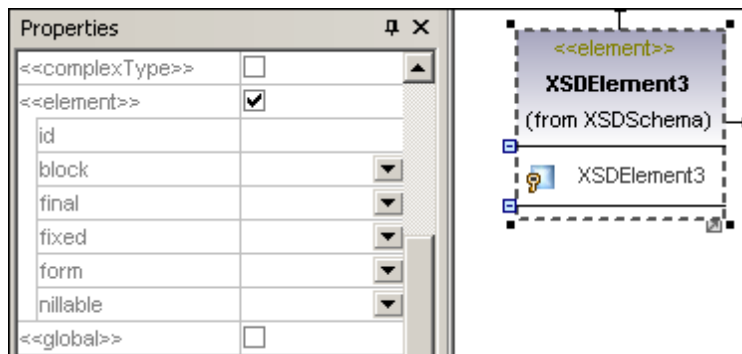
Inserts/defines the target namespace for the schema. The XSD Target Namespace must belong to an XSD Namespace Root package.

**XSD Schema**

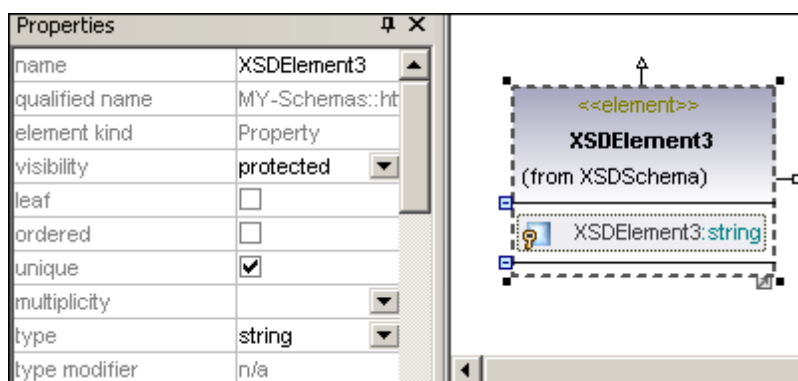
Inserts/defines an XML schema. The XSD schema must belong to an XSD Target Namespace package.

**Element (global)**

Inserts a global element into the diagram. Note that a property is also automatically generated in the attributes compartment.

**To define the property data type:**

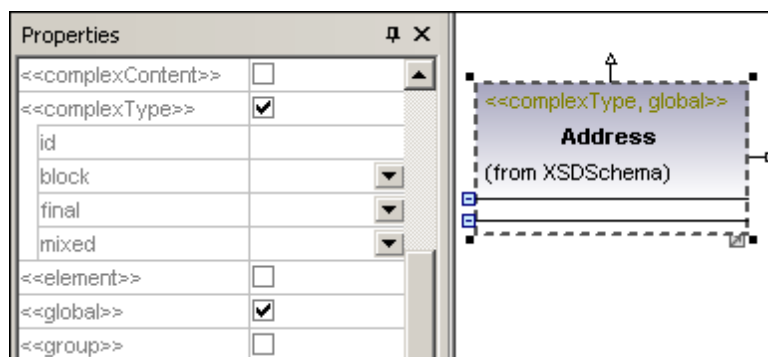
1. Double click the property and place the cursor at the end of the line.
2. Enter a colon character ":", and select the data type from the pop-up dialog box, e.g string.



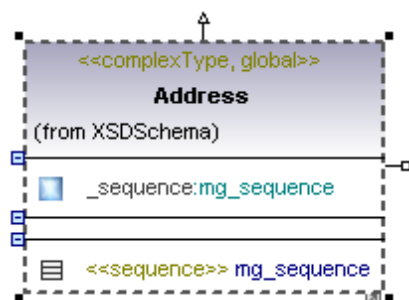
Creating a "content model" consisting of a complexType with mandatory elements:

This will entail inserting a complexType element, a sequence element/compositor, and three elements.

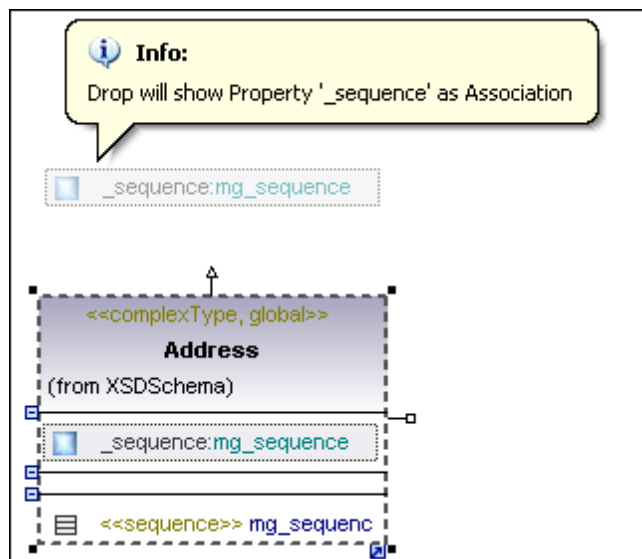
1. Click the XSD ComplexType icon , then click in the diagram to insert it.
2. Double click the name and change it to Address.



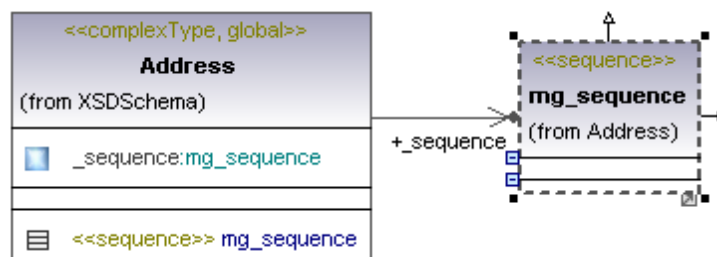
3. Right click Address and select **New | XSD Sequence**.



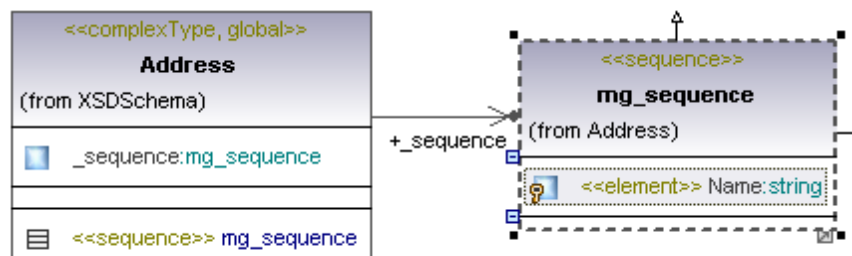
4. Click the **_sequence:mg_sequence** attribute in the attribute compartment, and drag it out into the diagram.



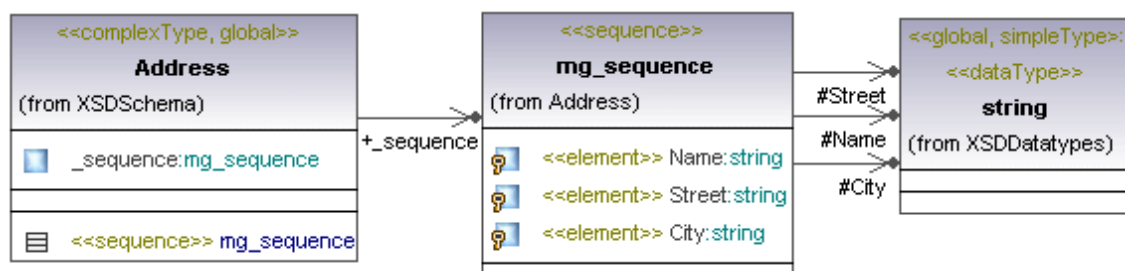
This creates a sequence class/compositor at the drop position.



5. Right click the sequence class and select **New | XSD Element (local)**. This adds a new property element.
6. Double click the property, enter the element name, e.g. Name, add a colon ":" and enter "string" as the data type.



7. Do the same for the two more elements naming them Street and City for example.
8. Click the Name property and drag it into the diagram.

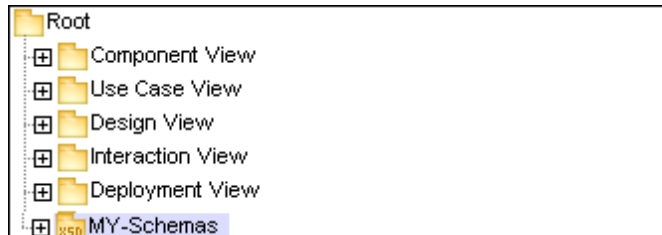


Creating and generating an XML Schema

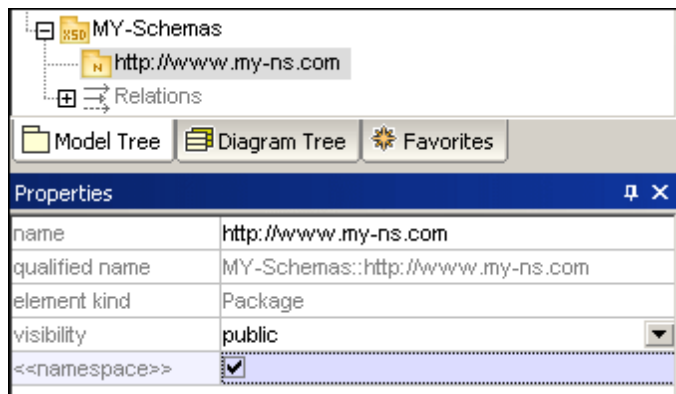
You would generally import a schema, edit it in UModel, and output the changes. It is however possible to generate a schema from scratch. This will only be described in broad detail however.

Creating a new schema in UModel:

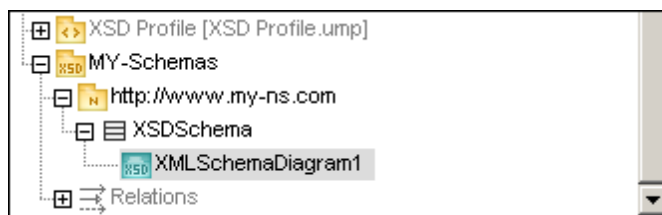
1. Create a new package in the Model Tree e.g. MY-Schemas.



2. Right click the new package and select the menu option **Code Engineering | Set as XSD namespace root**.
You are asked if you want to assign the XSD profile if this is the first XSD Namespace root in the project.
3. Click OK to assign the profile.
4. Right click the new package and select **New Element | Package**.
5. Double click in the package name field and change it to the namespace you want to use, e.g. <http://www.my-ns.com>.
6. Click the <<namespace>> check box in the Properties tab, to define this as the target namespace.



7. Right click the namespace package and select **New diagram | XML Schema diagram**.
You will be prompted if you want to add the Schema diagram to a new XSD Schema.
8. Click Yes to add the new diagram.

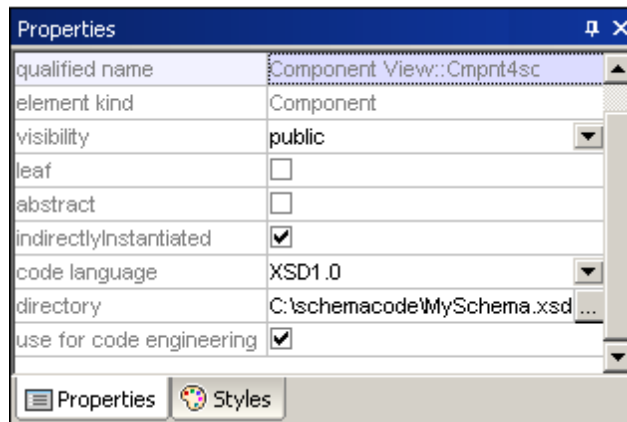


You can now create your schema using the icons in the XML Schema icon bar.

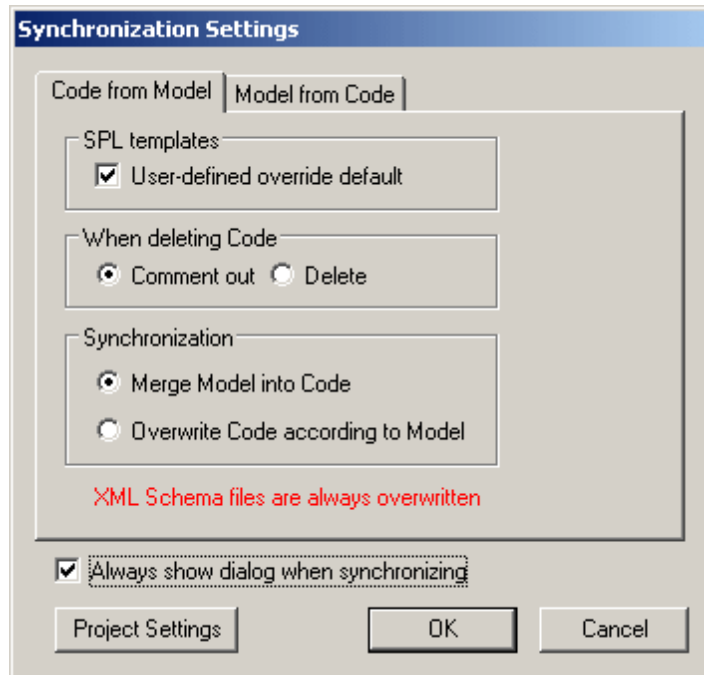
Generating the XML schema:

1. Drag the XSDSchema onto a component to create a Component Realization.

2. Make sure that you set the code language, of the component, to XSD1.0, and enter a path for the generated schema to be placed in, e.g. C:\schemacode\MySchema.xsd.



3. Select the menu option **Project | Overwrite Program Code from UModel project**, and click OK to generate the schema.



10.3.2 Business Process Modeling Notation 1.0 / 2.0

Altova web site:  [Business Process modeling in UModel](#)

BPMN is a standardized flow-chart notation which shows business processes as a workflow and is easily understandable by all involved in the business process. UModel now supports BPMN versions 1.0 and 2.0.

UModel adds the new BPMN 2.0 diagrams: [Choreography](#), [Collaboration](#), and BPMN 2.0 [Standard Business Process Diagrams](#).

Both BPMN 1.0 and BPMN 2.0 diagrams can coexist in the same UModel project. Conversion from BPMN 1.0 to BPMN 2.0 can be done at any time.

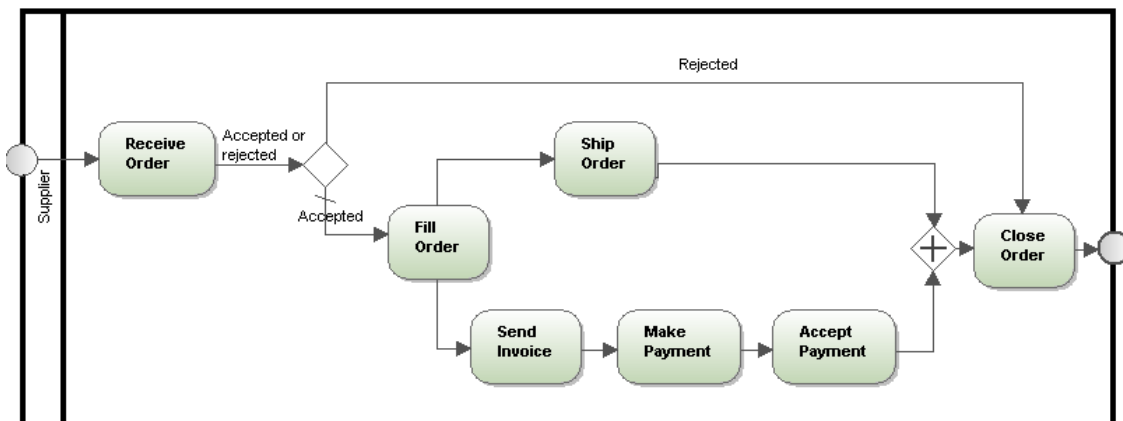
There are four basic element BPMN categories:

Flow objects	Events, Activities (Tasks or Sub-Processes), Gateways
Connecting objects	Sequence flow, Message Flow, Association
Swimlanes	Pool, Lane
Artifacts	Data Objects, Group, Text Annotation

Inserting BPMN diagrams and BPMN objects works in exactly the same way as inserting modeling elements in UModel.

Objects can be inserted using the icon bar; associations to other objects can be directly created by clicking on the object "handles" and dragging the connector to the target object. Properties can be viewed and set using the Properties tab.

Note that you can use the Layers tab to create multiple layers per BPMN diagram, please see [Layer](#) for more information.



To convert BPMN 1.0 diagrams to BPMN 2.0 diagrams:

1. Right click in a BPMN 1.0 diagram and select the option **Convert to BPMN 2.0 diagram**.

If more than one BPMN 1.0 diagram exists in the same package, you will be prompted to convert all of those in that package.

A second prompt appears, asking if you want to include the BPMN 2 Profile to the

project.

Clicking OK converts the diagrams.

Flow objects

Flow objects are the graphical elements that define the behaviour of a business process. There are three Flow Objects: Events, Activities and Gateways.

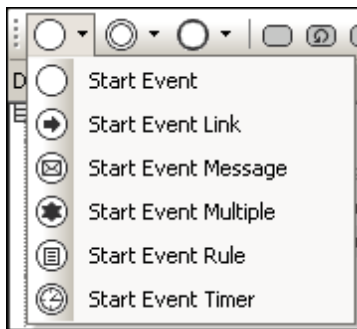
Events

An event is something that occurs during a business process and is represented by a circle. Events affect the flow of the process and generally have a cause (trigger) and a result. There are three different types of events: start, intermediate or end, where each group has its own drop-down combo box.

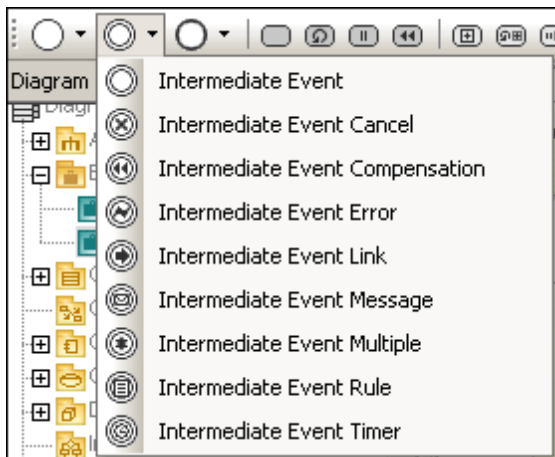
To insert an Event:

1. Click the combo box to open the drop-down list of the type of event you want insert.
2. Select the specific Event and click in the diagram tab to insert it.

Start Event

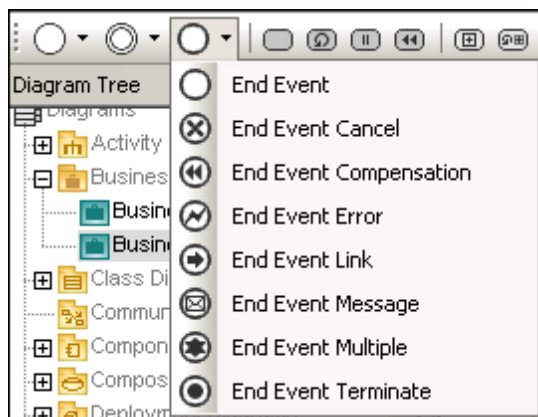


Intermediate Event



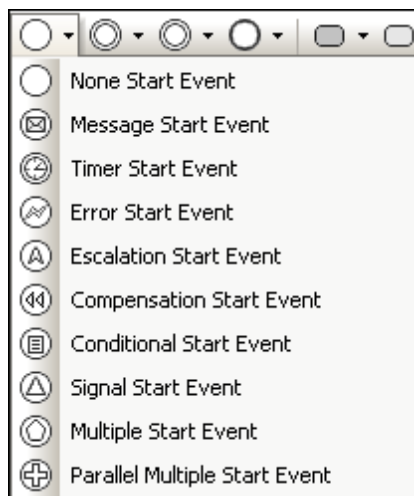
Intermediate events can be attached to the boundary of a Task or Sub-Process, and show that the activity is to be interrupted when the event is triggered.

End Event

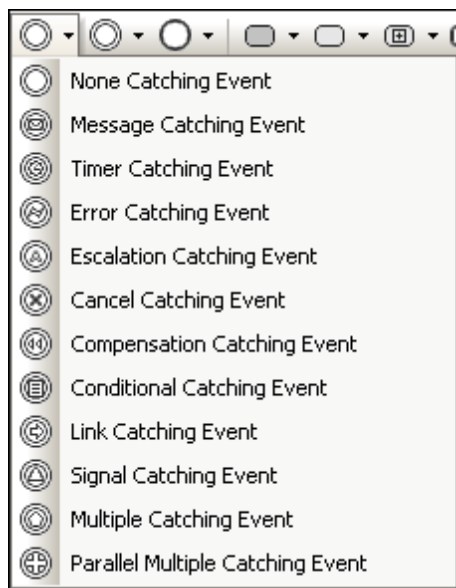


BMPN 2.0 Events:

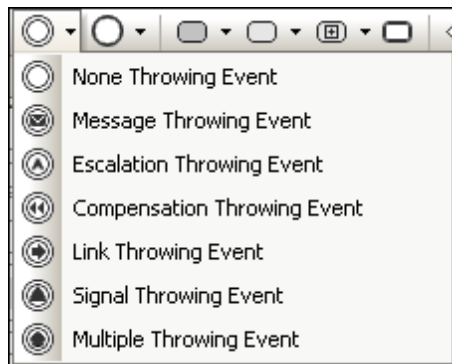
Start Events



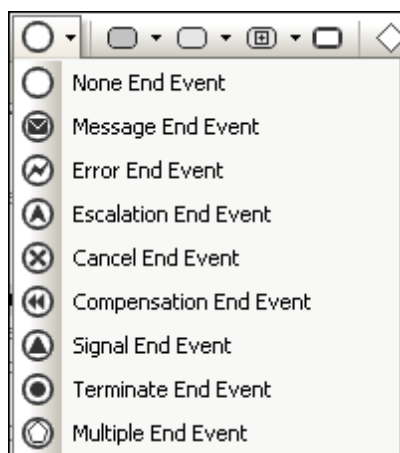
Catching Events



Throwing Events



End Events:



Activity

Activities are actions that are performed during a business process, and are represented by rounded rectangles. Process models can contain the following types of activity: Process, Sub-Process and Task. Activities can occur singly or multiple times within a loop.

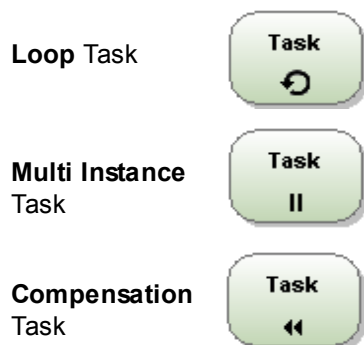


To insert an Activity:

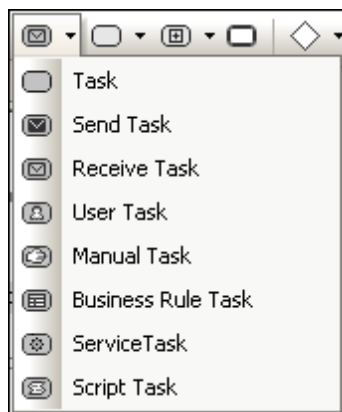
1. Click the specific Task or Sub-Process icon of the icon bar.
2. Click in the diagram tab.

Activity - Task

Tasks are activities that are included in a process. Tasks cannot be broken down into lower level subtasks, they are atomic.

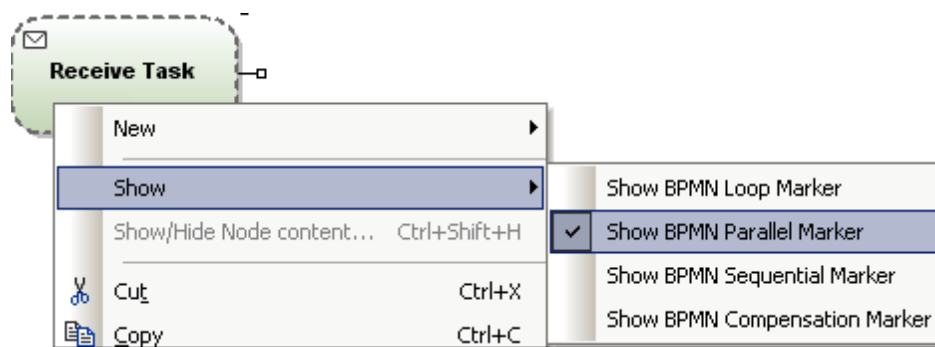


BPMN 2.0 Tasks



To define a Loop, Parallel, Sequential or Compensation marker:

1. Right click the inserted task and select the specific marker, e.g. **Show | Show BPMN Parallel Marker**.



Note: You can also define the Marker in the Properties tab under the "MultiInstanceLoopCharacteristics" entry.

Activity - Sub-Process

A Sub-Process is a compound activity that is included in a process, and allows hierarchical business process model development. A Sub-Process can be broken down into finer detail through various sub-activities.

A [collapsed Sub-Process](#) is displayed as a top-level element, where the details of the sub-process are not visible. A "plus" icon in the element shows that an additional layer of complexity exists.

An [expanded Sub-Process](#) displays the details of the Sub-Process within its boundaries. Note that a sequence flow cannot cross the boundary of a Sub-Process.

Gateway

Gateways are used to determine how Sequence Flows branch and merge within a process, and are always shown as a diamond.



Inclusive Gateway (OR)



Parallel Gateway (AND)



Data Based Exclusive Gateway (XOR)



Event Based Exclusive Gateway (XOR)



Complex Gateway (Decision/Merge)

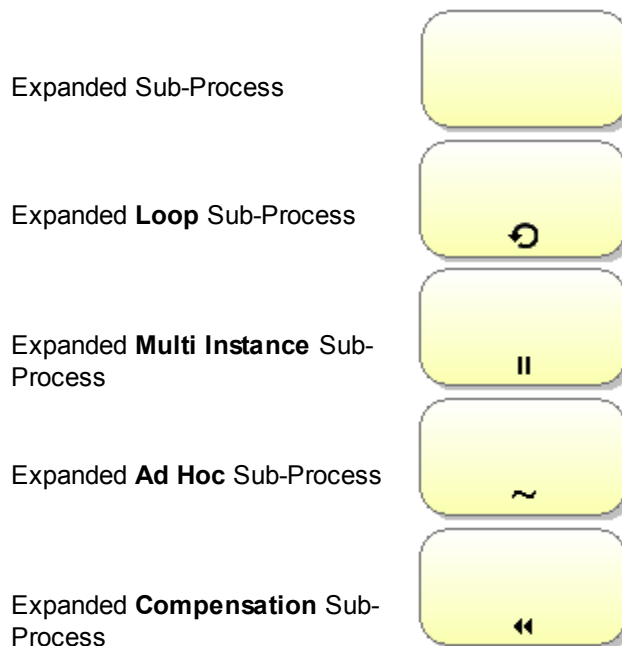


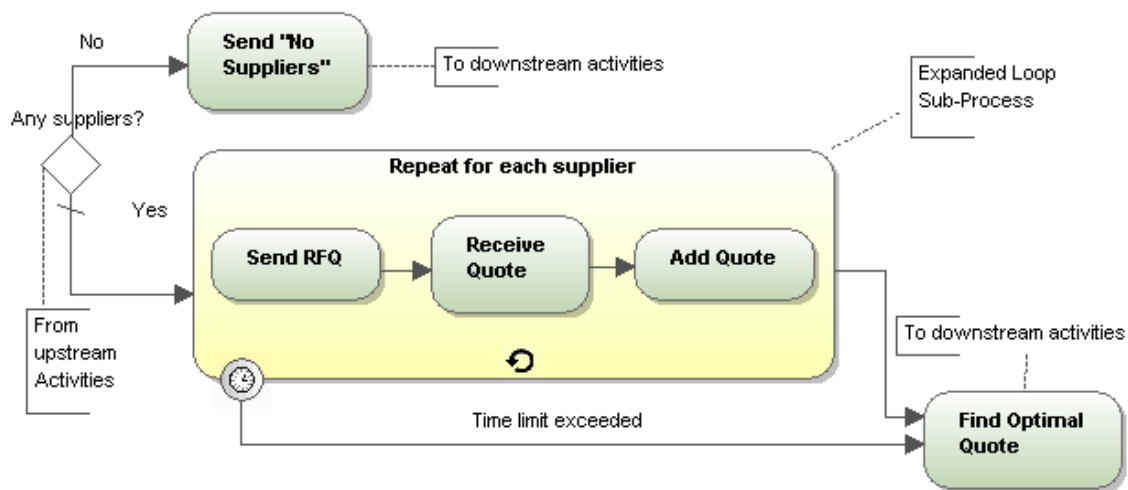
BPMN 2.0 Gateways:



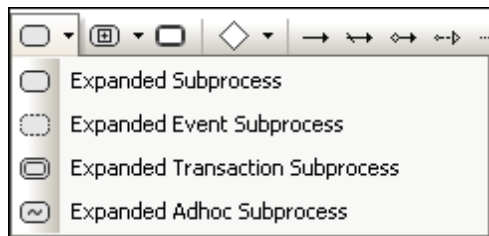
Expanded Sub Processes

Expanded versions of sub processes show the process detail within the element boundaries.



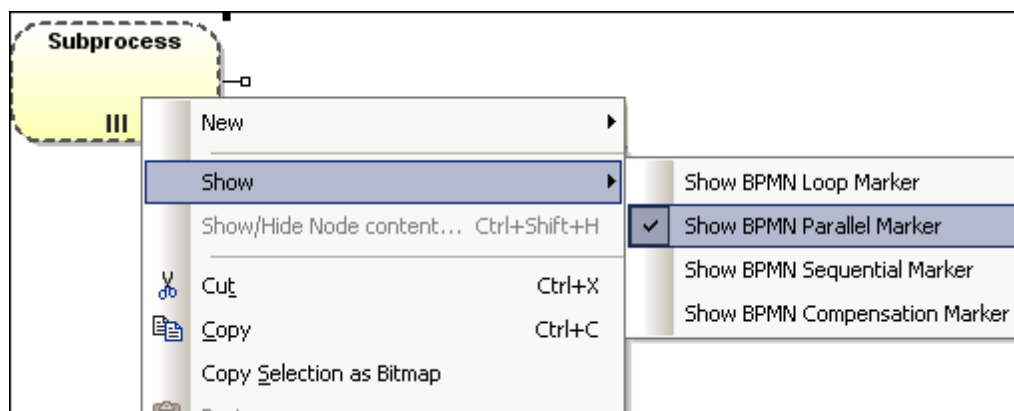


BPMN 2.0 Expanded Sub Processes:



To define a Loop, Parallel, Sequential or Compensation marker:

1. Right click the inserted task and select the specific marker, e.g. **Show | Show BPMN Parallel Marker**.

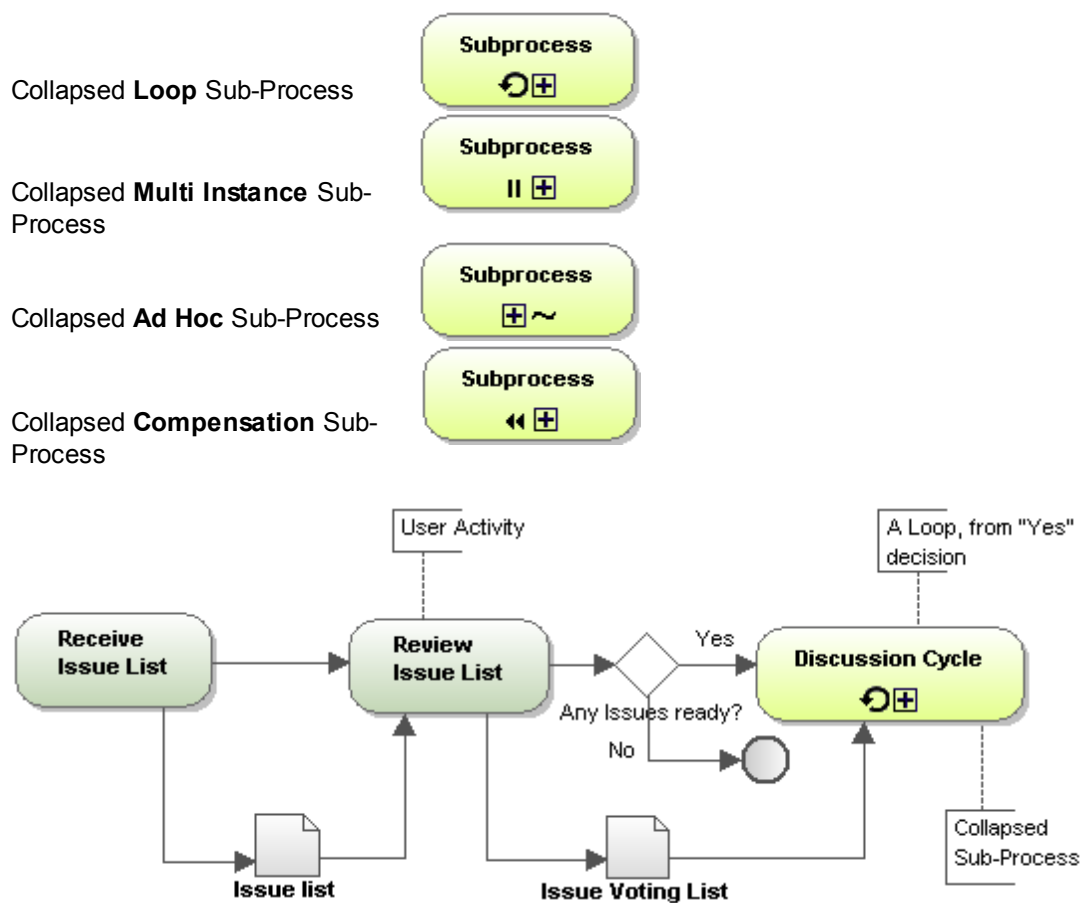


Collapsed Sub Processes

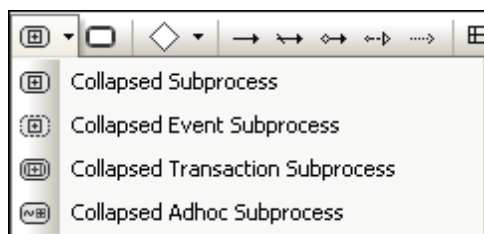
Collapsed versions of sub-processes hide the process detail. The specific type of Sub-Process is shown by the icon within the Sub-Process element.

Collapsed Sub-Process



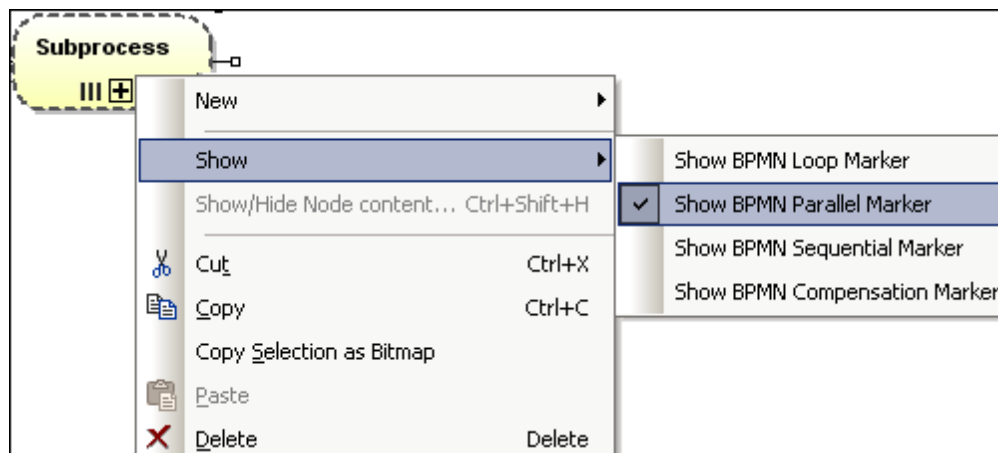


BPMN 2.0 Collapsed Sub Processes:



To define a Loop, Parallel, Sequential or Compensation marker:

1. Right click the inserted task and select the specific marker, e.g. **Show | Show BPMN Parallel Marker**.

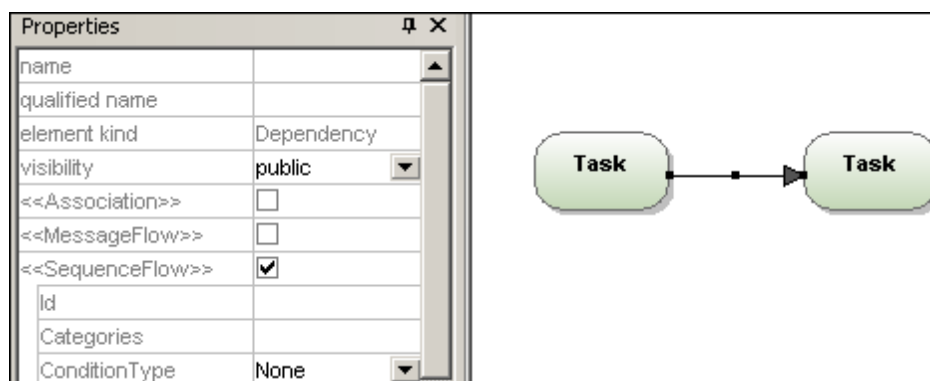


Connecting objects

There are two ways of connecting objects: a Flow (using a sequence or message), and an Association.

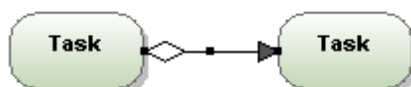
Sequence Flow

A Sequence Flow shows the order that activities are performed within a Process.



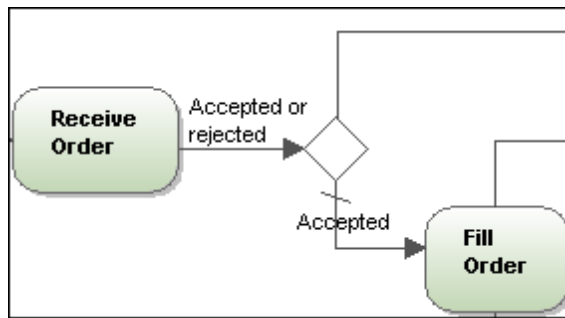
Conditional Flow

This type of Sequence Flow can have a conditional expression which is evaluated to determine if the flow will be used or not. If the conditional flow originates from an activity, then a mini diamond is displayed at the origin of the arrow.



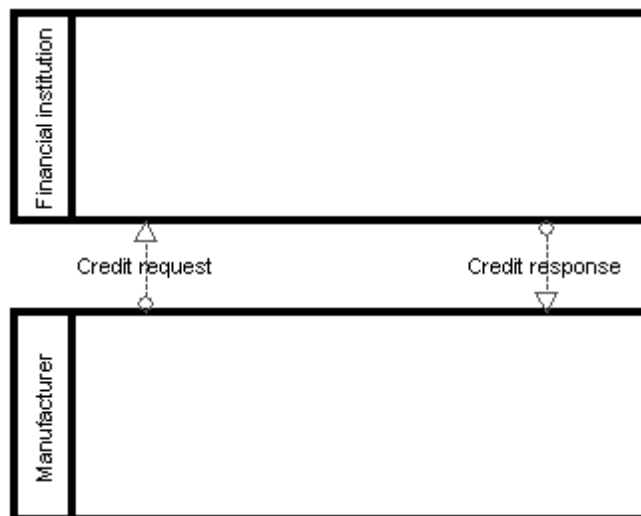
Default Flow

This type of flow is used if all other conditional flows are "false" in Data-Base Exclusive, or Inclusive decisions. A **diagonal slash** at the beginning of the arrow line is used as a visual indication, e.g. "Accepted" default flow.



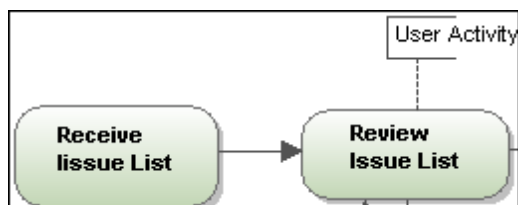
Message Flow

A Message Flow shows the flow of messages between two participants (entities or roles), that can send and receive them. Participants are shown as separate Pools in the diagram.



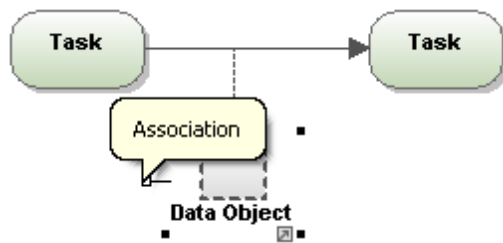
Association

Associations are used to associate Text and non-Flow Object data with Flow Objects, and show how data are input and output from Activities. The diagram below shows a Text annotation which provides the additional information "User Activity" for the Task "Review Issue List".



To create an Association between a Data Object and a Flow control:

1. Click the Association handle of the Data Object (on the left of the object).
2. Drag the connector onto the Flow Control arrow which is highlighted when you can drop it.



Alternatively,
Click the Association icon and drag from the Data Object to the Flow Control.

Pools / Swimlanes

Pool

Pools are used to partition and organize activities. A business process may show the interaction between various processes or participants. Each participant is represented by a rectangular box called a Pool. A participant could be a business role or entity.



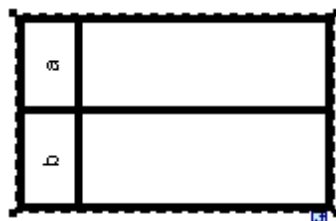
- BPMN objects placed within a pool become part of it when the pool boundary is highlighted.
- Objects within a pool can be individually selected using **Ctrl+Click**, or by dragging the marquee inside the pool.
- Click the pool boundary, or title, and drag to reposition it.

Lane

Pools can be further subdivided into Lanes, which categorize activities within a pool. Note that both horizontal and vertical lanes can be defined.

To add a new lane to a pool:

1. Right click the **header** of an existing pool object and select **New | Lane**. This adds a new lane to the pool. Each lane can be named separately, by double clicking in the name field.



Note:

Right clicking in one of the **lanes** allows you to add any of the elements allowed to be placed in a pool using the New option.

Artifacts

Artifacts allow you to show additional information about a Process i.e. how data, documents and other objects are used and updated during the business process. Artifacts are not directly related to sequence, or message flow, of the process.

Data Object

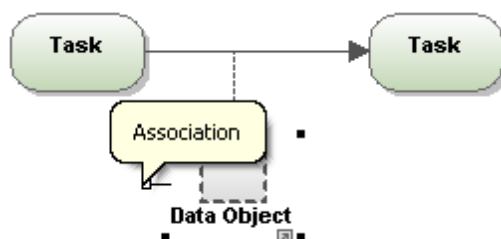
Data Objects are documents or other types of data, that show how data are used during a business process. Data objects can be used to define the input and output of data to/from activities.



Data Object

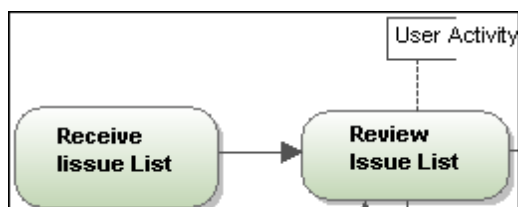
To create an Association between a Data Object and a Flow control:

1. Click the Association handle of the Data Object (on the left of the object).
2. Drag the connector onto the Flow Control object which is highlighted when you can drop it.



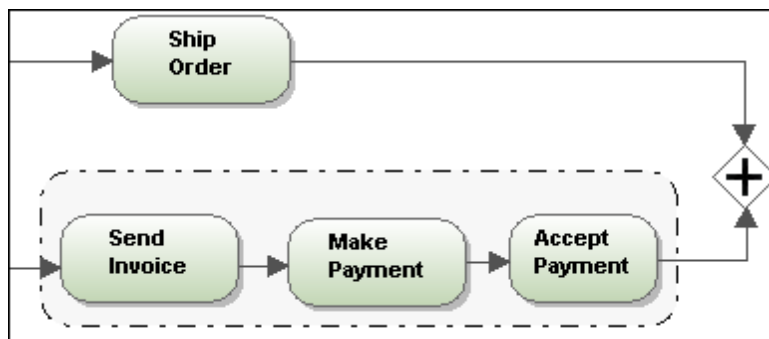
Text Annotation

Text Annotations allow you to annotate various sections of a business process and are connected to the specific object using an association.



Group

Groups are often used to highlight certain sections of a diagram, even across different pools. Groups cannot connect to a sequence or message flow. Group objects are generally placed behind task or process objects in the diagram.

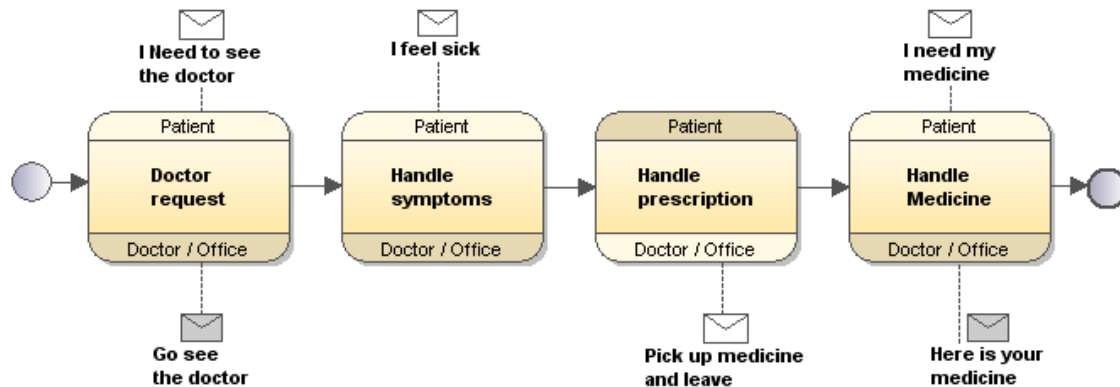


Choreography diagram

Choreography Diagrams specify the way business participants **coordinate** their interactions. They can also be seen as a business contract between participants, where the focus lies on the exchange of information (Messages) between the participants.

Business contracts are often in the form of a purchase order sent to a supplier, the confirmation by the supplier to process the order, then the fulfilling of the order. Choreographies also have Activities ordered by Sequence Flows.

Activities comprise of one or more interactions between the various participants. Interactions are often called Message Exchange Patterns (MEP). A MEP is the "Activity" of a choreography, and can also be called a **Choreography Task**.



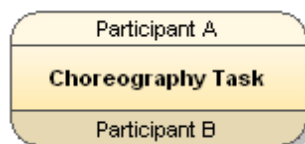
Choreography Tasks

There are four types of choreography tasks that can be inserted into the diagram:

-  Choreography Task
-  Sub Choreography Task (collapsed)
-  Sub Choreography Task (expanded)
-  Call Choreography Task

To insert a choreography task:

- Click the Task icon of the Task that you want to insert, e.g. Choreography Task, then click in the Choreography Diagram.



The screenshot shows the default view when the Task is inserted; the "Choreography Task" text is automatically highlighted.

- Enter text to rename the Choreography Task.
- Click in the top **band** to enter the name of Participant A, and in the bottom band to name Participant B.

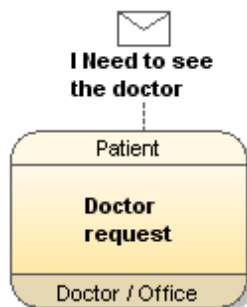
The Participant bands are shown as shaded/unshaded. The **initiator** of the Activity is the **unshaded** Participant, which is Participant A when the Task is first inserted.

To add/associate messages to a Choreography Task:

1. Click the message icon  in the icon bar, then click in the diagram to insert it.

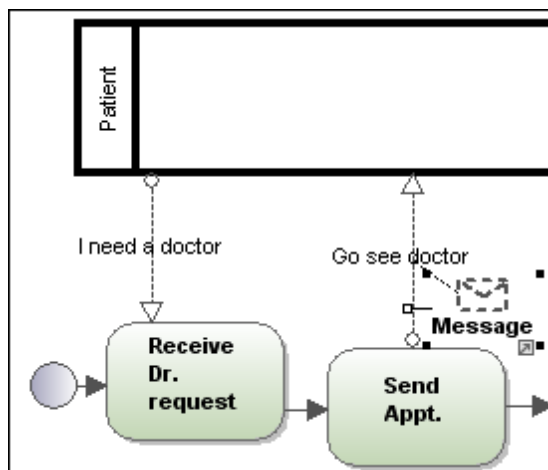


2. Enter the name of the message, e.g. "I need to see the doctor".
3. Click the **Association** handle (on the left) and drag it to the Choreography Task you want to associate it to.



To add a message to a line e.g. association:

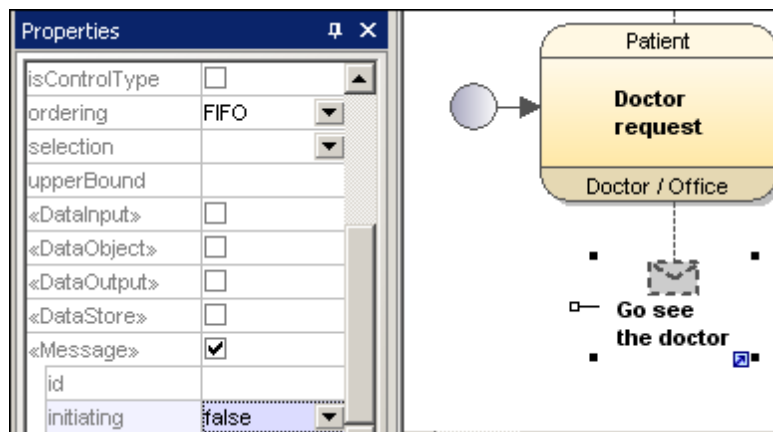
1. Click the **line** that you want to add the message to.
2. Click the Message icon in the icon bar.
3. Click the same line again to attach it.



The message is placed on top of the line and automatically attached to it.

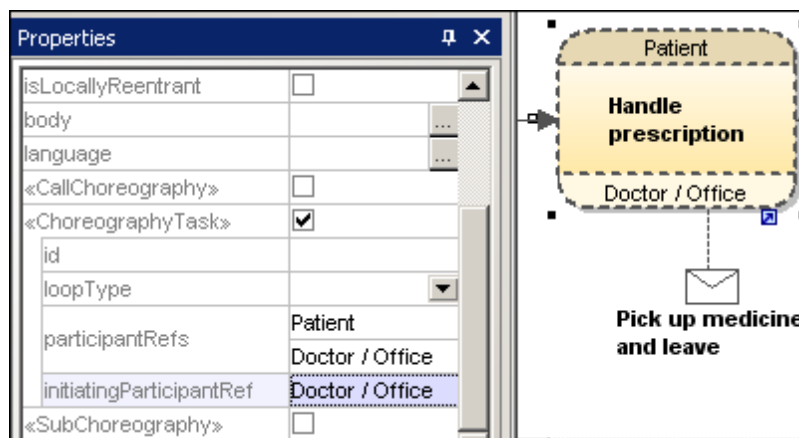
To change the initiating Message / Participant:

- When inserting a **Message**, it will automatically be defined as the initiating message, i.e. it is unshaded.
1. Click the Message and select **false** from the "initiating" combo box, in the Properties tab.



The message element is now shaded.

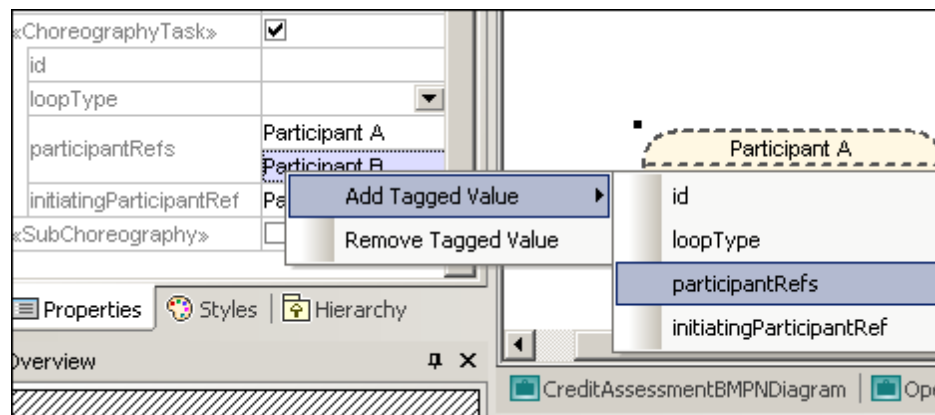
- When inserting a **Choreography Task**, Participant A is automatically defined as the Initiating Participant.
 1. Click the Choreography Task that contains the Participant you want to be the initiator.
 2. Enter the name of the Participant you want to define as the initiator in the "InitiatingParticipantRef" combo box, e.g. "Doctor / Office".



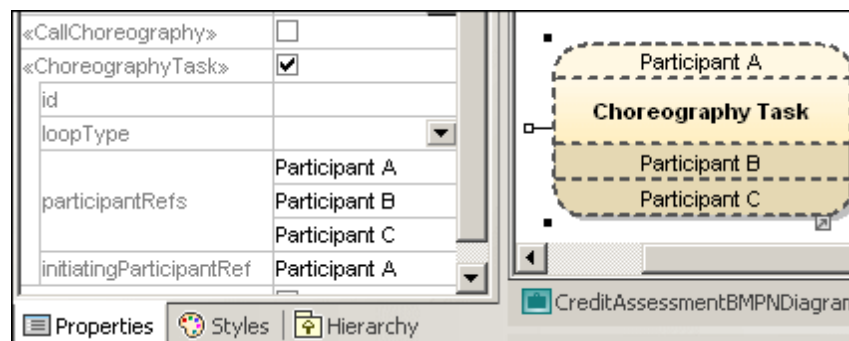
The Doctor / Office band is now unshaded, showing that it is the Initiating Participant. The Patient band is now shaded.

To add new Participants to a Choreography Task:

1. Click the Task you want to add the Participant to in the diagram window.

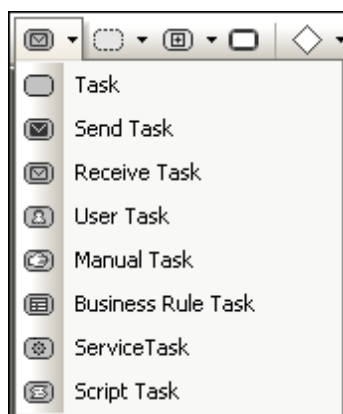


2. Right click the **participantsRefs** field in the Properties tab and select **Add Tagged Value | participantRefs**.
3. Enter the name of the new Participant e.g. Participant C.

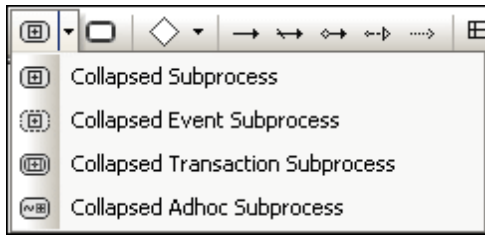


Tasks and Subprocesses

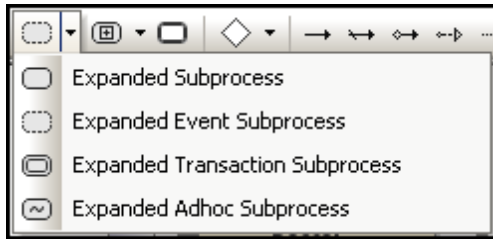
Click the Task drop-down icon to insert the specific Task.



Click the Collapsed Subprocess drop-down icon to insert the specific Collapsed Subprocess.



Use the Expanded Subprocess drop-down icon to insert the specific Expanded Subprocess.



Data Objects

Data is represented by five modeling elements and are inserted by clicking one of the following icons:



Data Object

Represents information flowing through the process, e.g. emails, business documents, etc. Data objects provide information about what activities require to be performed and/or what they produce.



Data Output

Represents the result of the process.



Data Input

Is an external input for the entire process. Can be read by an activity.



Data Collection

Represents a collection of information e.g. a list of order items.



Data Store

A place where the process can read or write data e.g. a database.

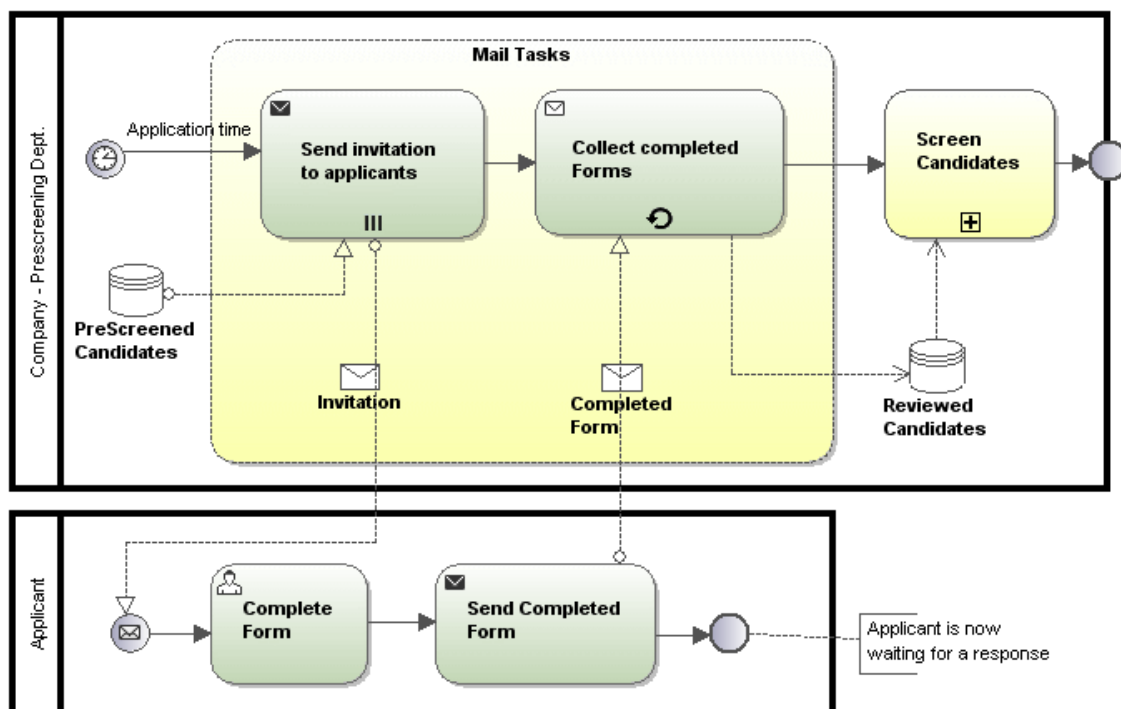
Collaboration diagram

Collaboration Diagrams specify the **interactions** between two or more processes.

A Collaboration generally consists of two or more pools which represent the participants in the collaboration. Message exchanges between participants are shown by Message Flows that connect the two pools, or the objects within the pools. Pools may also be empty, in this case they are black boxes.

All combinations of Pools, Processes, and a Choreography are allowed in a Collaboration

diagram.



Conversations

A Conversation is a simplified version of a Collaboration, and has access to the same modeling elements. A Conversation defines a set of logically related message exchanges, where the message exchanges are related to each other reflecting a distinct business scenario, e.g. a request followed by a response.

A Conversation has two other graphical elements not available in any other BPMN diagrams:

- Conversation node elements (Conversation, Sub-Conversation and Call-Conversation)
- Conversation links



Conversation



Sub-Conversation



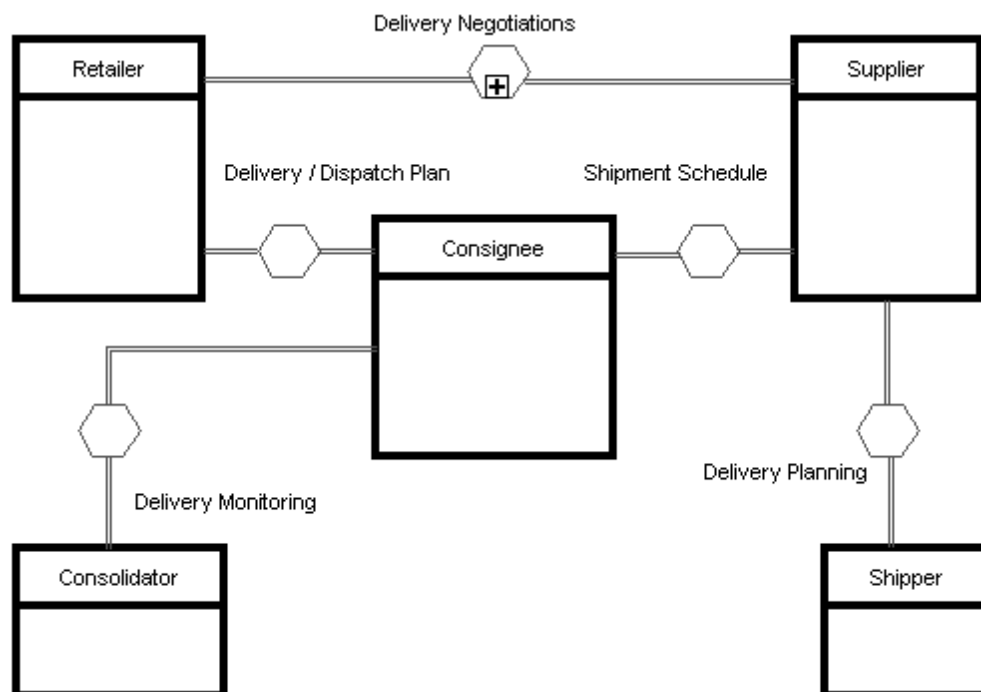
Call-Conversation



Participant / Pool

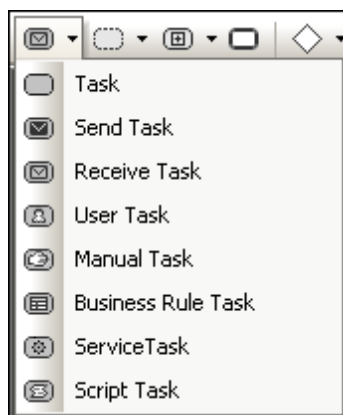


Conversation Link

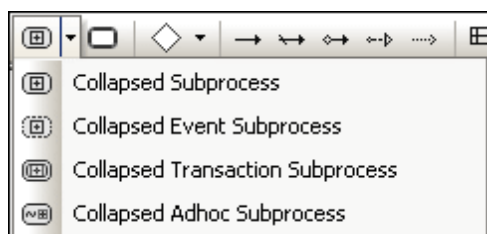


Tasks and Subprocesses

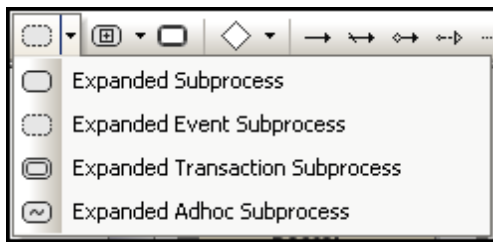
Click the Task drop-down icon to insert the specific Task.



Click the Collapsed Subprocess drop-down icon to insert the specific Collapsed Subprocess.



Use the Expanded Subprocess drop-down icon to insert the specific Expanded Subprocess.



Data Objects

Data is represented by five modeling elements and are inserted by clicking one of the following icons:



Data Object

Represents information flowing through the process, e.g. emails, business documents, etc. Data objects provide information about what activities require to be performed and/or what they produce.



Data Output

Represents the result of the process.



Data Input

Is an external input for the entire process. Can be read by an activity.



Data Collection

Represents a collection of information e.g. a list of order items.

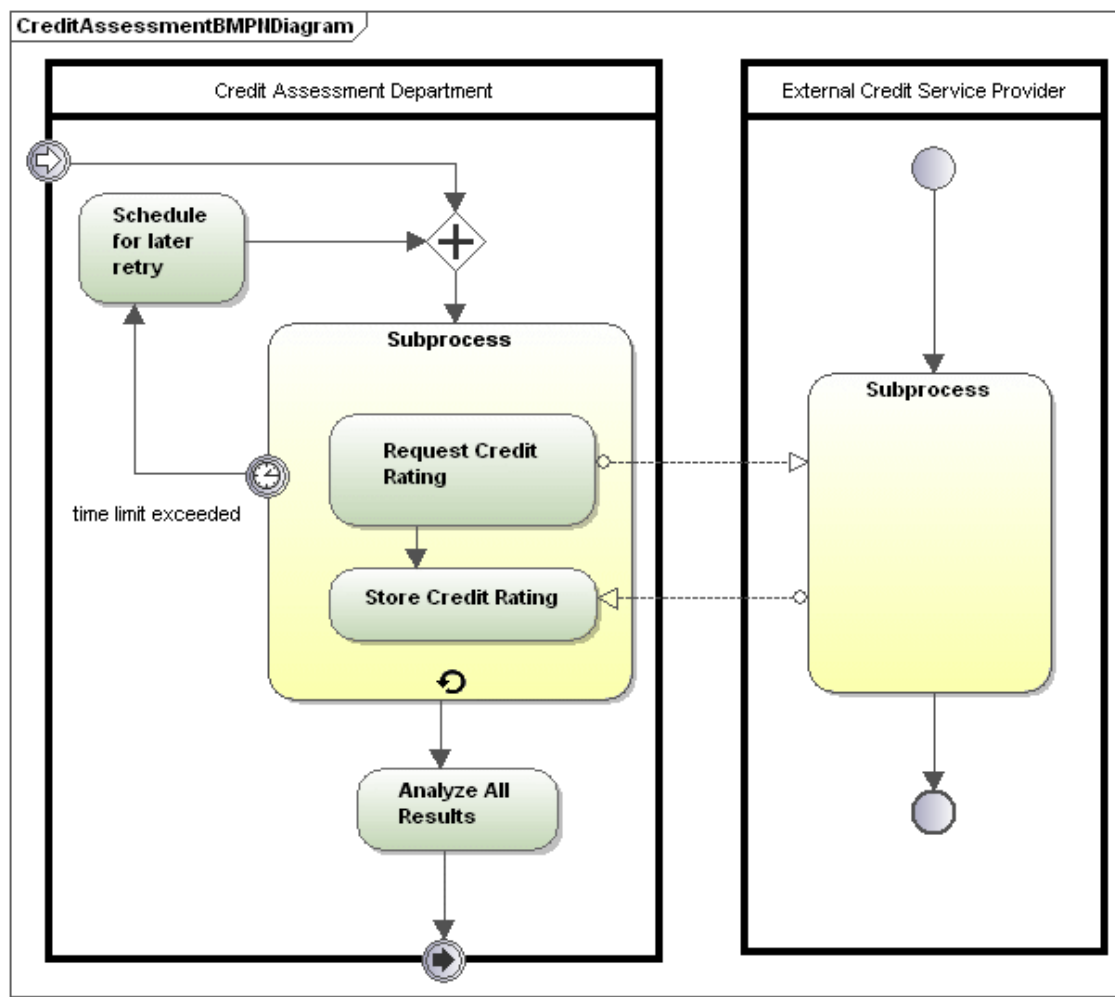


Data Store

A place where the process can read or write data e.g. a database.

Standard Business Process diagram BPMN 2.0

Business Process diagrams cover a wide range of information and employ several different types of modeling, to create Business Processes.



There are three types of Business Processes:

Private non-executable (internal) Business Processes:

- Non executable Processes are those where there is not enough detail for the process to execute; generally during the development cycle.

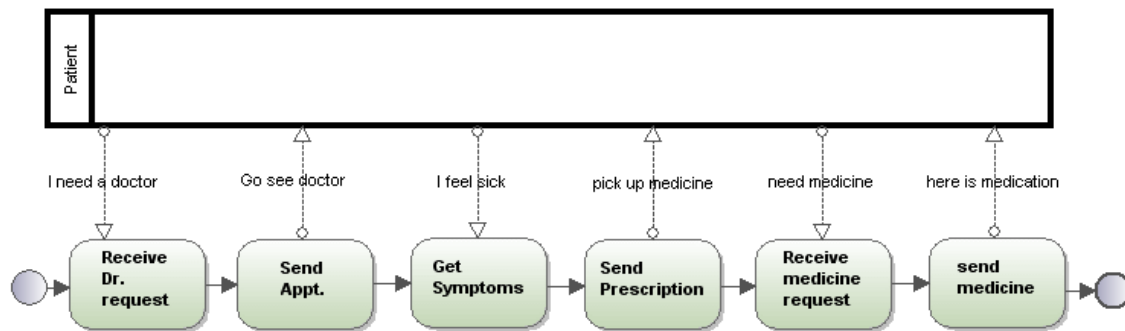
Private executable (internal) Business Processes:

- Executable Processes are processes that are executable, due to the fact that they have been completely modeled according to the BPMN 2.0 semantics.



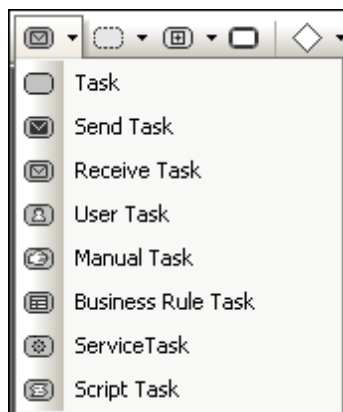
Public Processes:

- Define the interaction between a **private** process and a separate process, or participant. E.g. Doctor, Patient interactions.

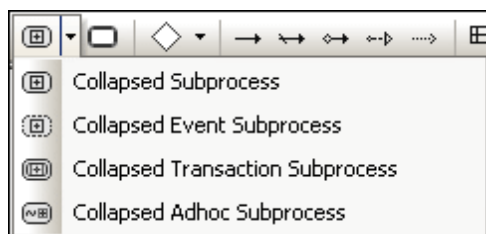


Tasks and Subprocesses

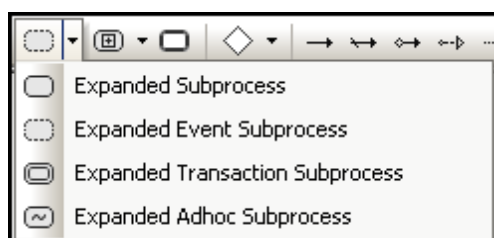
Click the Task drop-down icon to insert the specific Task.



Click the Collapsed Subprocess drop-down icon to insert the specific Collapsed Subprocess.



Use the Expanded Subprocess drop-down icon to insert the specific Expanded Subprocess.



Data Objects

Data is represented by five modeling elements and are inserted by clicking one of the following icons:



Data Object

Represents information flowing through the process, e.g. emails, business documents, etc. Data objects provide information about what activities require to be performed and/or what they produce.



Data Output

Represents the result of the process.



Data Input

Is an external input for the entire process. Can be read by an activity.



Data Collection

Represents a collection of information e.g. a list of order items.



Data Store

A place where the process can read or write data e.g. a database.

10.3.3 SysML Diagrams

Altova web site:  [Modeling SysML diagrams in UModel](#)

SysML 1.2 is a graphical modeling language for system engineers and supports specification, design and validation of systems including hardware, software, procedures etc. SysML models and data can be exchanged via XML in UModel.

SysML introduces two new diagram types and modifies three others while retaining some of the UML 2.0 diagrams.

New diagrams:	Requirement and Parametric diagrams
Modified UML diagrams:	Activity, Block Definition and Internal Block diagrams
Unchanged UML diagrams:	Sequence Diagrams, State Machine Diagrams, Use Case Diagram, and Package Diagram

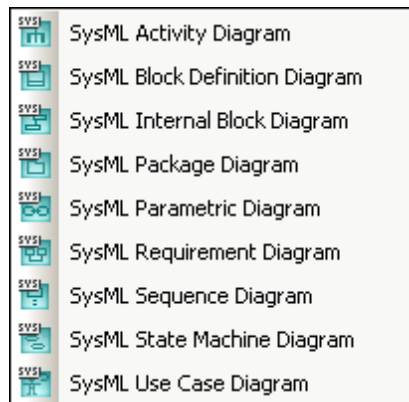
SysML diagrams are split into two groups:

- [Structure](#) diagrams
- [Behavior](#) diagrams

SysML diagrams are available through the **New Diagram | SysML diagrams** context menu option in the Model Tree tab.

To insert a SysML diagram:

1. Right click a package in the Model Tree tab.
2. Select the SysML diagram you want to insert, e.g. **New Diagram | SysML diagram | SysML Activity Diagram**.



Please note:

SysML diagrams are created in their own SysML package, when you try to add them to the root package.

Structure diagrams

The Structure diagrams available in SysML are:

- [Block Definition](#) diagram
- [Internal Block Definition](#) diagram
- [Parametric](#) diagram

- [Package](#) diagram

The Block Definition and Internal Block definition diagrams are modified UML 2.0 diagrams while the Parametric diagram is new. The Package diagram follows the UML 2.0 standard.

Blocks are used to describe the structure of an element or system, and can have multiple compartments that define its characteristics. Note that behaviors can be assigned to a block.

Block Definition Diagrams

describe the relationship between blocks: associations, generalizations, dependencies.

Internal Block Diagrams

describes the **internal** structure of a block in terms of properties and connectors between properties, i.e. the relationship of its constituent parts; using ports, connectors, and flows.

Parametric diagrams

use constraint blocks, defined in a Block Definition Diagram, to constrain the properties of other blocks in the Parametric diagram and integrate engineering analysis with design modeling.

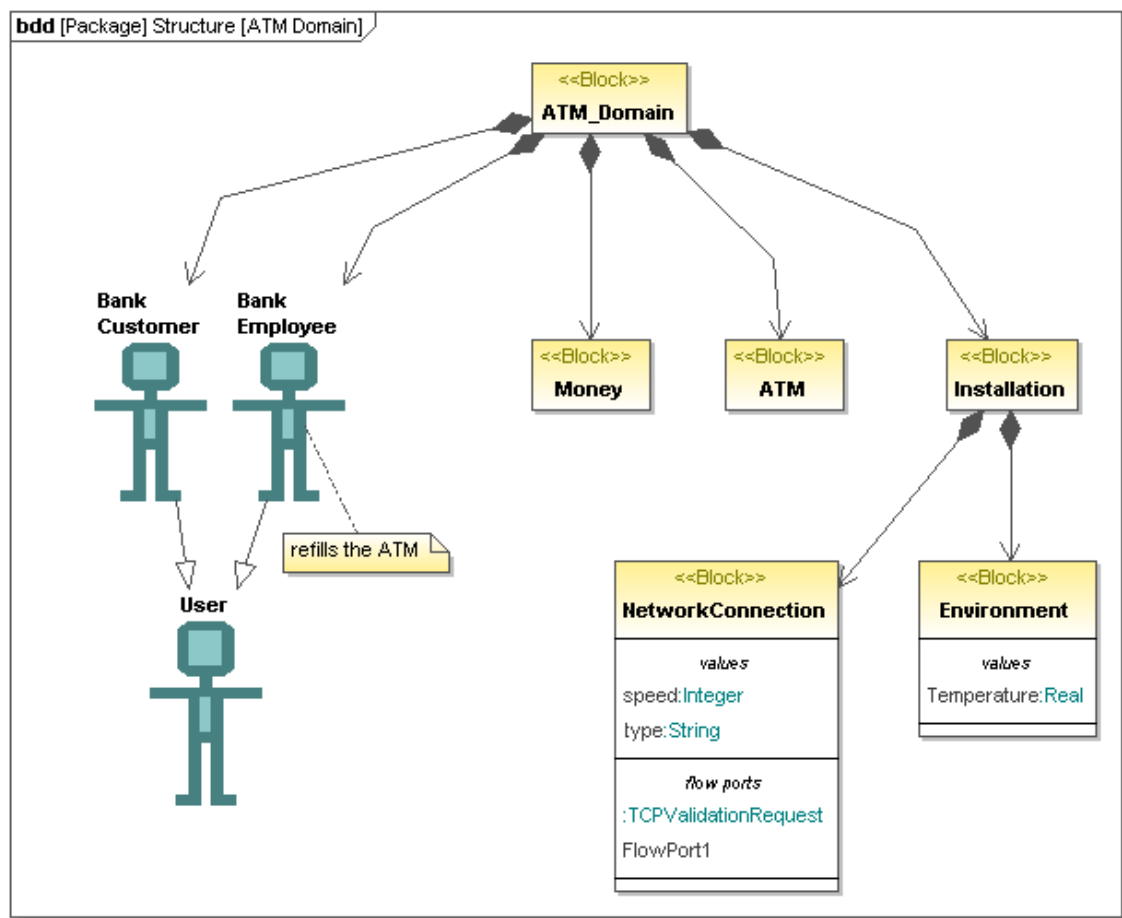
Package diagrams

organize model elements into packageable elements and define dependencies between the packages and/or model elements within the package.

Block Definition diagram

Block Definition Diagrams are based on the [UML class](#) diagram with restrictions and extensions as defined by SysML.

The Block Definition Diagram describes the relationship between various blocks, their associations, generalizations, as well as dependencies. This can lead to the modelling of a system hierarchy, or classification tree.

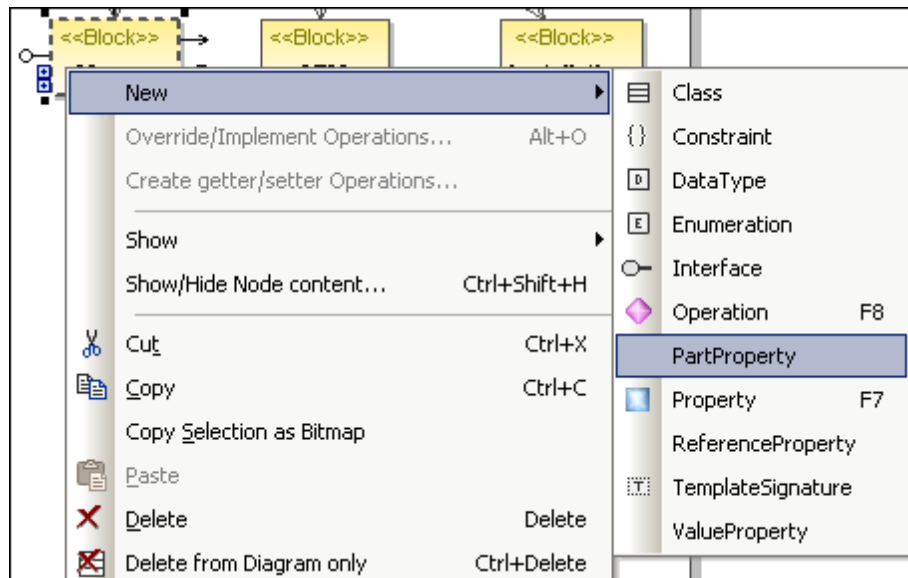


Blocks have several structures called "property"

- Part property
- Reference property
- Value property

To add a property to a block:

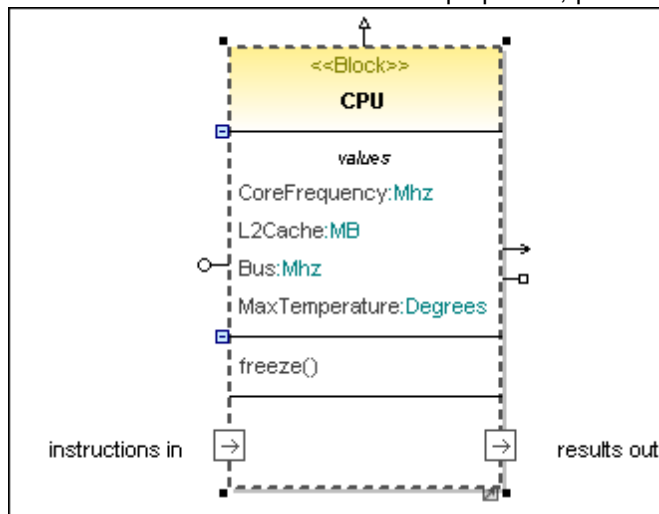
1. Right click an existing block and select **New | Property** (or PartProperty, ReferenceProperty, ValueProperty etc.).

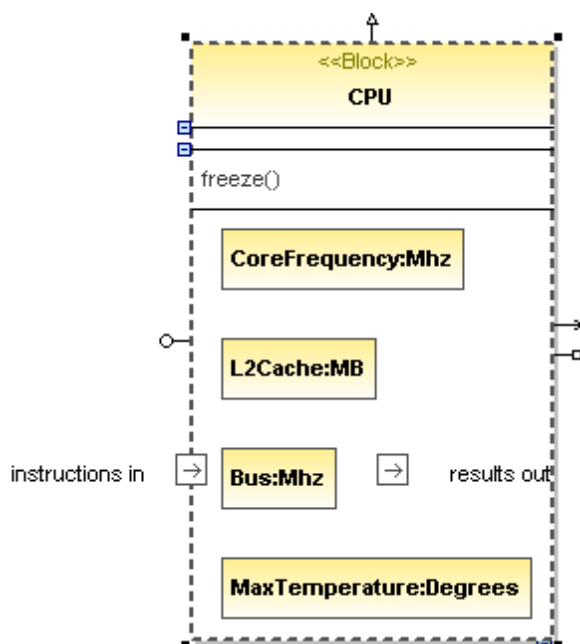


Note: depending on what you insert, a new compartment is added to the Block, e.g. **parts** for a PartProperty, or **values** for a ValueProperty.

To show block properties as nodes:

1. Right click the Block and select **Show | Show properties as nodes on node**. This shows the various Block properties, ports etc. as 'objects' in a compartment.

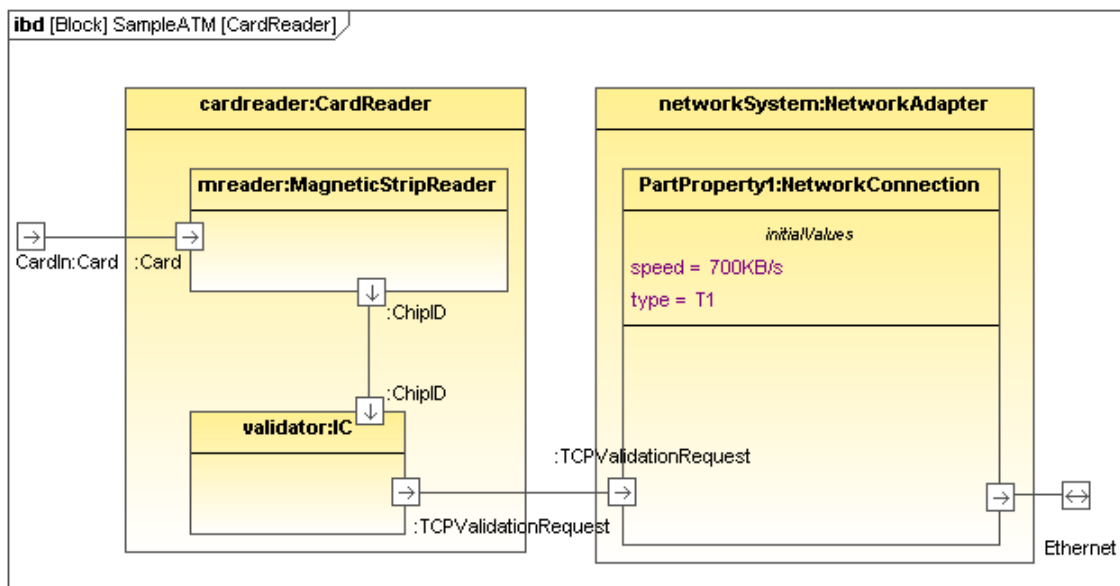




Internal Block diagram

Internal Block Diagrams are based on the UML [Composite Structure Diagram](#) with restrictions and extensions as defined by SysML.

The Internal Block Diagram describes the **internal** structure of a block in terms of properties and connectors between properties, i.e. the relationship of its constituent parts, using ports, connectors, and flows.



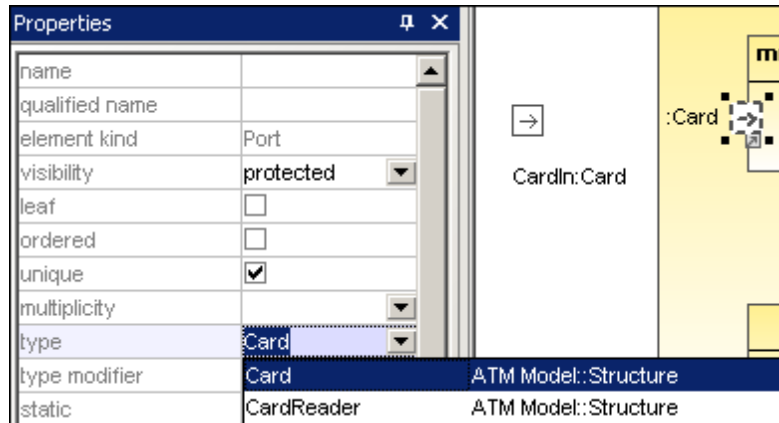
To add initial values to a Property:

1. Right click the Property and select **New | Initial Values**.
2. Double click the placeholder and enter the values e.g. speed = 700KB/s

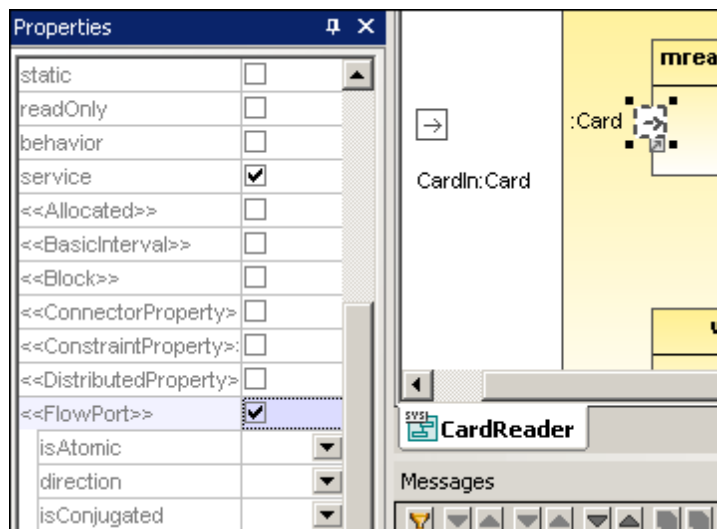
To insert a FlowPort:

Making sure that the type of the property is set to a block or classifier:

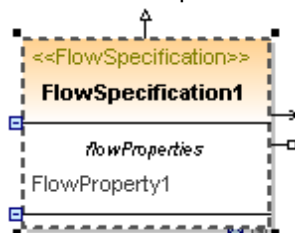
1. Click the FlowPort icon  in the icon bar, then click the border of Property that you want to add the port to.
The FlowPort is now placed on the border of the Property.



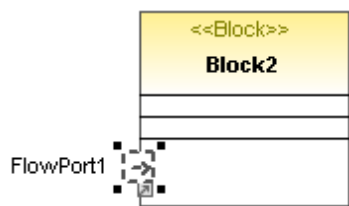
2. Enter the **name** of the FlowPort in the name field of the Properties tab if necessary.
3. Select the port's **type** from the type list box e.g. Card.
While the port icon is still selected:
4. Scroll down the Properties tab and notice that additional FlowPort parameters are available.

**To create a atomic conjugated flow port:**

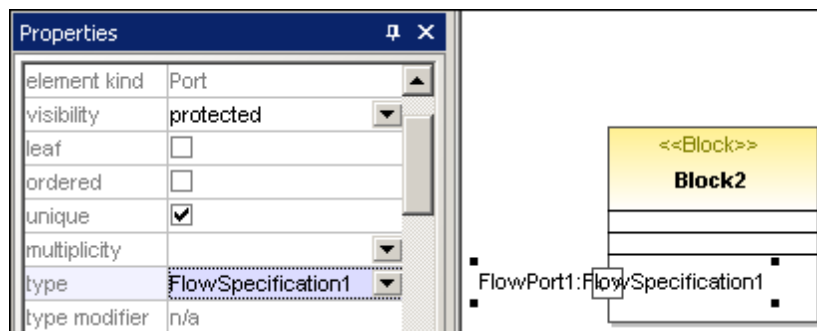
1. Create a FlowSpecification (interface) in a Block Definition Diagram.



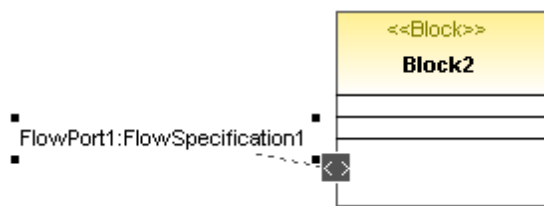
- Click the Flow port you want to define in the Internal Block Diagram (or BDD).



- Set the Flow port "type" to a FlowSpecification in the Properties tab.



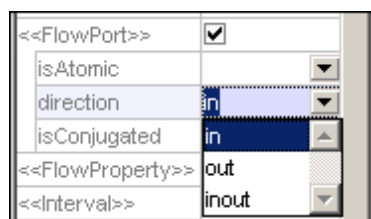
- Set 'isConjugated' to true.



This type of port is shown with a dark background.

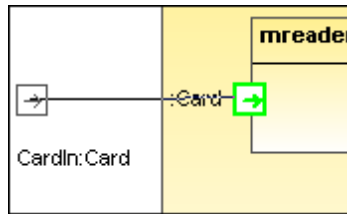
To change the FlowPort direction:

- Click the existing FlowPort entry and select the in, out, or inout, from the **direction** combo box.



To join two ports:

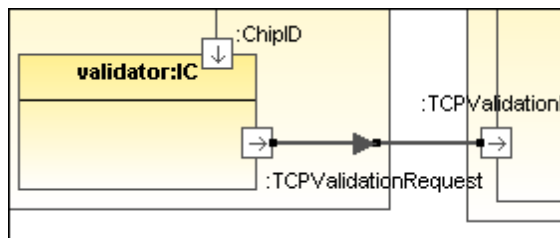
- Click the Connector icon  in the icon bar.
- Drag and drop from the first to the second port.
- Drop the connector on the port, when the port object is highlighted in the diagram.



To create Item flows:

Item flows can be created between block associations, or on other connectors between parts of SysML diagrams.

1. Right click an existing connector and select **New | Item flow** (L to R, or R to L).



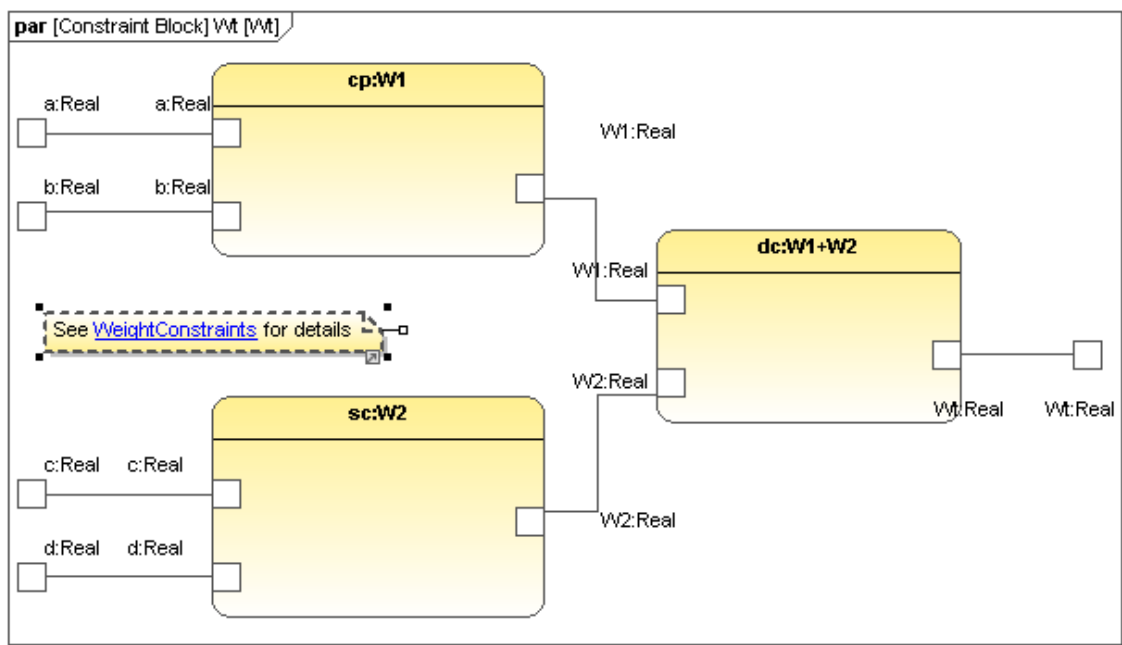
An arrowhead has been added to the connector displaying the direction of the item flow.

Parametric diagram

The Parametric diagram is a new diagram type specific to SysML that integrates engineering analysis with design modeling. A parametric diagram is similar to an [internal block diagram](#) with the exception that only those type of connectors may be shown, which are connected to constraint parameters on at least one of their ends.

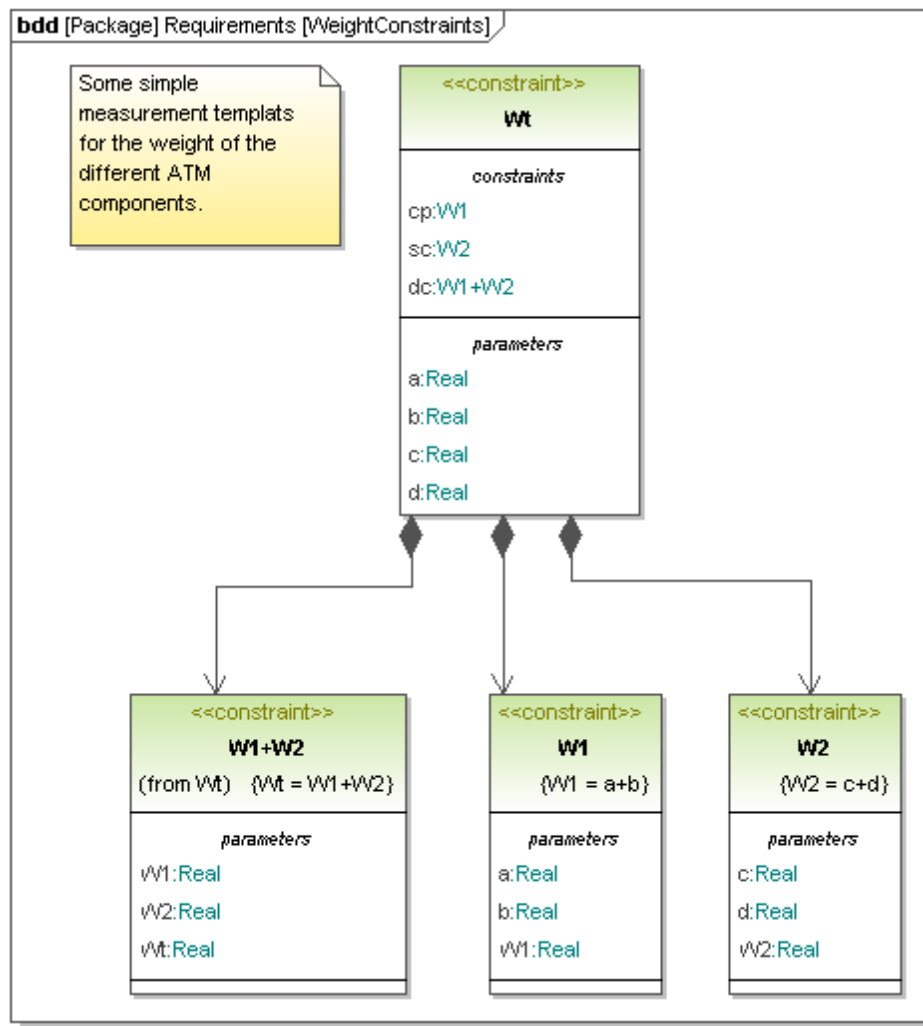
The Parametric diagram makes **use** of constraint blocks, defined in a Block Definition Diagram, to constrain the properties of other blocks in the Parametric diagram. Constraint blocks are shown with rounded corners rather than being square as an ordinary part.

Parametric Diagram Wt



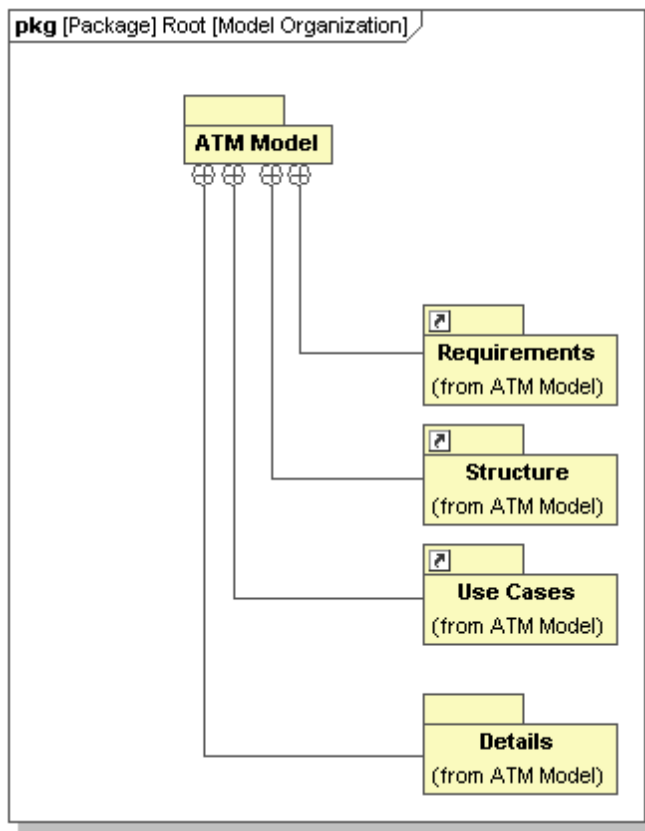
Block Definition Diagram - definition of constraints

The «constraint» keyword on a block definition states that the block is a constraint block. Parameters of the constraint are shown in a "**parameters**" compartment.



Package diagram

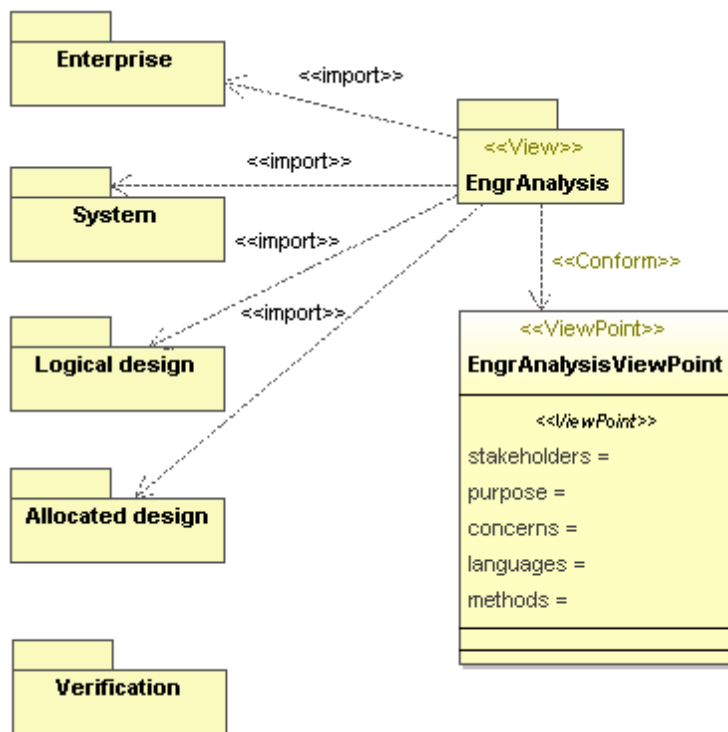
The package diagram is used to organize model elements into package-able elements and define dependencies between the packages and/or model elements within the package. The package defines a namespace for the package-able elements.



The model can be organized in multiple ways where each viewpoint shows a single aspect of the model e.g.:

- system hierarchy (by level)
- by diagram type (by domain)
- by viewpoints (IPT - integrated product team)

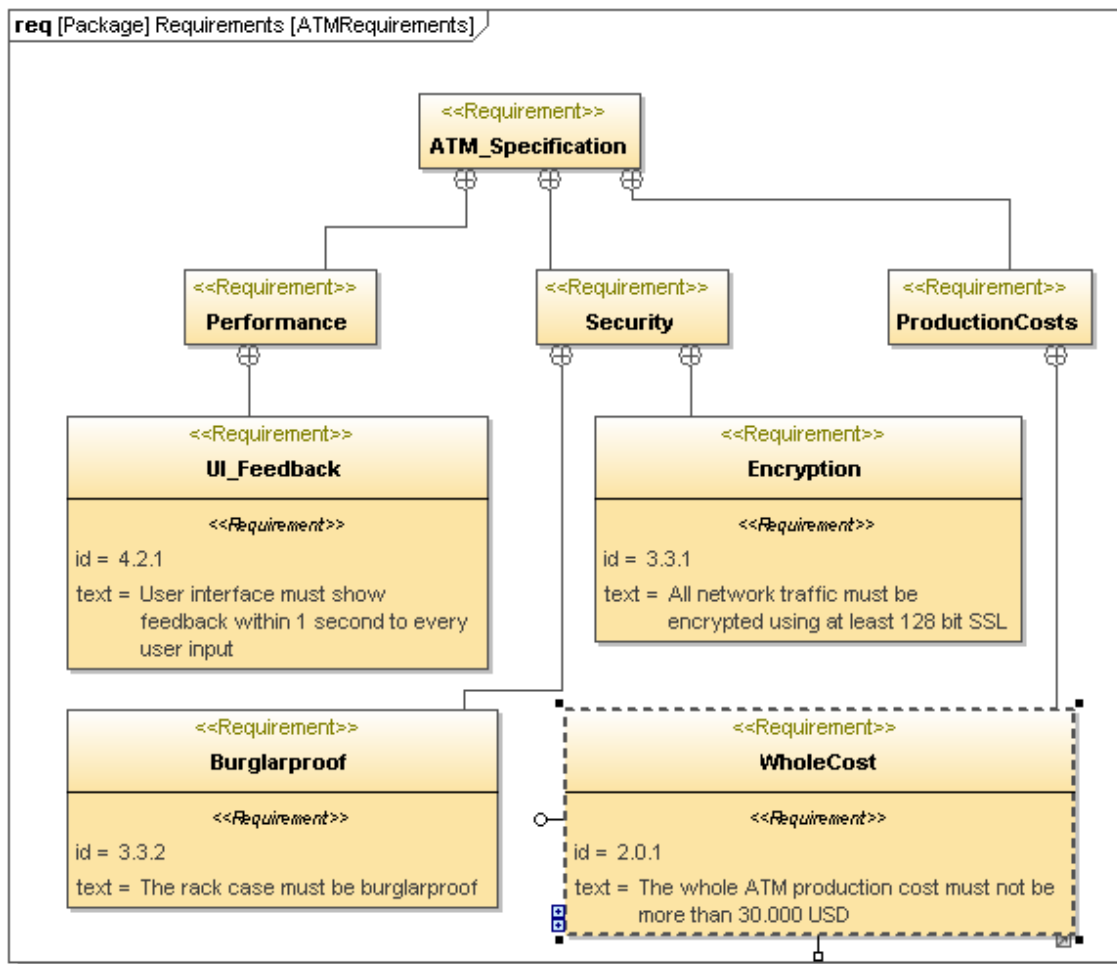
Viewpoints allow you to view the model in a different way, e.g. in an analysis view which shows multiple hierarchy levels, e.g. Enterprise, System, Logical design etc.



Requirement diagram

The requirement diagram is a new diagram type designed specifically for SysML and is used to integrate the behavior and structure models of SysML with engineering analysis models, e.g. performance, reliability models etc.

It is a modeling construct for text-based requirements, and the relationship between requirements and other model elements that satisfy or verify them.



Behavior diagrams

The Behavior diagrams available in SysML are:

- [Activity](#) diagram
- [Sequence](#) diagram
- [State Machine](#) diagram
- [Use Case](#) diagram

The Activity diagram is modified from UML 2.0 and is the one discussed in greater detail the following section, while the rest follow the UML 2.0 standard.

Activity diagram

The SysML Activity diagram is modified from UML 2.0. Please follow the links below to find out about the Activity diagram and how to create Activity diagrams using the various modeling elements.

- For a **general** introduction to Activity diagrams, see: [Activity Diagram](#)
- To **insert** Activity diagram elements, see [Inserting Activity Diagram elements](#)
- To create **branches** and merges, see [Creating branches and merges](#)

- To find an explanation with examples of Activity diagram **elements**, see [Activity Diagram elements](#)

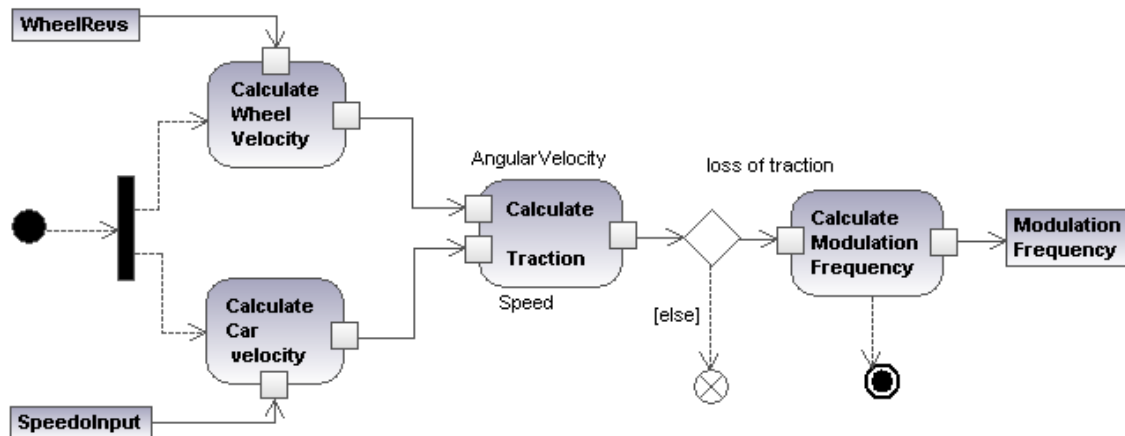
In UML all behaviors, Activities included, are classes and Activity **instances** are the actual execution of the activity. Creating an activity instance, starts the execution, while destroying the instance terminates it.

SysML extensions to Activities diagrams:

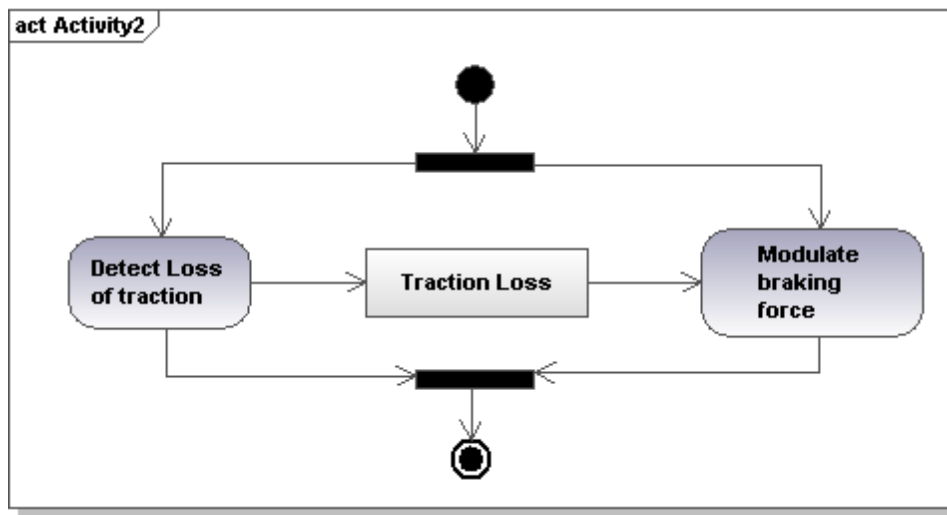
- Support for **continuous** flow modeling using the continuous stereotype (streaming activities)
- Alignment of Activities using the Enhanced Functional Flow Block diagram (EFFBD) (which is basically an Activity diagram, with Allocate Partitions)

In SysML, Activities can appear in Block Definition diagrams as regular blocks with the <<Activity>> keyword/stereotype.

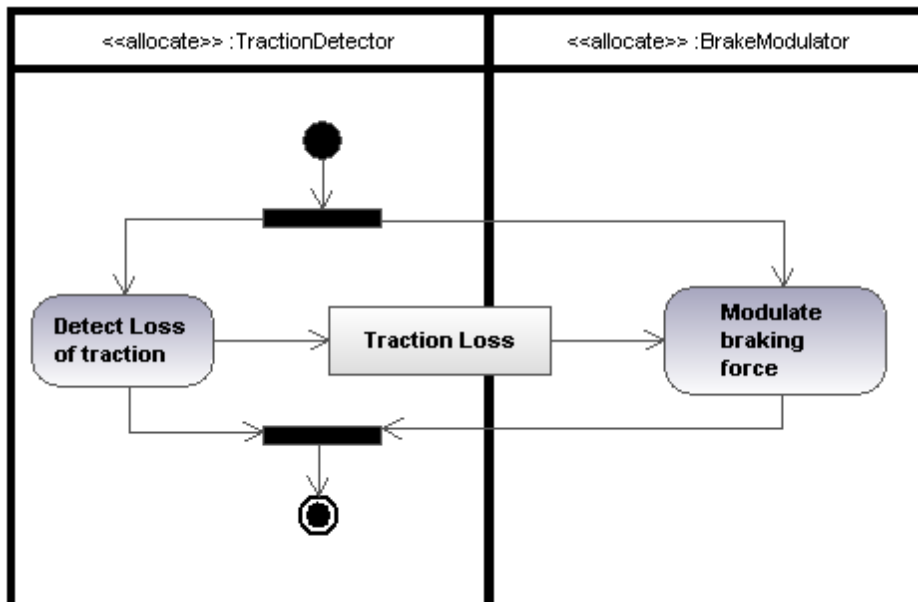
SysML Activity diagram:



Activity diagram without swimlanes



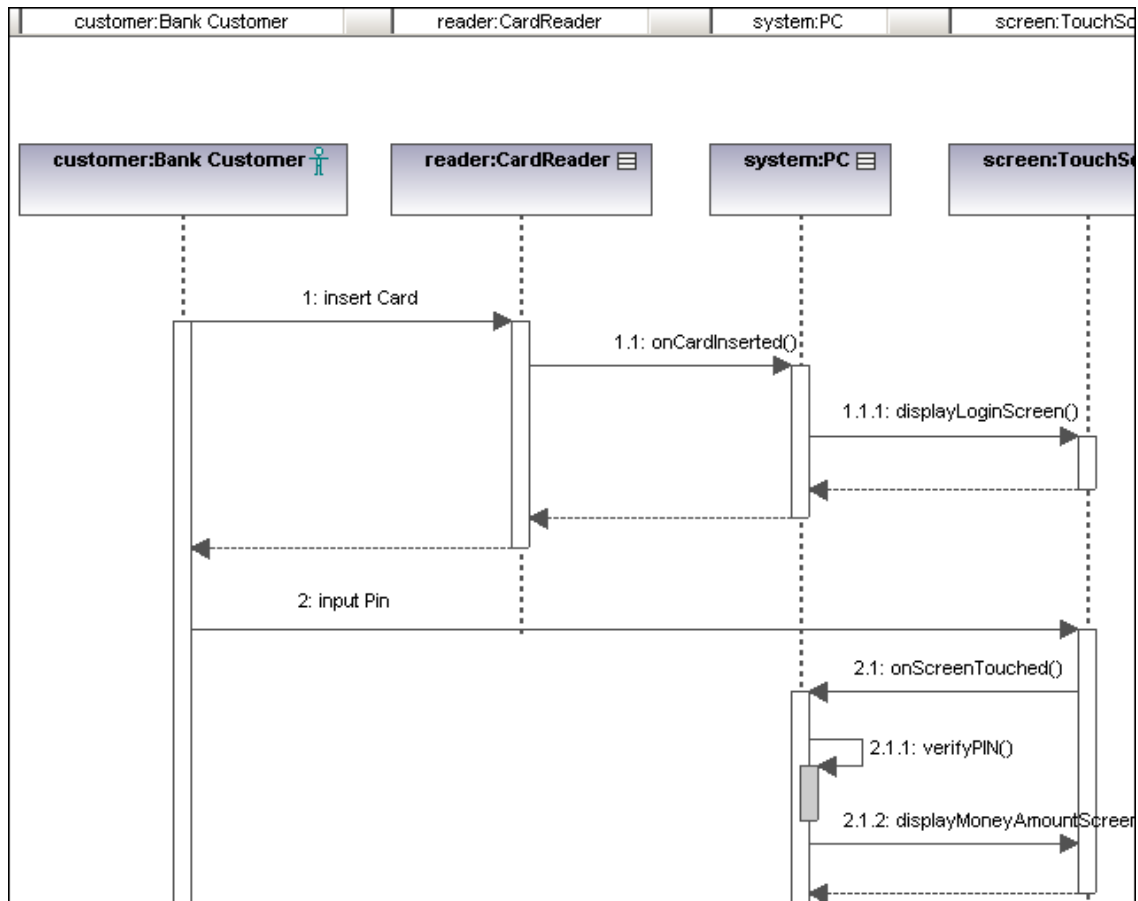
Activity diagram with swimlanes



Sequence diagram

The SysML Sequence diagram follows the UML 2.0 specification; for information on how to create Sequence diagrams use the following links:

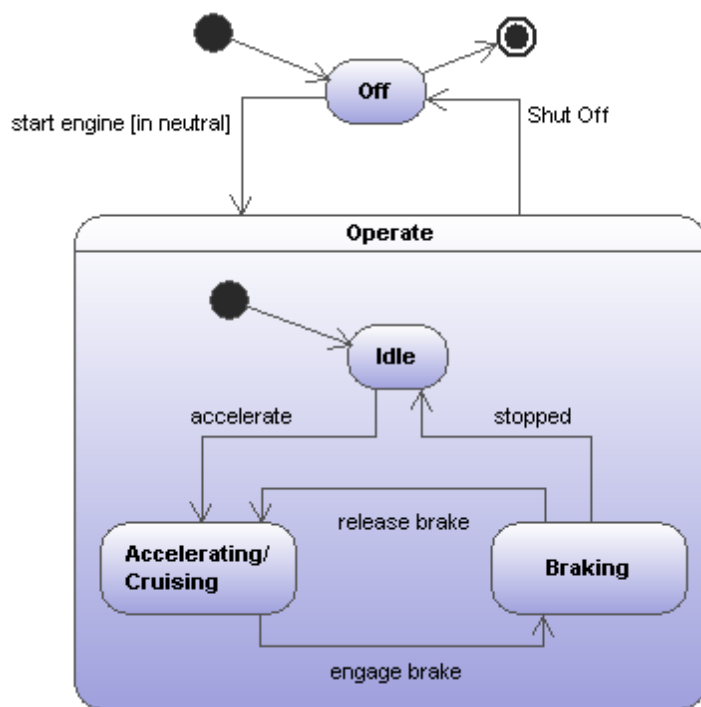
- For **general** introduction to Sequence diagrams, see [Sequence Diagram](#)
- To **insert** the various Sequence diagram elements: Lifelines, Combined fragments etc., see [Inserting sequence diagram elements](#)
- To **generate** Sequence diagrams from source code, see [Generating Sequence Diagrams from source code](#)

SysML Sequence diagram:**State Machine diagram**

The SysML State Machine diagram follows the UML 2.0 specification; for information on how to create State Machine diagrams use the following links:

- For a **general** introduction to State Machine diagrams, see [State Machine Diagram](#)
- To **insert** the various Sequence diagram elements, see [Creating states, activities and transitions](#)
- To **create** Composite States, see [Composite states](#)
- To find an explanation with **examples** of State Machine diagram elements, see [State Machine Diagram elements](#)

SysML State Machine diagram:

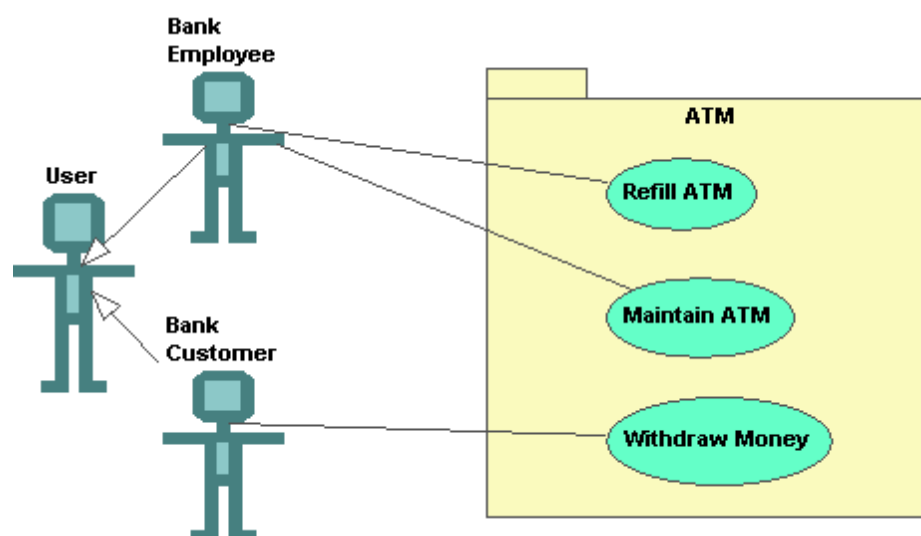


Use Case diagram

The SysML Use Case diagram follows the UML 2.0 specification; for information on how to create Use Case diagrams use the following links:

- To insert Use Case diagram elements, see the [Use Cases](#) section in the tutorial

SysML Use Case diagram



Chapter 11

UModel and Databases

11 UModel and Databases

UModel enables you to import/export SQL databases, and to display the database structure as database modeling elements in a UML Database Diagram.

The following databases are supported. The available root object for each database is also listed. While Altova endeavors to support other databases, successful connection and data processing have only been tested with the databases listed below. If your Altova application is a 64-bit version, ensure that you have access to the 64-bit database drivers needed for the specific database you are connecting to.

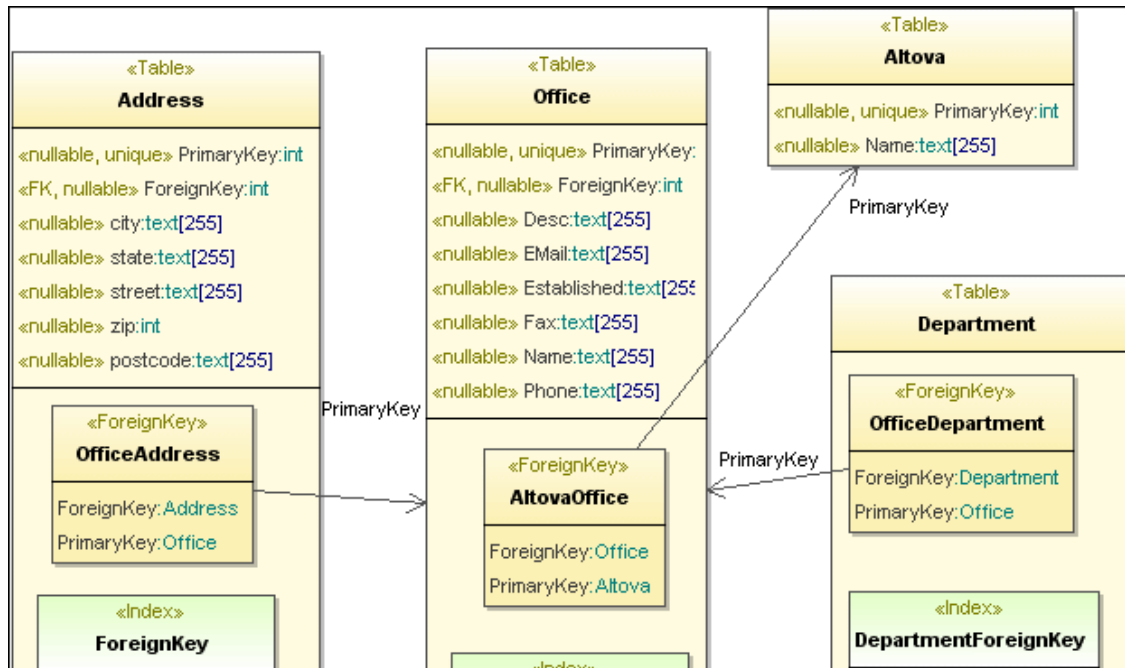
Database	Root Object	Notes
Firebird 2.5.4	database	
IBM DB2 8.x, 9.1, 9.5, 9.7, 10.1, 10.5	schema	
IBM DB2 for i 6.1, 7.1	schema	Logical files are supported and shown as views.
IBM Informix 11.70	database	
Microsoft Access 2003, 2007, 2010, 2013	database	
Microsoft SQL Server 2005, 2008, 2012, 2014	database	
MySQL 5.0, 5.1, 5.5, 5.6	database	
Oracle 9i, 10g, 11g, 12c	schema	
PostgreSQL 8.0, 8.1, 8.2, 8.3, 9.0.10, 9.1.6, 9.2.1, 9.4	database	
SQLite 3.x	database	SQLite connections are supported as native, direct connections to the SQLite database file. No separate drivers are required.
Sybase ASE15	database	

The following database elements can be **imported** to the database model:

- Tables
- Check Constraints
- Primary / Foreign / Unique keys
- Indices
- Views
- Triggers
- Stored procedures
- Functions

Note: Views, Triggers, Stored procedures and Functions can only be imported, though not added, in UModel.

Once the modeling process has been completed, the changed database model structure can be exported to the database using the "Merge Program Code from UModel project" menu item. This creates a database change script file which can be executed, or saved, for later execution.



11.1 Connecting to a Database

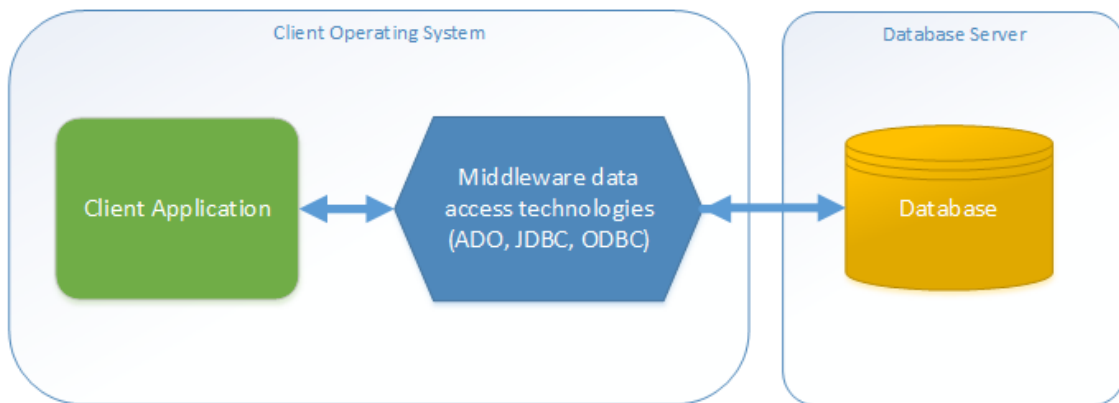
A database typically resides on a database server (either local or remote) which does not necessarily use the same operating system as the application that connects to it and consumes data. For example, while UModel runs on a Windows operating system, the database from which you want to access data (for example, MySQL) might run on a Linux machine.

UModel uses a database connection mechanism which relies on the data connection interfaces and database drivers that are already available on your operating system or released periodically by the major database vendors. In the constantly evolving landscape of database technologies, this approach caters for better cross-platform flexibility and interoperability. More specifically, UModel can access any of the major database types through the following data access technologies:

- ADO (Microsoft® ActiveX® Data Objects), which, in its turn, uses an underlying OLE DB (Object Linking and Embedding, Database) provider
- JDBC (Java Database Connectivity)
- ODBC (Open Database Connectivity)

Direct native connections to SQLite databases are also supported. To connect to a SQLite database, no additional drivers are required to be installed on your system.

The following diagram illustrates a simplified, generic representation of the typical data exchange between a client application such as UModel and a database.



Typical data exchange between a client application and a database server

Whether you should use ADO, ODBC or JDBC as a data connection interface largely depends on your existing software infrastructure. You will typically choose the data access technology and the database driver which integrates tighter (preferably natively) with the database system to which you want to connect. For example, to connect to a Microsoft Access 2013 database, you would build an ADO connection string that uses a native provider such as the **Microsoft Office Access Database Engine OLE DB Provider**. To connect to Oracle, on the other hand, you may want to download and install the latest JDBC or ODBC drivers from the Oracle website.

While drivers for Windows products (such as Microsoft Access or SQL Server) may already be available on your Windows operating system, they may not be available for other database types. Major database vendors routinely release publicly available database client software and drivers which provide cross-platform access to the respective database through any combination of OLE

DB, ODBC, or JDBC. In addition to this, several third party drivers may be available for any of the above technologies. In most cases, there is more than one way to connect to the required database from your operating system, and, consequently, from UModel. The available features, performance parameters, and the known issues will typically vary based on the data access technology or drivers used.

This section contains the following topics:

- [Starting the Database Connection Wizard](#)
- [Database Drivers Overview](#)
- [Setting up an ADO Connection](#)
- [Setting up an ODBC Connection](#)
- [Setting up a JDBC Connection](#)
- [Examples](#)

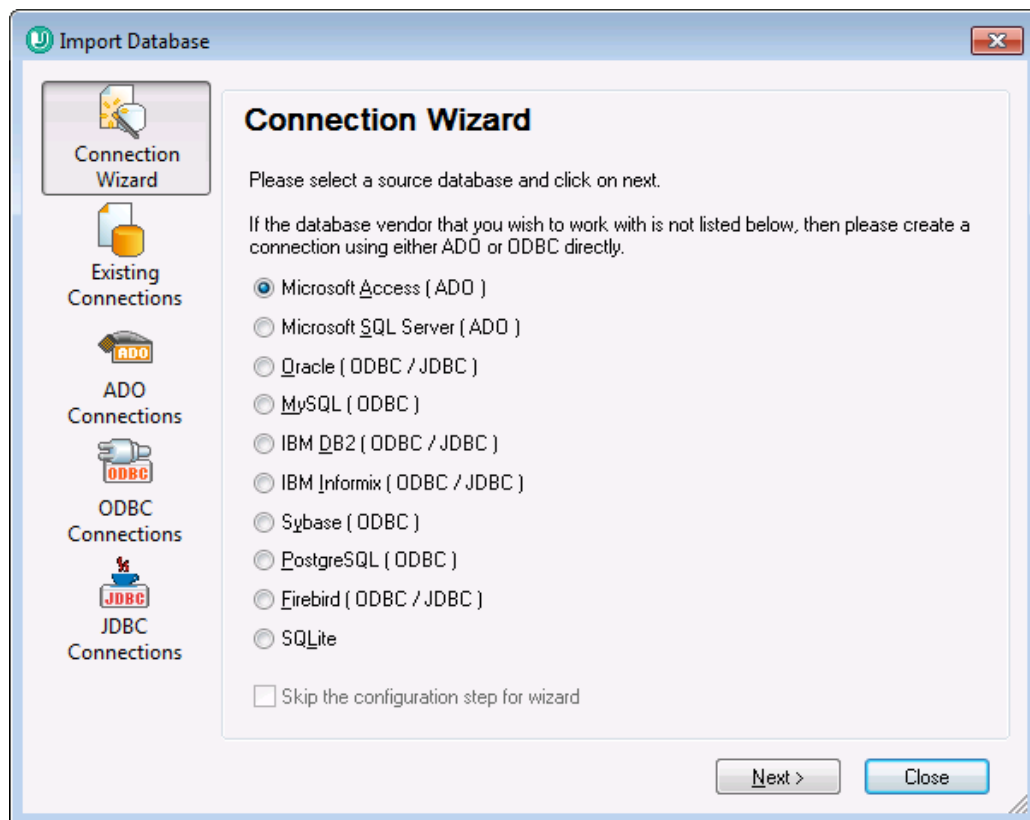
11.1.1 Starting the Database Connection Wizard

Whenever you take an action that requires a database connection, a wizard appears that guides you through the steps required to set up the connection.

Before you go through the wizard steps, be aware that for some database types it is necessary to install and configure separately several database prerequisites, such as a database driver or database client software. These are normally provided by the respective database vendors, and include documentation tailored to your specific Windows version. For a list of database drivers grouped by database type, see [Database Drivers Overview](#).

To start the database connection wizard:

1. On the **Project** menu, click **Import SQL Database**.
2. Click **New**.



After you select a database type and click **Next**, the on-screen instructions will depend on the database kind, technology (ADO, ODBC, JDBC) and driver used.

For examples applicable to each database type, see the [Examples](#) section. For instructions applicable to each database access technology, refer to the following topics:

- [Setting up an ADO Connection](#)
- [Setting up an ODBC Connection](#)
- [Setting up a JDBC Connection](#)

11.1.2 Database Drivers Overview

The following table lists common database drivers you can use to connect to a particular database through a particular data access technology. Note that this list does not aim to be either exhaustive or prescriptive; you can use other native or third party alternatives in addition to the drivers shown below.

Even though a number of database drivers are available by default on your Windows operating system, you may still want or need to download an alternative driver. For some databases, the latest driver supplied by the database vendor is likely to perform better than the driver that shipped with the operating system.

With some exceptions, most database vendors provide database client software which normally includes any required database drivers, or provide you with an option during installation to select the drivers and components you wish to install. Database client software typically consists of administration and configuration utilities used to simplify database administration and connectivity, as well as documentation on how to install and configure the database client and any of its components.

Configuring the database client correctly is crucial for establishing a successful connection to the database. If you have not installed your database client software yet, it is strongly recommended to read carefully the installation and configuration instructions of the database client, since they typically vary for each database version and for each Windows version.

Database	ODBC	JDBC	ADO
Firebird	Firebird ODBC driver (http:// www.firebirdsql.org/en/ odbc-driver/)	Firebird JDBC driver (http:// www.firebirdsql.org/en/ jdbc-driver/)	
IBM DB2	IBM DB2 ODBC Driver	IBM Data Server Driver for JDBC and SQLJ	IBM OLE DB Provider for DB2
IBM DB2 for i	iSeries Access ODBC Driver	IBM Toolbox for Java JDBC Driver	• IBM DB2 for i5/OS IBMDA400 OLE DB Provider • IBM DB2 for i5/OS IBMDARLA OLE DB Provider • IBM DB2 for i5/OS IBMDASQL OLE DB Provider
IBM Informix	IBM Informix ODBC Driver	IBM Informix JDBC Driver	IBM Informix OLE DB Provider
Microsoft Access	• Microsoft Access Driver		• Microsoft Jet OLE DB Provider • Microsoft Access Database Engine OLE DB Provider
Microsoft	• SQL Server Native	• Microsoft JDBC Driver	• Microsoft OLE DB

Database	ODBC	JDBC	ADO
SQL Server	Client	for SQL Server (http://msdn.microsoft.com/en-us/data/aa937724.aspx)	Provider for SQL Server • SQL Server Native Client
MySQL	Connector/ODBC (http://dev.mysql.com/downloads/connector/odbc/)	Connector/J (http://dev.mysql.com/downloads/connector/j/)	
Oracle	• Microsoft ODBC for Oracle • Oracle ODBC Driver (typically installed during the installation of your Oracle database client)	• JDBC Thin Driver • JDBC Oracle Call Interface (OCI) Driver These drivers are typically installed during the installation of your Oracle database client. Connect through the OCI Driver (not the Thin Driver) if you are using the Oracle XML DB component.	• Microsoft OLE DB Provider for Oracle
PostgreSQL	psqlODBC (https://odbc.postgresql.org/)	Postgre JDBC Driver (https://jdbc.postgresql.org/download.html)	
Sybase	Sybase ASE ODBC Driver	jConnect™ for JDBC	Sybase ASE OLE DB Provider

* The drivers highlighted in bold are Microsoft-supplied. If not already available on Windows system, they can be downloaded from the official Microsoft web site.

To understand the capabilities and limitations of each data access technology with respect to each database type, refer to the documentation of that particular database product and also test the connection against your specific environment. To avoid common connectivity issues, consider the following general notes and recommendations:

- Since 32-bit and 64-bit drivers may not be compatible, make sure to install and configure the driver version applicable to your Altova application. For example, if you are using a 32-bit Altova application on a 64-bit operating system, set up your database connection using the 32-bit driver version.
- The latest driver versions may provide features not available in older editions.
- When setting up an ODBC data source, it is generally recommended to create the data source name (DSN) as *System DSN* instead of *User DSN*.
- When setting up a JDBC data source, ensure that JRE (Java Runtime Environment) is installed and that the CLASSPATH environment variable of the operating system is configured.
- For the support details and known issues applicable to Microsoft-supplied database drivers, refer to the MSDN documentation.
- For the installation instructions and support details of any drivers or database client

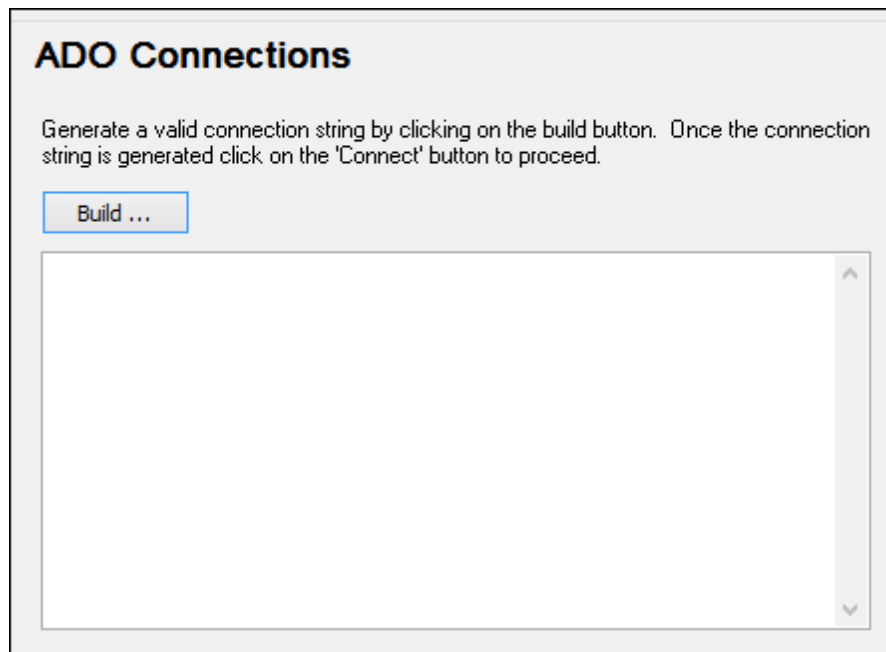
software that you install from a database vendor, check the documentation provided with the installation package. Whether you are using an official or third party database driver, the most comprehensive information and the configuration procedures applicable to that specific driver on your specific operating system is normally part of the driver installation package.

11.1.3 Setting up an ADO Connection

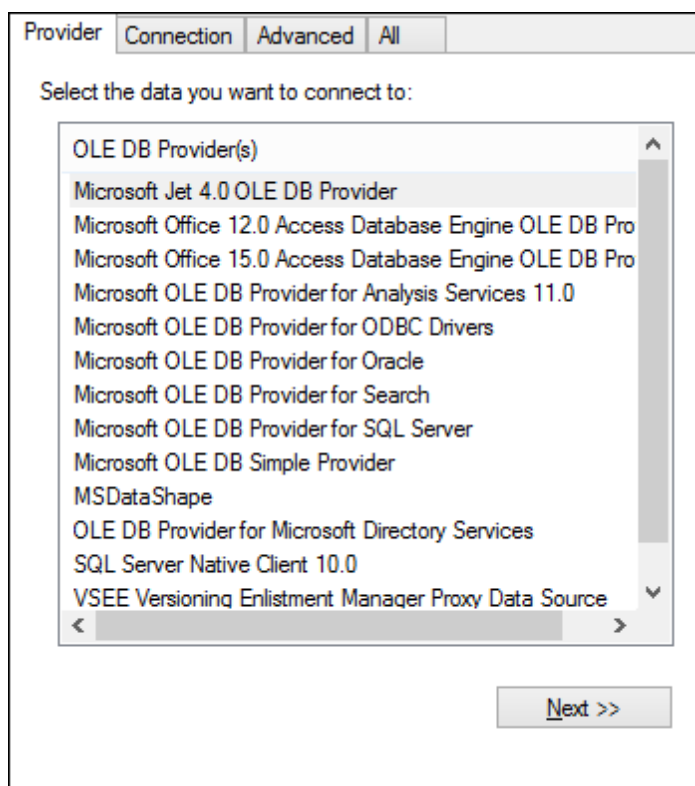
Microsoft ActiveX Data Objects (ADO) is a data access technology that enables you to connect to a variety of data sources through OLE DB. OLE DB is an alternative interface to ODBC or JDBC; it provides uniform access to data in a COM (Component Object Model) environment. ADO is the typical choice for connecting to Microsoft native databases such as Microsoft Access or SQL Server, although you can also use it for other data sources.

To set up an ADO connection:

1. [Start the database connection wizard.](#)
2. Click **ADO Connections**.



3. Click **Build**.



4. Select the data provider through which you want to connect. The table below lists a few common scenarios.

To connect to this database...	Use this provider...
Microsoft Access	• Microsoft Office Access Database Engine OLE DB Provider When connecting to Microsoft Access 2003, you can also use the Microsoft Jet OLE DB Provider.
SQL Server	• SQL Server Native Client • Microsoft OLE DB Provider for SQL Server
Other database	Select the provider applicable to your database. If an OLE DB provider to your database is not available, install the required driver from the database vendor (see Database Drivers Overview). Alternatively, set up an ODBC or JDBC connection. If the operating system has an ODBC driver to the required database, you can also use the Microsoft OLE DB Provider for ODBC Drivers.

5. Click **Next** and complete the wizard.

The subsequent wizard steps are specific to the provider you chose. For SQL Server, you will need to provide or select the host name of the database server, as well as the database username and password. For Microsoft Access, you will be asked to browse for or provide the path to the database file.

The complete list of initialization properties (connection parameters) is available in the **All** tab of the connection dialog box—these properties vary depending on the chosen provider. The following sections provide guidance on configuring the basic initialization properties for Microsoft Access and SQL Server databases:

- [Setting up the SQL Server Data Link Properties](#)
- [Setting up the Microsoft Access Data Link Properties](#)

Connecting to an Existing Microsoft Access Database

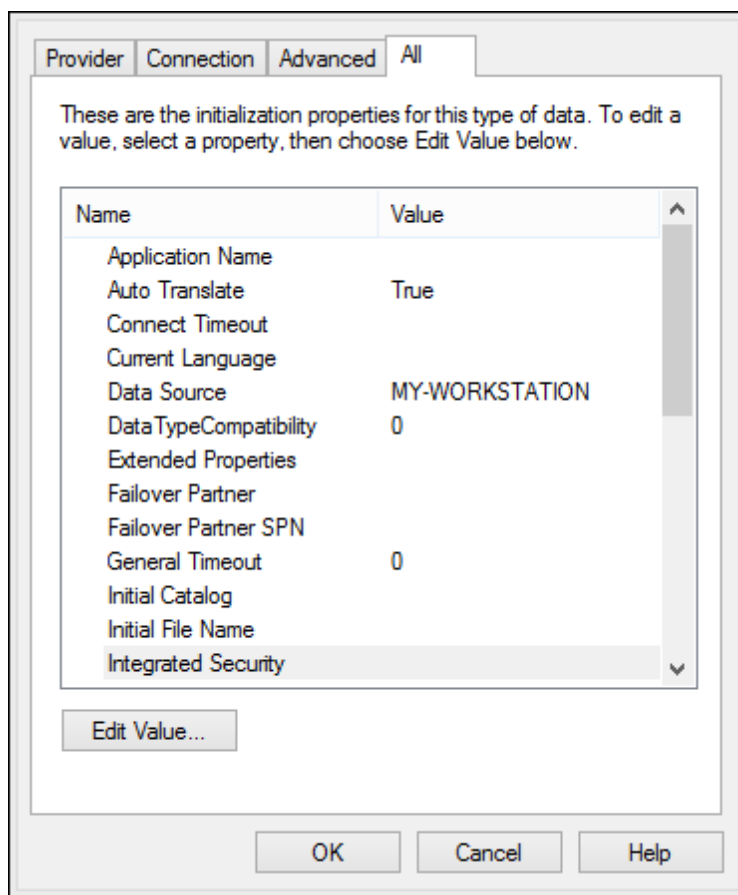
This approach is suitable when you want to connect to a Microsoft Access database which is not password-protected. If the database is password-protected, set up the database password as shown in [Connecting to Microsoft Access \(ADO\)](#).

To connect to an existing Microsoft Access database:

1. Run the database connection wizard (see [Starting the Database Connection Wizard](#)).
2. Select **Microsoft Access (ADO)**, and then click **Next**.
3. Browse for the database file, or enter the path to it (either relative or absolute) .
4. Click **Connect**.

Setting up the SQL Server Data Link Properties

When you connect to a Microsoft SQL Server database through ADO (see [Setting up an ADO Connection](#)), ensure that the following data link properties are configured correctly in the **All** tab of the Data Link Properties dialog box.

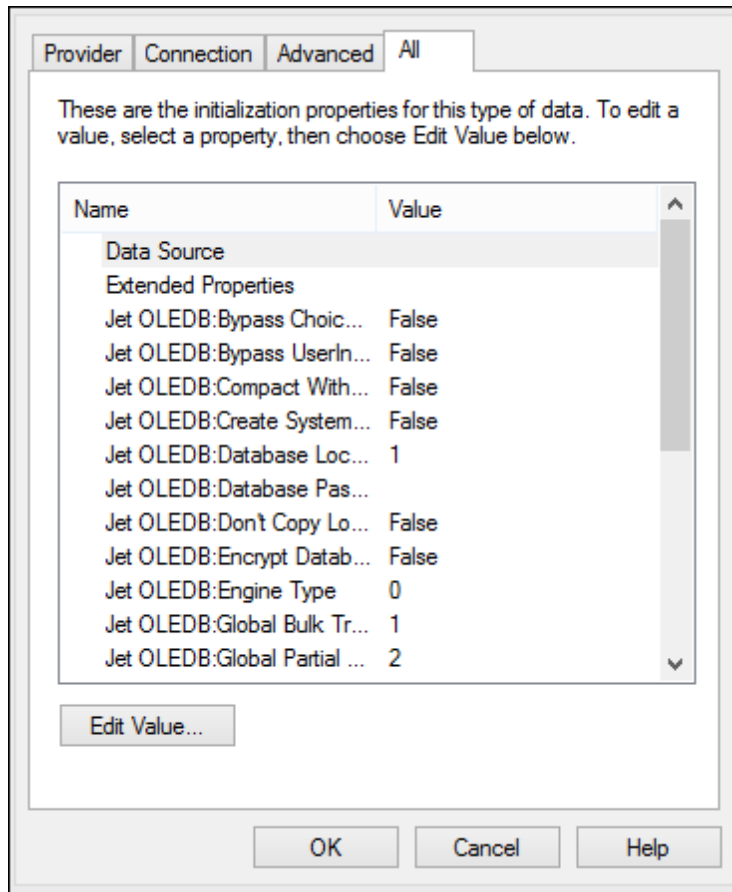


Data Link Properties dialog box

Property	Notes
Integrated Security	If you selected the SQL Server Native Client data provider on the Provider tab, set this property to a space character.
Persist Security Info	Set this property to True .

Setting up the Microsoft Access Data Link Properties

When you connect to a Microsoft Access database through ADO (see [Setting up an ADO Connection](#)), ensure that the following properties are configured correctly in the **All** tab of the Data Link Properties dialog box.



Data Link Properties dialog box

Property	Notes
Data Source	This property stores the path to the Microsoft Access database file. To avoid database connectivity issues, it is recommended to use the UNC (Universal Naming Convention) path format, for example: <pre>\\anyserver\share\$\filepath</pre>
Jet OLEDB:System Database	This property stores the path to the workgroup information file. You may need to explicitly set the value of this property before you can connect to a Microsoft Access database. If you cannot connect due to a "workgroup information file" error, locate the workgroup information file (System.MDW) applicable to your user profile (see http://support.microsoft.com/kb/305542 for instructions), and set the property value to the path of the System.MDW file.

Property	Notes
	Property Description Jet OLEDB:System database Property Value C:\Users\john.doe\AppData\Roaming\Microsoft\Access Reset Value OK Cancel
Jet OLEDB:Database Password	If the database is password-protected, set the value of this property to the database password. Property Description Jet OLEDB:Database Password Property Value •••••••• Reset Value OK Cancel

11.1.4 Setting up an ODBC Connection

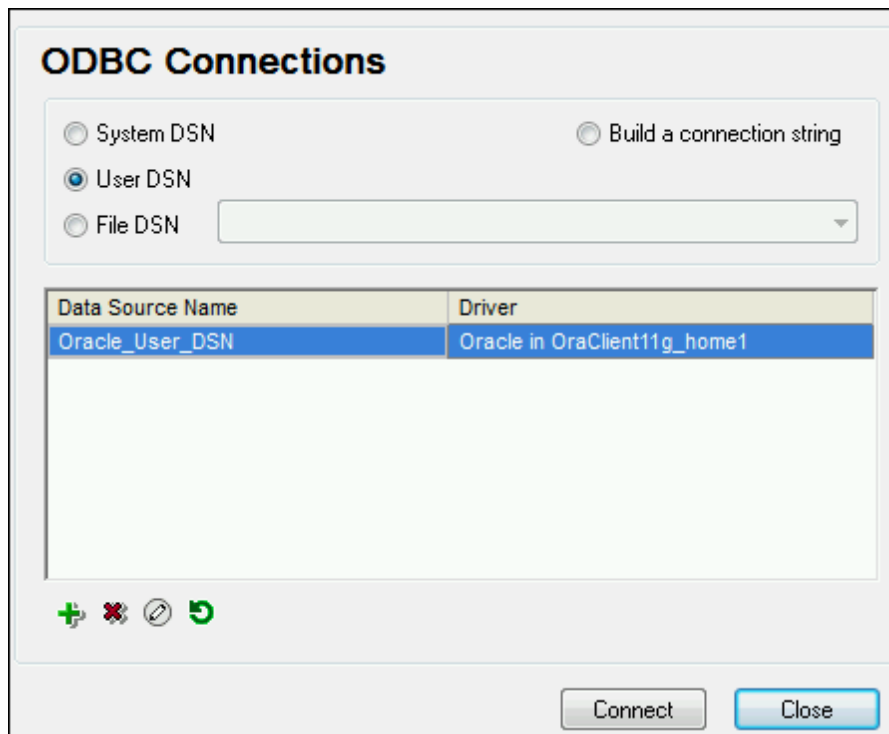
ODBC (Open Database Connectivity) is a widely used data access technology that enables you to connect to a database from UModel. It can be used either as primary means to connect to a database, or as an alternative to OLE DB- or JDBC-driven connections.

To connect to a database through ODBC, first you need to create an ODBC data source name (DSN) on the operating system. This step is not required if the DSN has already been created, perhaps by another user of the operating system. The DSN represents a uniform way to describe the database connection to any ODBC-aware client application on the operating system, including UModel. DSNs can be of the following types:

- System DSN
- User DSN
- File DSN

A *System* data source is accessible by all users with privileges on the operating system. A *User* data source is available to the user who created it. Finally, if you create a *File DSN*, the data source will be created as a file with the .dsn extension which you can share with other users, provided that they have installed the drivers used by the data source.

Any DSNs already available on your machine are listed by the database connection dialog box when you click **ODBC connections** on the ODBC connections dialog box.



ODBC Connections dialog box

If a DSN to the required database is not available, the UModel database connection wizard will assist you to create it; however, you can also create it directly on your Windows operating system. In either case, before you proceed, ensure that the ODBC driver applicable for your

database is in the list of ODBC drivers available to the operating system (see [Viewing the Available ODBC Drivers](#)).

To connect by using a new DSN:

1. [Start the database connection wizard](#).
2. On the database connection dialog box, click **ODBC Connections**.
3. Select a data source type (User DSN, System DSN, File DSN).

To create a System DSN, you need administrative rights on the operating system.

4. Click **Add** .
5. Select a driver, and then click **User DSN** or **System DSN** (depending on the type of the DSN you want to create). If the driver applicable to your database is not listed, download it from the database vendor and install it (see [Database Drivers Overview](#)).
6. On the dialog box that pops up, fill in any driver specific connection information to complete the setup.

For the connection to be successful, you will need to provide the host name (or IP address) of the database server, as well as the database username and password. There may be other optional connection parameters—these parameters vary between database providers. For detailed information about the parameters specific to each connection method, consult the documentation of the driver provider. Once created, the DSN becomes available in the list of data source names. This enables you to reuse the database connection details any time you want to connect to the database. Note that User DSNs are added to the list of User DSNs whereas System DSNs are added to the list of System DSNs.

To connect by using an existing DSN:

1. [Start the database connection wizard](#).
2. Click **ODBC Connections**.
3. Choose the type of the existing data source (User DSN, System DSN, File DSN).
4. Click the existing DSN record, and then click **Connect**.

To build a connection string based on an existing .dsn file:

1. [Start the database connection wizard](#).
2. Click **ODBC Connections**.
3. Select **Build a connection string**, and then click **Build**.
4. If you want to build the connection string using a File DSN, click the **File Data Source** tab. Otherwise, click the **Machine Data Source** tab. (System DSNs and User DSNs are known as "Machine" data sources.)
5. Select the required .dsn file, and then click **OK**.

To connect by using a prepared connection string:

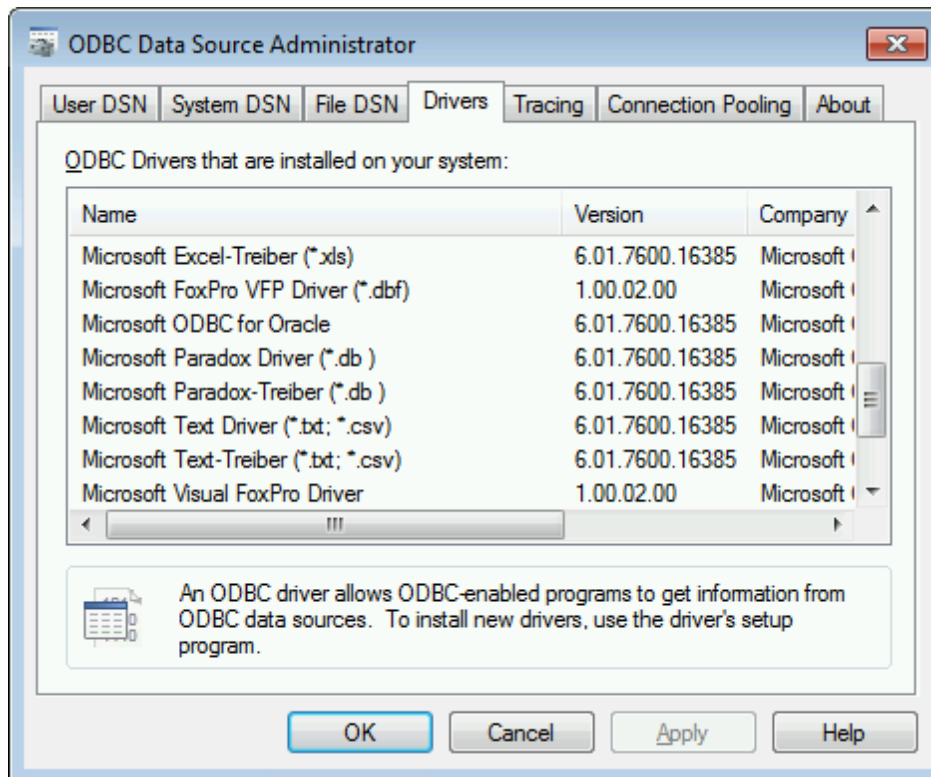
1. [Start the database connection wizard](#).
2. Click **ODBC Connections**.
3. Select **Build a connection string**.
4. Paste the connection string into the provided box, and then click **Connect**.

Viewing the Available ODBC Drivers

You can view the ODBC drivers available on your operating system in the ODBC Data Source Administrator. You can access the ODBC Data Source Administrator (**Odbcad32.exe**) from the Windows Control Panel, under **Administrative Tools**. On 64-bit operating systems, there are two versions of this executable:

- The 32-bit version of the **Odbcad32.exe** file is located in the **C:\Windows\SysWoW64** directory (assuming that **C:** is your system drive).
- The 64-bit version of the **Odbcad32.exe** file is located in the **C:\Windows\System32** directory.

Any installed 32-bit database drivers are visible in the 32-bit version of ODBC Data Source Administrator, while 64-bit drivers—in the 64-bit version. Therefore, ensure that you check the database drivers from the relevant version of ODBC Data Source Administrator.



ODBC Data Source Administrator

If the driver to your target database does not exist in the list, or if you want to add an alternative driver, you will need to download it from the database vendor (see [Database Drivers Overview](#)). Once the ODBC driver is available on your system, you are ready to create ODBC connections with it (see [Setting up an ODBC Connection](#)).

11.1.5 Setting up a JDBC Connection

JDBC (Java Database Connectivity) is a database access interface which is part of the Java software platform from Oracle. JDBC connections are generally more resource-intensive than ODBC connections but may provide features not available through ODBC. It is generally recommended to use a JDBC connection if you are using database features not available through an ODBC connector, for example, support for the XML DB technology in Oracle databases.

Prerequisites:

- JRE (Java Runtime Environment) or Java Development Kit (JDK) must be installed. If you have not installed it already, check the official Java website for the download package and installation instructions.
- The JDBC drivers from the database vendor must be installed. If you are connecting to an Oracle database, note that some Oracle drivers are specific to certain JRE versions and may require additional components and configuration. The documentation of your Oracle product (for example, the "Oracle Database JDBC Developer's Guide and Reference") includes detailed instructions about the configuration procedure for each JDBC driver.
- The operating system's `PATH` environment variable must include the path to the `bin` directory of the JRE or JDK installation directory, for example `C:\Program Files (x86)\Java\jre1.8.0_51\bin`.
- The `CLASSPATH` environment variable must include the path to the JDBC driver on your Windows operating system. When you install some database clients, the installer may configure this variable automatically. The documentation of the JDBC driver will typically include step-by-step instructions on setting the `CLASSPATH` variable (see also [Configuring the CLASSPATH](#)).

To set up a JDBC connection:

1. [Start the database connection wizard](#).
2. Click **JDBC Connections**.
3. Do one of the following:
 - a. Select a JDBC driver from the Driver list. This list contains any JDBC drivers configured through the `CLASSPATH` environment variable (see [Configuring the CLASSPATH](#)).
 - b. Enter a Java class name.
4. Enter the username and password to the database in the corresponding boxes.
5. In the Database URL text box, enter the JDBC connection string in the format specific to your database type (see the [JDBC connection formats](#) below).
6. Click **Connect**.

JDBC connection formats

The following table describes the syntax of JDBC connection strings for common database types.

Database	JDBC Connection Format
Firebird	<code>jdbc:firebirdsql://<host>[:<port>]/<database path or alias></code>
IBM DB2	<code>jdbc:db2://<hostName>:<port>/<databaseName></code>

Database	JDBC Connection Format
IBM Informix	<code>jdbc:informix-sqli://hostName:port/ databaseName:INFORMIXSERVER=myserver</code>
Microsoft SQL Server	<code>jdbc:sqlserver://hostName:port;databaseName=name</code>
MySQL	<code>jdbc:mysql://hostName:port/databaseName</code>
Oracle	<code>jdbc:oracle:thin:@//hostName:port:databaseName</code>
Oracle XML DB	<code>jdbc:oracle:oci:@//hostName:port:databaseName</code>
PostgreSQL	<code>jdbc:postgresql://hostName:port/databaseName</code>
Sybase	<code>jdbc:sybase:Tds:hostName:port/databaseName</code>

Note: Syntax variations to the formats listed above are also possible (for example, the database URL may exclude the port or may include the username and password to the database). Check the documentation of the database vendor for further details.

Configuring the CLASSPATH

The `CLASSPATH` environment variable is used by the Java Runtime Environment (JRE) to locate Java classes and other resource files on your operating system. When you connect to a database through JDBC, this variable must be configured to include the path to the JDBC driver on your operating system, and, in some cases, the path to additional library files specific to the database type you are using.

The following table lists sample file paths that must be typically included in the `CLASSPATH` variable. Importantly, you may need to adjust this information based on the location of the JDBC driver on your system, the JDBC driver name, as well as the JRE version present on your operating system. To avoid connectivity problems, check the installation instructions and any pre-installation or post-installation configuration steps applicable to the JDBC driver installed on your operating system.

Database	Sample CLASSPATH entries
Firebird	<code>C:\Program Files\Firebird\Jaybird-2.2.8-JDK_1.8\jaybird-full-2.2.8.jar</code>
IBM DB2	<code>C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc.jar;C:\Program Files (x86)\IBM\SQLLIB\java\db2jcc_license_cu.jar;</code>
IBM Informix	<code>C:\Informix_JDBC_Driver\lib\ifxjdbc.jar;</code>
Microsoft SQL Server	<code>C:\Program Files\Microsoft JDBC Driver 4.0 for SQL Server\sqljdbc_4.0\enu\sqljdbc.jar</code>
MySQL	<code>mysql-connector-java-version-bin.jar;</code>
Oracle	<code>ORACLE_HOME\jdbc\lib\ojdbc6.jar;</code>
Oracle (with XML DB)	<code>ORACLE_HOME\jdbc\lib\ojdbc6.jar;ORACLE_HOME\LIB</code>

Database	Sample CLASSPATH entries
	\xmlparserv2.jar;ORACLE_HOME\RDBMS\jlib\xdb.jar;
PostgreSQL	<installation directory>\postgresql.jar
Sybase	C:\sybase\jConnect-7_0\classes\jconn4.jar

- Changing the CLASSPATH variable may affect the behavior of Java applications on your machine. To understand possible implications before you proceed, refer to the Java documentation.
- Environment variables can be user or system. To change system environment variables, you need administrative rights on the operating system.
- After you change the environment variable, restart any running programs for settings to take effect. Alternatively, log off or restart your operating system.

To configure the CLASSPATH on Windows 7:

1. Open the **Start** menu and right-click **Computer**.
2. Click **Properties**.
3. Click **Advanced system settings**.
4. In the **Advanced** tab, click **Environment Variables**.
5. Locate the CLASSPATH variable under user or system environment variables, and then click **Edit**. If the CLASSPATH variable does not exist, click **New** to create it.
6. Edit the variable value to include the path on your operating system where the JDBC driver is located. To separate the JDBC driver path from other paths that may already be in the CLASSPATH variable, use the semi-colon separator (;).

To configure the CLASSPATH on Windows 8:

1. Right-click the Windows Start button, and then click **System**.
2. Click **Advanced System Settings**.
3. Click **Environment Variables**.
4. Locate the CLASSPATH variable under user or system environment variables, and then click **Edit**. If the CLASSPATH variable does not exist, click **New** to create it.
5. Edit the variable value to include the path on your operating system where the JDBC driver is located. To separate the JDBC driver path from other paths that may already be in the CLASSPATH variable, use the semi-colon separator (;).

11.1.6 Setting up a SQLite Connection

SQLite (<http://www.sqlite.org>) is a file-based, self-contained database type, which makes it ideal in scenarios where portability and ease of configuration is important. Since SQLite databases are natively supported by UModel, you do not need to install any drivers to connect to them.

Connecting to an Existing SQLite Database

To connect to an existing SQLite database:

1. Run the database connection wizard (see [Starting the Database Connection Wizard](#)).
2. Select **SQLite**, and then click **Next**.
3. Browse for the SQLite database file, or enter the path (either relative or absolute) to the database. The **Connect** button becomes enabled once you enter the path to a SQLite database file.
4. Click **Connect**.

11.1.7 Examples

This section includes sample procedures for connecting to a database from UModel. Note that your Windows machine, the network environment, and the database client or server software is likely to have a configuration that is not exactly the same as the one presented in the following examples.

Note: For most database types, it is possible to connect using more than one data access technology (ADO, ODBC, JDBC) or driver. The performance of the database connection, as well as its features and limitations will depend on the selected driver, database client software (if applicable), and any additional connectivity parameters that you may have configured outside UModel.

This section includes the following topics:

- [Connecting to Firebird \(ODBC\)](#)
- [Connecting to Firebird \(JDBC\)](#)
- [Connecting to IBM DB2 \(ODBC\)](#)
- [Connecting to IBM DB2 for i \(ODBC\)](#)
- [Connecting to IBM Informix \(JDBC\)](#)
- [Connecting to Microsoft Access \(ADO\)](#)
- [Connecting to Microsoft SQL Server \(ADO\)](#)
- [Connecting to Microsoft SQL Server \(ODBC\)](#)
- [Connecting to MySQL \(ODBC\)](#)
- [Connecting to Oracle \(ODBC\)](#)
- [Connecting to PostgreSQL \(ODBC\)](#)
- [Connecting to Sybase \(JDBC\)](#)

Connecting to Firebird (ODBC)

This topic provides sample instructions for connecting to a Firebird 2.5.4 database running on a Linux server.

Prerequisites:

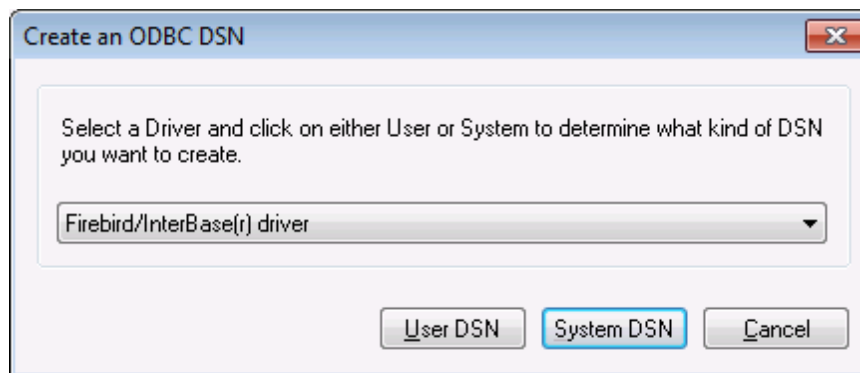
- The Firebird database server is configured to accept TCP/IP connections from clients.
- The Firebird ODBC driver must be installed on your operating system. This example uses the Firebird ODBC driver version 2.0.3.154 downloaded from the Firebird website (<http://www.firebirdsql.org/>).
- The Firebird client must be installed on your operating system. Note that there is no standalone installer available for the Firebird 2.5.4 client; the client is part of the Firebird server installation package. You can download the Firebird server installation package from the Firebird website (<http://www.firebirdsql.org/>), look for "Windows executable installer for full Superclassic/Classic or Superserver". To install only the client files, choose "**Minimum client install - no server, no tools**" when going through the wizard steps.

Important:

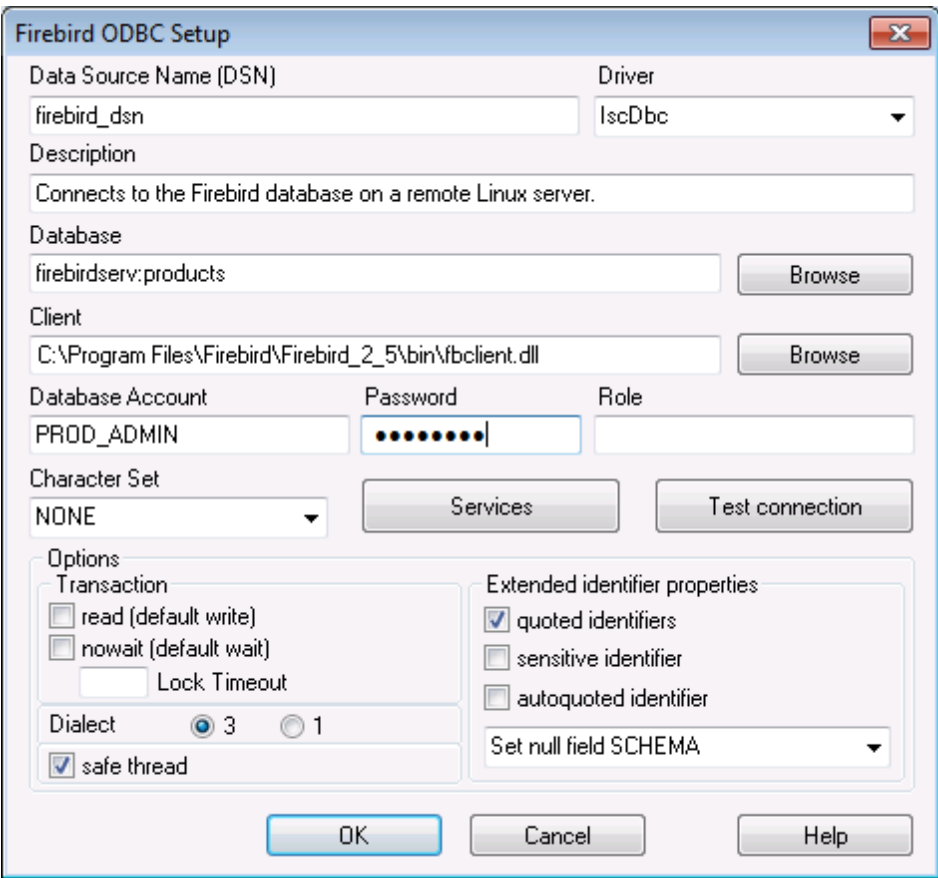
- The platform of both the Firebird ODBC driver and client (32-bit or 64-bit) must correspond to that of UModel.
 - The version of the Firebird client must correspond to the version of Firebird server to which you are connecting.
- You have the following database connection details: server host name or IP address, database path (or alias) on the server, user name, and password.

To connect to Firebird via ODBC:

1. [Start the database connection wizard.](#)
2. Click **ODBC Connections**.
3. Select **User DSN** (or **System DSN**, if you have administrative privileges), and then click **Add** .



4. Select the Firebird driver, and then click **User DSN** (or **System DSN**, depending on what you selected in the previous step). If the Firebird driver is not available in the list, make sure that it is installed on your operating system (see also [Viewing the Available ODBC Drivers](#)).



5. Enter the database connection details as follows:

<i>Data Source Name (DSN)</i>	Enter a descriptive name for the data source you are creating.
<i>Database</i>	Enter the server host name or IP address, followed by a colon, followed by the database alias (or path). In this example, the host name is <code>firebirdserv</code>, and the database alias is <code>products</code>, as follows: <code>firebirdserv:products</code> Using a database alias assumes that, on the server side, the database administrator has configured the alias <i>products</i> to point to the actual Firebird (.fdb) database file on the server (see the Firebird documentation for more details). You can also use the server IP address instead of the host name, and a path instead of an alias; therefore, any of the following sample connection strings are valid: <code>firebirdserver:/var/Firebird/databases/ butterflies.fdb</code>

	127.0.0.1:D:\Misc\Lenders.fdb If the database is on the local Windows machine, click Browse and select the Firebird (.fdb) database file directly.
<i>Client</i>	Enter the path to the fbclient.dll file. By default, this is the <code>bin</code> subdirectory of the Firebird installation directory.
<i>Database Account</i>	Enter the database user name supplied by the database administrator (in this example, <code>PROD_ADMIN</code>).
<i>Password</i>	Enter the database password supplied by the database administrator.

6. Click **OK**.

Connecting to Firebird (JDBC)

This topic provides sample instructions for connecting to a Firebird database server through JDBC.

Prerequisites:

- Java Runtime Environment (JRE) or Java Development Kit (JDK) must be installed on your operating system.
- The operating system's `PATH` environment variable must include the path to the `bin` directory of the JRE or JDK installation directory, for example `C:\Program Files (x86)\Java\jre1.8.0_51\bin`.
- The Firebird JDBC driver must be installed on your operating system. This example uses *Jaybird 2.2.8* downloaded from the Firebird website (<http://www.firebirdsql.org/>).
- The operating system's `CLASSPATH` environment variable must include the path to the Jaybird driver, for example `C:\jdbc\firebird\jaybird-full-2.2.8.jar`. See also [Configuring the CLASSPATH](#).
- You have the following database connection details: host, database path or alias, username, and password.

To connect to Firebird through JDBC:

1. [Start the database connection wizard](#).
2. Click **JDBC Connections**.
3. In the Driver box, select **org.firebirdsql.jdbc.FBDriver**. If the entry is not available, check if the `CLASSPATH` and `PATH` environment variables are set correctly (see the prerequisites above).

JDBC Connections

Enter a connection string and select (or enter manually) a valid JDBC driver. Click on 'Connect' to proceed.

Driver:

Username:

Password:

Database URL:

4. Enter the username and password to the database in the corresponding text boxes.
5. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:firebirdsql://<host>[:<port>]/<database path or alias>
```

6. Click **Connect**.

Connecting to IBM DB2 (ODBC)

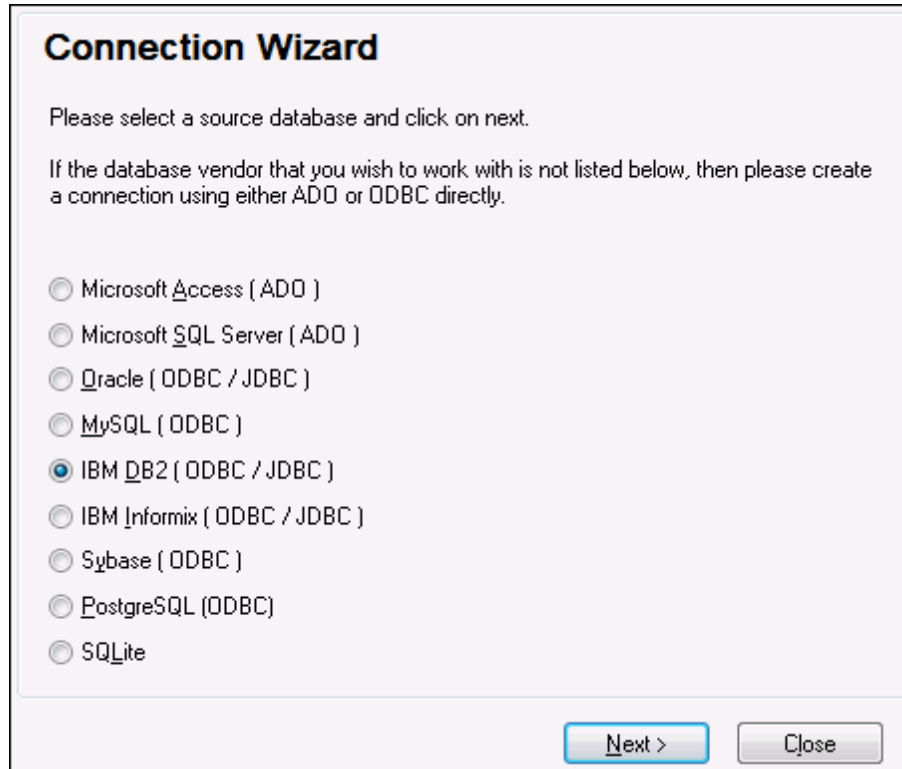
This topic provides sample instructions for connecting to an IBM DB2 database through ODBC.

Prerequisites:

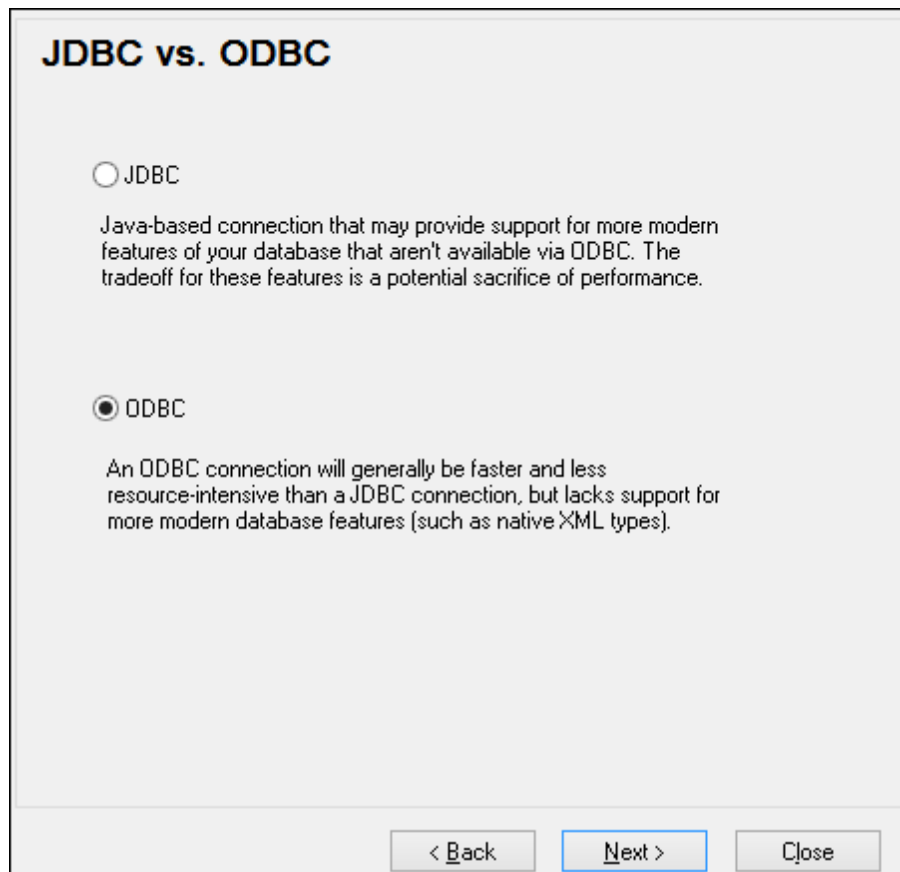
- IBM Data Server Client must be installed and configured on your operating system (this example uses IBM Data Server Client 9.7). For installation instructions, check the documentation supplied with your IBM DB2 software. After installing the IBM Data Server Client, check if the ODBC drivers are available on your machine (see [Viewing the Available ODBC Drivers](#)).
- Create a database alias. There are several ways to do this:
 - From IBM DB2 Configuration Assistant
 - From IBM DB2 Command Line Processor
 - From the ODBC data source wizard (for this case, the instructions are shown below)
- You have the following database connection details: host, database, port, username, and password.

To connect to IBM DB2:

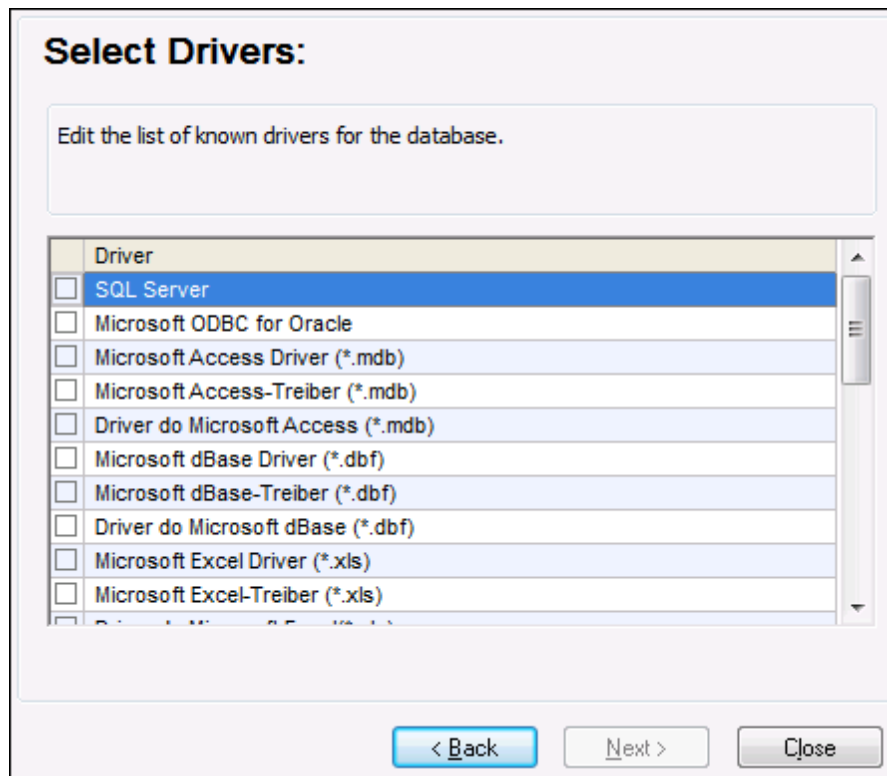
1. [Start the database connection wizard](#) and select **IBM DB2 (ODBC/JDBC)**.



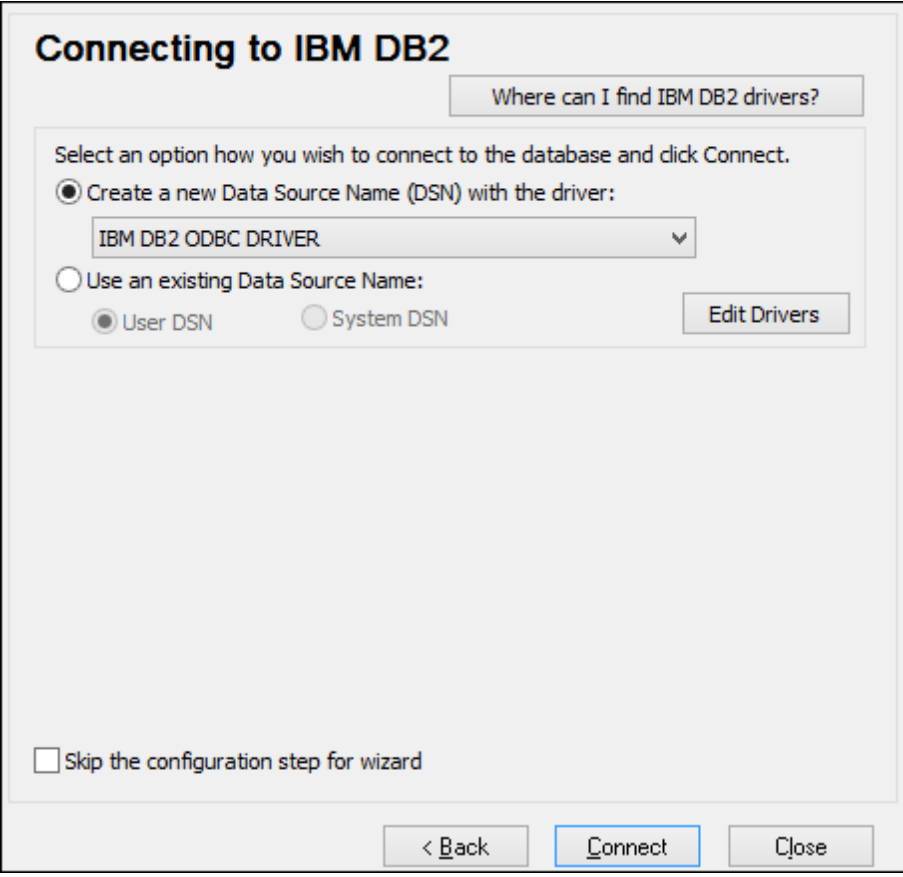
2. Click **Next**.



3. Select **ODBC**, and click **Next**. If prompted to edit the list of known drivers for the database, select the database drivers applicable to IBM DB2 (see [Prerequisites](#)), and click **Next**.



4. Select the IBM DB2 driver from the list, and then click **Connect**. (To edit the list of available drivers, click **Edit Drivers**, and then check or uncheck the IBM DB2 drivers you wish to add or remove, respectively.)



Connecting to IBM DB2

Where can I find IBM DB2 drivers?

Select an option how you wish to connect to the database and click Connect.

☒ Create a new Data Source Name (DSN) with the driver:

IBM DB2 ODBC DRIVER

☐ Use an existing Data Source Name:

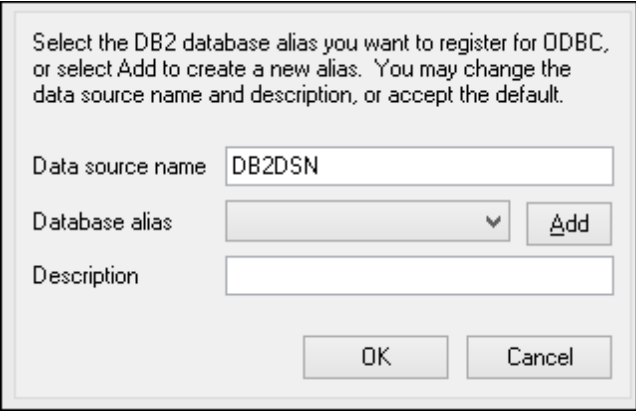
☒ User DSN ☐ System DSN

Edit Drivers

☐ Skip the configuration step for wizard

< Back Connect Close

5. Enter a data source name (in this example, **DB2DSN**), and then click **Add**.



Select the DB2 database alias you want to register for ODBC, or select Add to create a new alias. You may change the data source name and description, or accept the default.

Data source name DB2DSN

Database alias Add

Description

OK Cancel

6. On the **Data Source** tab, enter the user name and password to the database.

The screenshot shows a dialog box titled 'Data Source' with four tabs: 'Data Source', 'TCP/IP', 'Security options', and 'Advanced Settings'. The 'TCP/IP' tab is selected. It contains the following fields and controls:

- 'Data source name' text box with the value 'DB2DSN'.
- 'Description' text box, currently empty.
- A group box containing:
 - 'User ID' text box with the value 'john_doe'.
 - 'Password' text box with masked characters (dots).
 - 'Save password' checkbox, which is unchecked.

At the bottom of the dialog are four buttons: 'OK', 'Cancel', 'Apply', and 'Help'.

7. On the **TCP/IP** tab, enter the database name, a name for the alias, the host name and the port number, and then click OK.

This screenshot shows the same 'Data Source' dialog box with the 'TCP/IP' tab selected. The fields are populated with the following values:

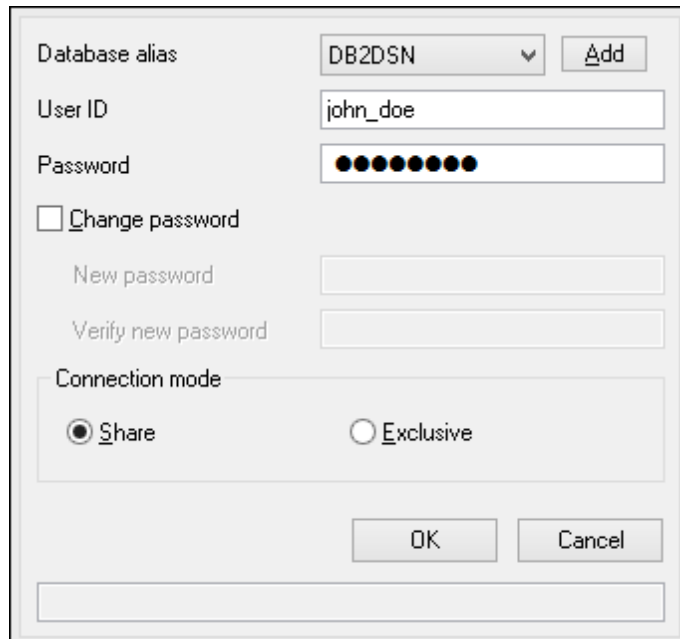
- 'Database name' text box: 'database1'.
- 'Database alias' text box: 'alias1'.
- 'Host name' text box: 'host1'.
- 'Port number' text box: '50000'.

Below these fields are the following options:

- An unchecked checkbox: 'The database physically resides on a host or Q/S/400 system.'
- Two radio buttons:
 - 'Connect directly to the server' (selected).
 - 'Connect to the server via the gateway' (unchecked).
- An unchecked checkbox: 'DCS Parameters'.
- A text box containing '„INTERRUPT_ENABLED.....' (disabled).
- 'Optimize for application' dropdown menu (set to a default value).

The 'OK', 'Cancel', 'Apply', and 'Help' buttons are at the bottom.

8. Enter again the username and password, and then click **OK**.



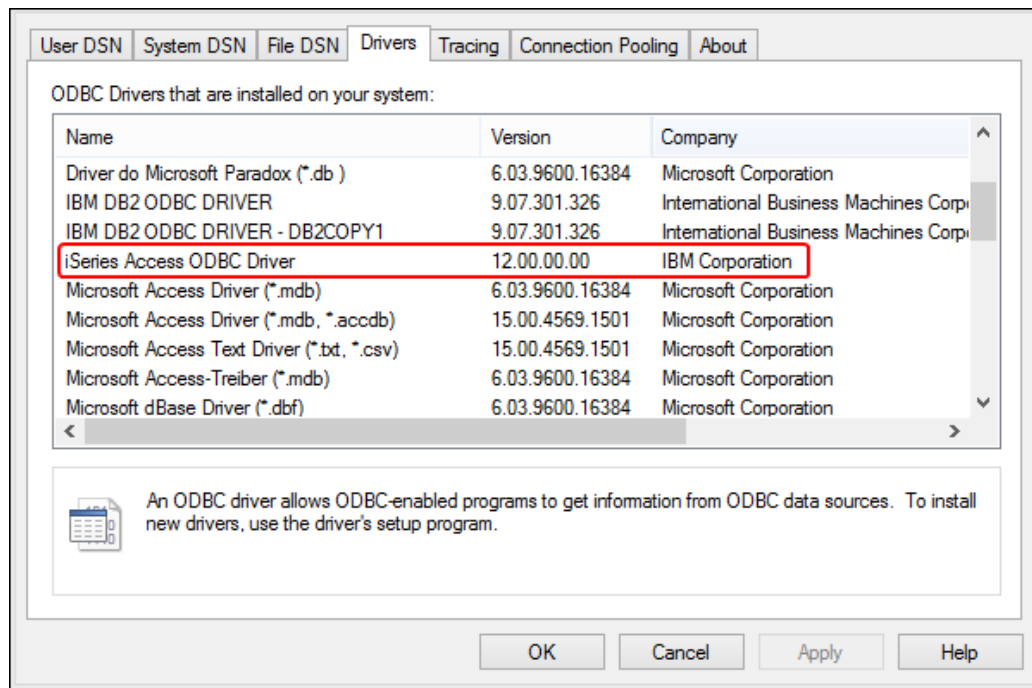
The screenshot shows a dialog box for configuring an ODBC connection to IBM DB2 for i. It includes fields for 'Database alias' (set to 'DB2DSN'), 'User ID' (set to 'john_doe'), and 'Password' (masked with dots). There is a checkbox for 'Change password' and fields for 'New password' and 'Verify new password'. The 'Connection mode' section has two radio buttons: 'Share' (selected) and 'Exclusive'. At the bottom are 'OK' and 'Cancel' buttons.

Connecting to IBM DB2 for i (ODBC)

This topic provides sample instructions for connecting to an *IBM DB2 for i* database through ODBC.

Prerequisites:

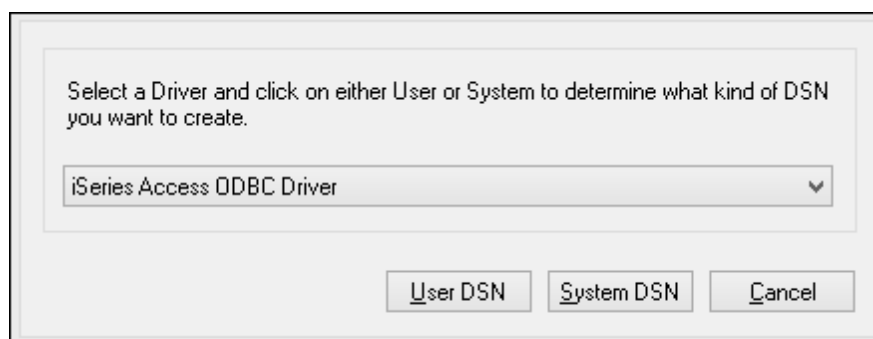
- *IBM System i Access for Windows* must be installed on your operating system (this example uses *IBM System i Access for Windows V6R1M0*). For installation instructions, check the documentation supplied with your *IBM DB2 for i* software. After installation, check if the ODBC driver is available on your machine (see [Viewing the Available ODBC Drivers](#)).



- You have the following database connection details: the I.P. address of the database server, database user name, and password.
- Run *System i Navigator* and follow the wizard to create a new connection. When prompted to specify a system, enter the I.P. address of the database server. After creating the connection, it is recommended to verify it (click on the connection, and select **File > Diagnostics > Verify Connection**). If you get connectivity errors, contact the database server administrator.

To connect to IBM DB2 for i:

1. [Start the database connection wizard](#).
2. Click **ODBC connections**.
3. Click **User DSN** (alternatively, click **System DSN**, or **File DSN**, in which case the subsequent instructions will be similar).
4. Click **Add** .
5. Select the **iSeries Access ODBC Driver** from the list, and click **User DSN** (or **System DSN**, if applicable).



6. Enter a data source name and select the connection from the System combo box. In this

example, the data source name is **iSeriesDSN** and the System is **192.0.2.0**.

The screenshot shows a dialog box with the 'General' tab selected. It contains the following fields and controls:

- Data source name:** A text box containing 'iSeriesDSN'.
- Description:** A text box containing 'System i Access for Windows ODBC data source'.
- System:** A dropdown menu currently showing '192.0.2.0'.
- Connection Options...:** A button located to the right of the System dropdown.
- Buttons:** 'OK', 'Cancel', 'Apply', and 'Help' buttons are located at the bottom of the dialog.

7. Click **Connection Options**, select **Use the User ID specified below** and enter the name of the database user (in this example, **DBUSER**).

Default user ID

☐ Use Windows user name

☒ Use the user ID specified below

DBUSER

☐ None

☐ Use System i Navigator default

☐ Use Kerberos principal

Signon dialog prompting

☒ Prompt for SQLConnect if needed

☐ Never prompt for SQLConnect

Security

☐ Do not use Secured Sockets Layer (SSL)

☐ Use Secured Sockets Layer (SSL)

☒ Use same security as System i Navigator connection

OK Cancel Help

8. Click **OK**. The new data source becomes available in the list of DSNs.
9. Click **Connect**.
10. Enter the user name and password to the database when prompted, and then click **OK**.

Connecting to IBM Informix (JDBC)

This topic provides sample instructions for connecting to an IBM Informix database server through JDBC.

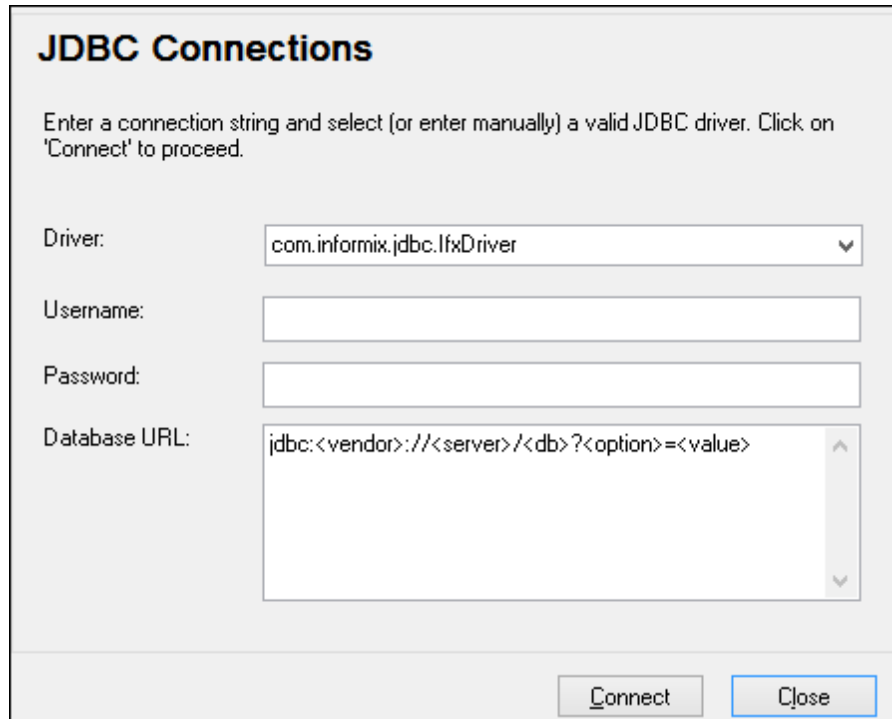
Prerequisites:

- Java Runtime Environment (JRE) must be installed on your operating system.
- The JDBC driver must be installed on your operating system (in this example, IBM Informix JDBC driver version 3.70 is used). For the driver's installation instructions, see the documentation accompanying the driver or the "IBM Informix JDBC Driver Programmer's Guide").
- The operating system's `CLASSPATH` environment variable includes the path where the Informix JDBC driver (`ifxjdbc.jar`) was installed. In this example, the Informix JDBC driver is installed in the directory `C:\Informix_JDBC_Driver`, and the value of `CLASSPATH` variable is `C:\Informix_JDBC_Driver\lib\ifxjdbc.jar`. For more information, see [Configuring the CLASSPATH](#).
- You have the following database connection details: host, name of the Informix server, database, port, username, and password.

To connect to IBM Informix through JDBC:

1. [Start the database connection wizard](#).

2. Click **JDBC Connections**.
3. Select the Informix JDBC driver from the list of available JDBC drivers (in this example, **com.informix.jdbc.IfxDriver**). If the list does not contain an Informix driver, it is either not installed correctly, or not included in the `CLASSPATH` variable (see the list of prerequisites above).



4. Enter the username and password to the database in the corresponding text boxes.
5. Enter the connection string to the database server in the Database URL text box, by replacing the highlighted values with the ones applicable to your database server.

```
jdbc:informix-sqli://hostName:port/  
databaseName:INFORMIXSERVER=myserver;
```

6. Click **Connect**.

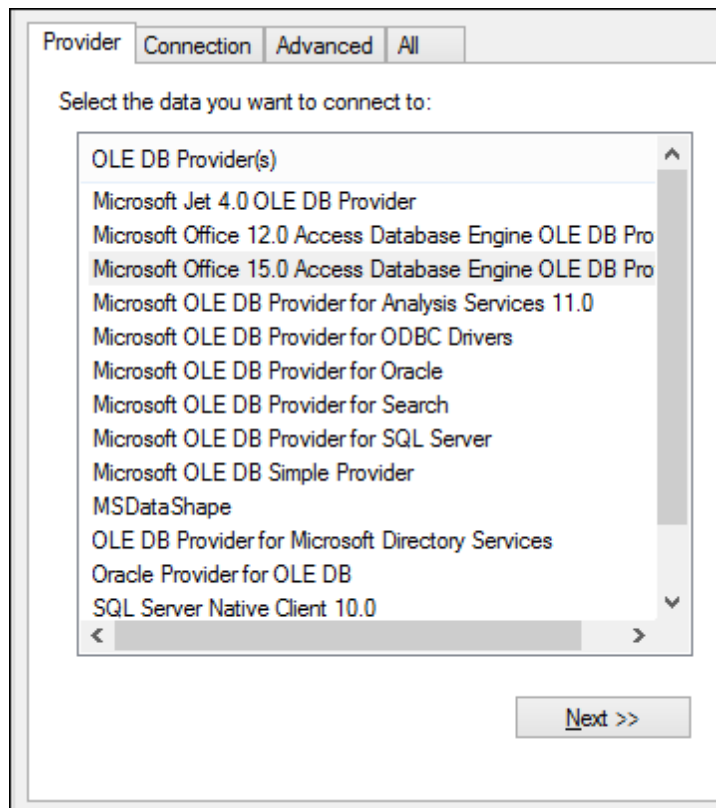
Connecting to Microsoft Access (ADO)

A simple way to connect to a Microsoft Access database is to follow the wizard and browse for the database file, as shown in [Connecting to an Existing Microsoft Access Database](#). An alternative approach is to set up an ADO connection explicitly, as shown in this topic. This approach is useful if your database is password-protected.

It is also possible to connect to Microsoft Access through an ODBC connection, but there are some limitations in this scenario, so it is best to avoid it.

To connect to a password-protected Microsoft Access database:

1. [Start the database connection wizard](#).
2. Click **ADO Connections**.
3. Click **Build**.



4. Select the **Microsoft Office 15.0 Access Database Engine OLE DB Provider**, and then click **Next**.

Provider Connection Advanced All

Specify the following to connect to this data:

- Enter the data source and/or location of the data:

Data Source:

Location:
- Enter information to log on to the server:

☐ Use Windows NT Integrated security

☒ Use a specific user name and password:

User name:

Password:

☒ Blank password ☐ Allow saving password
- Enter the initial catalog to use:

Test Connection

- In the Data Source box, enter the path to the Microsoft Access file. Because the file is on the local network share **U:\Departments\Finance\Reports\Revenue.accdb**, we will convert it to UNC format, and namely **\\server1\dfs\Departments\Finance\Reports\Revenue.accdb**, where **server1** is the name of the server and **dfs** is the name of the network share.
- On the **All** tab, double click the **Jet OLEDB:Database Password** property and enter the database password as property value.

Property Description

Jet OLEDB:Database Password

Property Value

.....

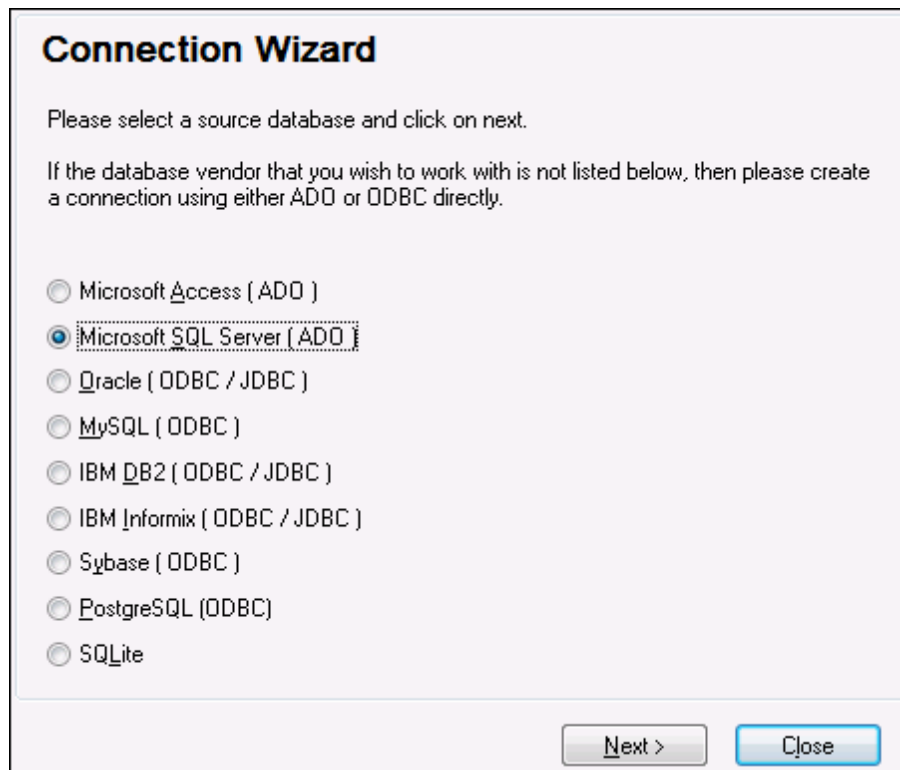
Reset Value OK Cancel

Note: If you are still unable to connect, locate the workgroup information file (**System.MDW**) applicable to your user profile (see <http://support.microsoft.com/kb/305542> for instructions), and set the value of the **Jet OLEDB: System database** property to the path of the **System.MDW** file.

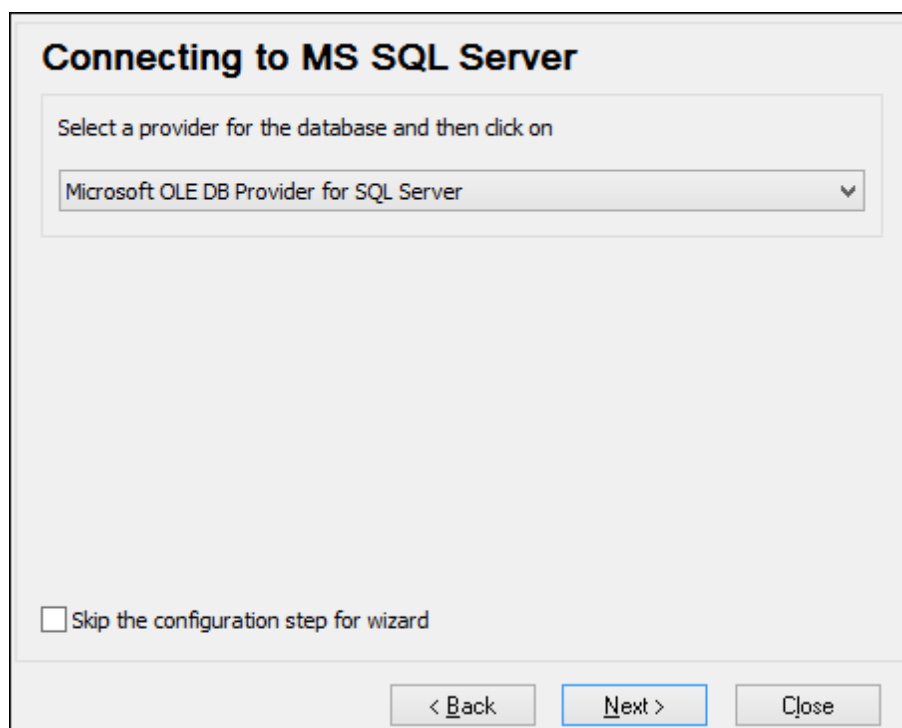
Connecting to Microsoft SQL Server (ADO)

To connect to SQL Server using the Microsoft OLE DB Provider:

1. [Start the database connection wizard.](#)



2. Select **Microsoft SQL Server (ADO)**, and then click **Next**. The list of available ADO drivers is displayed.



3. Select **Microsoft OLE DB Provider for SQL Server**, and then click **Next**.

4. Select or enter the name of the database server (in this example, **SQLSERV01**). To view the list of all servers on the network, expand the drop-down list.
5. If the database server was configured to allow connections from users authenticated on the Windows domain, select **Use Windows NT integrated security**. Otherwise, select **Use a specific user name and password**, and type them in the relevant boxes.
6. Select the database to which you are connecting (in this example, **NORTHWIND**).
7. To test the connection at this time, click **Test Connection**. This is an optional, recommended step.
8. Do one of the following:
 - a. Select the **Allow saving password** check box.
 - b. On the **All** tab, change the value of the **Persist Security Info** property to **True**.

Provider Connection Advanced All

Specify the following to connect to SQL Server data:

1. Select or enter a server name:
SQLSERV01 Refresh

2. Enter information to log on to the server:
☐ Use Windows NT Integrated security
☒ Use a specific user name and password:
User name: john_doe
Password: ••••
☐ Blank password ☒ Allow saving password

3. ☒ Select the database on the server:
NORTHWIND
☐ Attach a database file as a database name:
Using the filename: ...

Test Connection

OK Cancel Help

9. Click **OK**.

Connecting to Microsoft SQL Server (ODBC)

To connect to SQL Server using ODBC:

1. [Start the database connection wizard](#).
2. Click **ODBC Connections**.
3. Select **User DSN** (or **System DSN**, if you have administrative privileges), and then click **Add** .

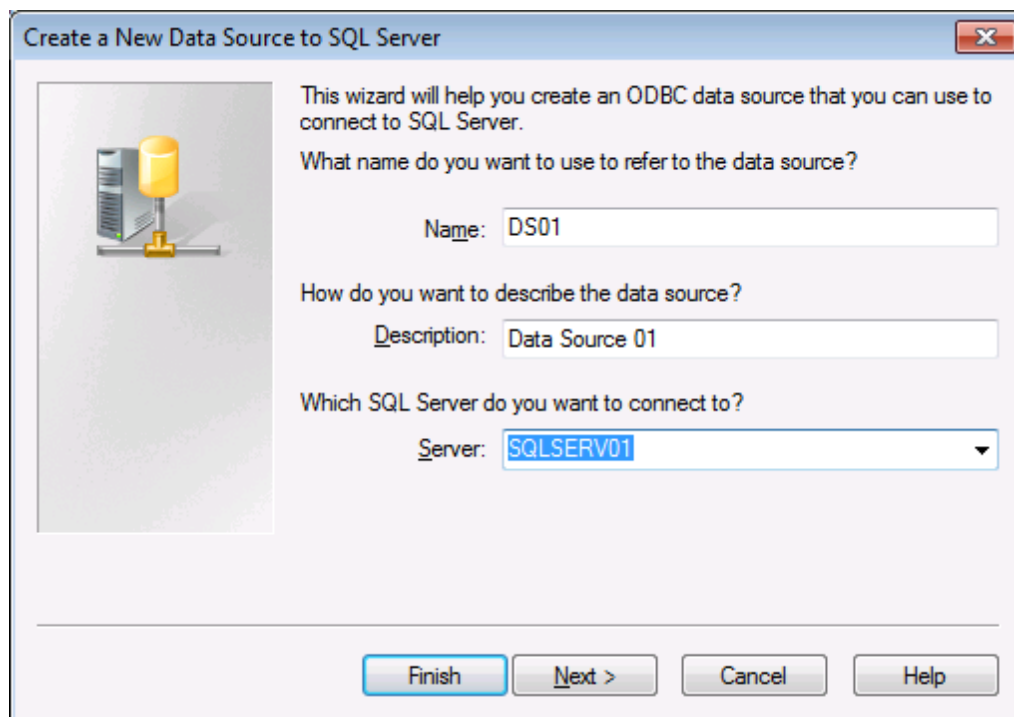
Create an ODBC DSN

Select a Driver and click on either User or System to determine what kind of DSN you want to create.

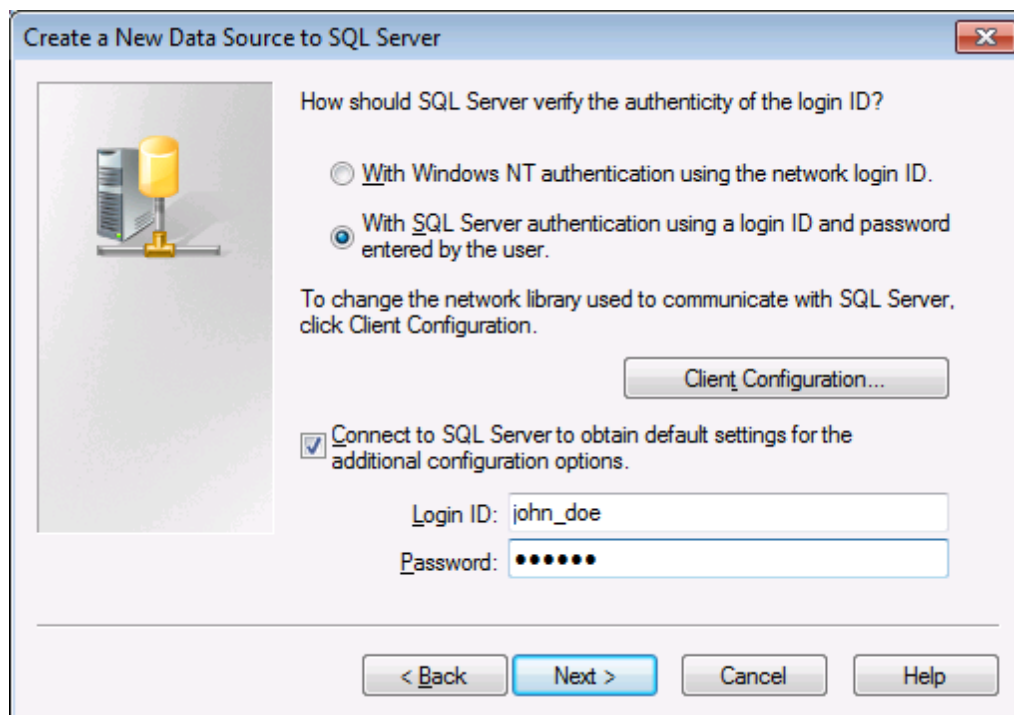
SQL Server

User DSN System DSN Cancel

4. Select **SQL Server** (or **SQL Server Native Client**, if available), and then click **User DSN** (or **System DSN** if you are creating a System DSN).

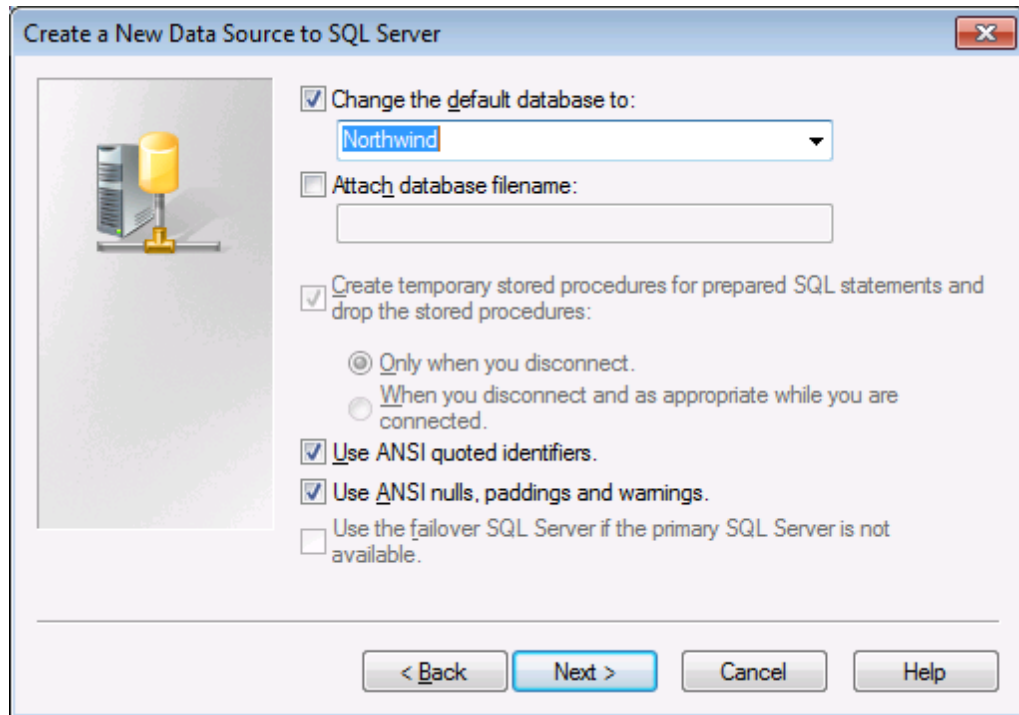


5. Enter a name and description to identify this connection, and then select from the list the SQL Server to which you are connecting (**SQLSERV01** in this example).



6. If the database server was configured to allow connections from users authenticated on the Windows domain, select **With Windows NT authentication**. Otherwise, select **With SQL Server authentication...** and type the user name and password in the relevant

boxes.



7. Select the name of the database to which you are connecting (in this example, **Northwind**).
8. Click **Finish**.

Connecting to MySQL (ODBC)

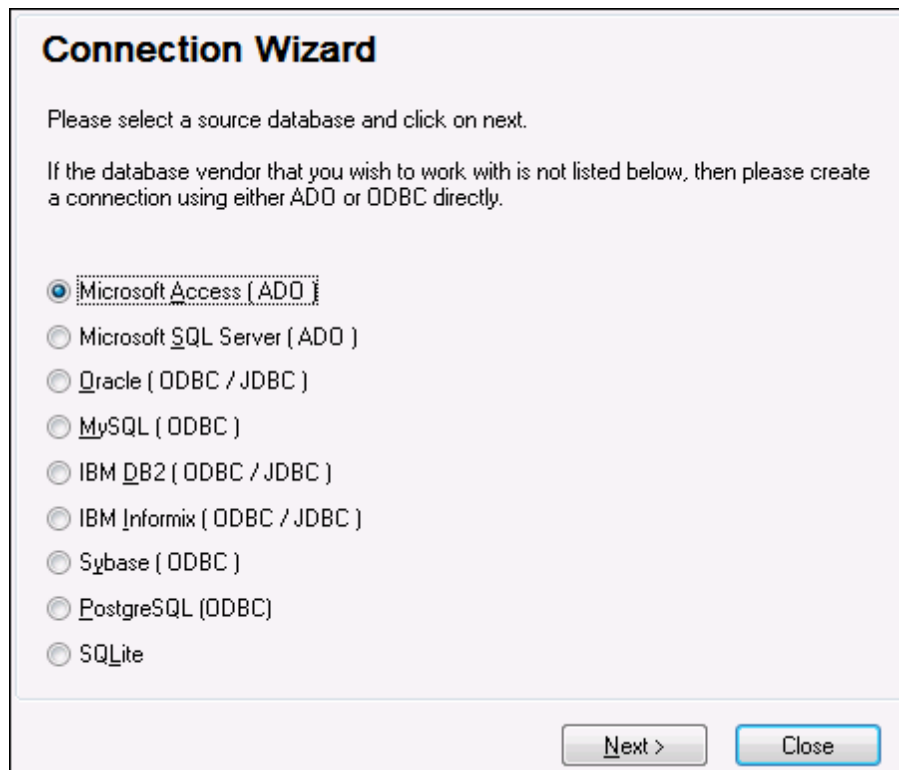
This topic provides sample instructions for connecting to a MySQL database server from a Windows machine through the ODBC driver. The MySQL ODBC driver is not available on Windows, so it must be downloaded and installed separately. This example uses MySQL ODBC driver version 5.3.4 downloaded from the official website (see also [Database Drivers Overview](#)).

Prerequisites:

- MySQL ODBC driver must be installed on your operating system (for installation instructions, check the documentation supplied with the driver).
- You have the following database connection details: host, database, port, username, and password.

To connect to MySQL via ODBC:

1. [Start the database connection wizard](#).



2. Select **MySQL (ODBC)**, and then click **Next**.

Connecting to MySQL

Where can I find MySQL drivers?

Select an option how you wish to connect to the database and click Connect.

☒ Create a new Data Source Name (DSN) with the driver:

MySQL ODBC 5.3 Unicode Driver

☐ Use an existing Data Source Name:

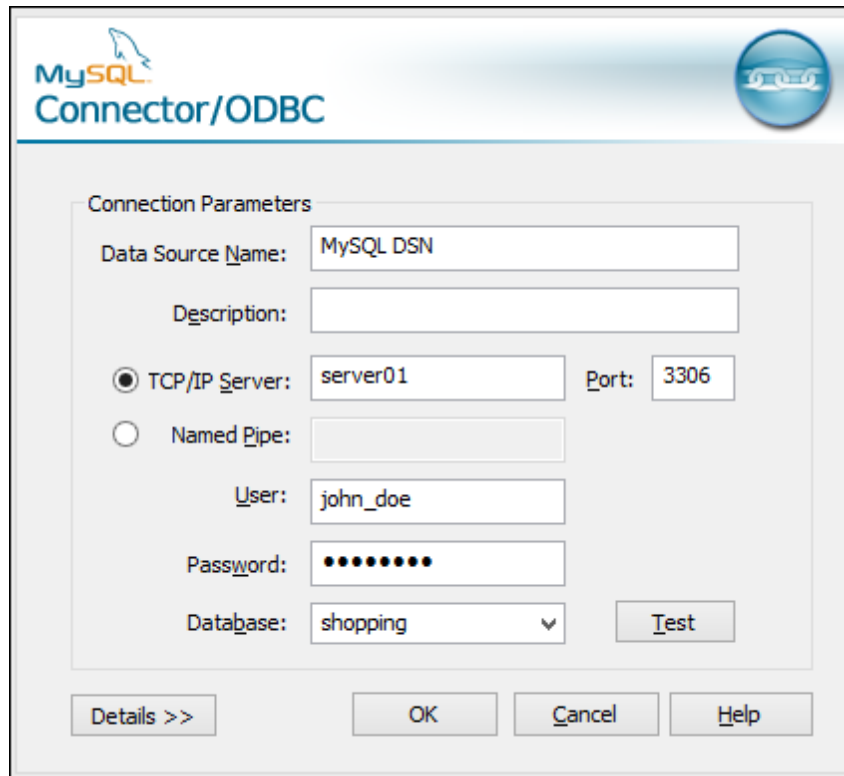
☒ User DSN ☐ System DSN

Edit Drivers

☐ Skip the configuration step for wizard

< Back Connect Close

3. Select **Create a new Data Source Name (DSN) with the driver**, and select a MySQL driver. If no MySQL driver is available in the list, click **Edit Drivers**, and select any available MySQL drivers (the list contains all ODBC drivers installed on your operating system).
4. Click **Connect**.



5. In the Data Source Name box, enter a descriptive name that will help you identify this ODBC data source in future.
6. Fill in the database connection credentials (TCP/IP Server, User, Password), select a database, and then click **OK**.

Note: If the database server is remote, it must be configured by the server administrator to accept remote connections from your machine's IP address. Also, if you click **Details>>**, there are several additional parameters available for configuration. Check the driver's documentation before changing their default values.

Connecting to Oracle (ODBC)

This example illustrates a common scenario where you connect from UModel to an Oracle database server on a network machine, through an Oracle database client installed on the local operating system.

The example includes instructions for setting up an ODBC data source (DSN) using the database connection wizard in UModel. If you have already created a DSN, or if you prefer to create it directly from ODBC Data Source administrator in Windows, you can do so, and then select it when prompted by the wizard. For more information about ODBC data sources, see [Setting up an ODBC Connection](#).

Prerequisites:

- The Oracle database client (which includes the ODBC Oracle driver) must be installed and configured on your operating system. For instructions on how to install and configure an Oracle database client, refer to the documentation supplied with your Oracle software.
- The **tnsnames.ora** file located in Oracle home directory contains an entry that describes

the database connection parameters, in a format similar to this:

```
ORCL =  
  (DESCRIPTION =  
    (ADDRESS_LIST =  
      (ADDRESS = (PROTOCOL = TCP) (HOST = server01) (PORT = 1521))  
    )  
    (CONNECT_DATA =  
      (SID = orcl)  
      (SERVER = DEDICATED)  
    )  
  )  
)
```

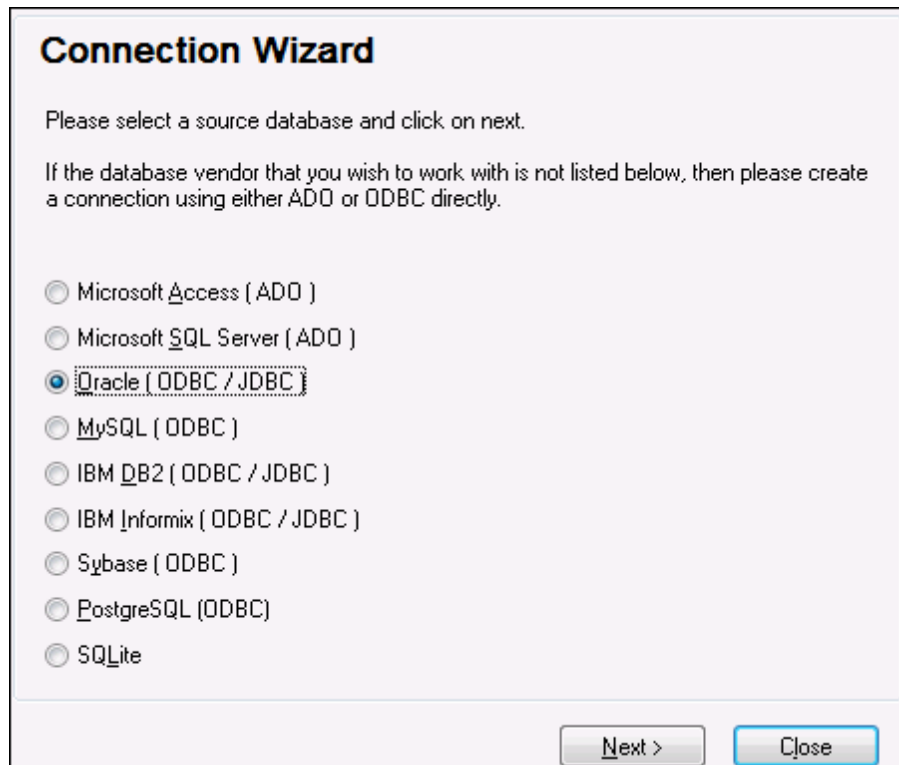
The path to the **tnsnames.ora** file depends on the location where Oracle home directory was installed. For Oracle database client 11.2.0, the default Oracle home directory path could be as follows:

```
C:\app\username\product\11.2.0\client_1\network\admin\tnsnames.ora
```

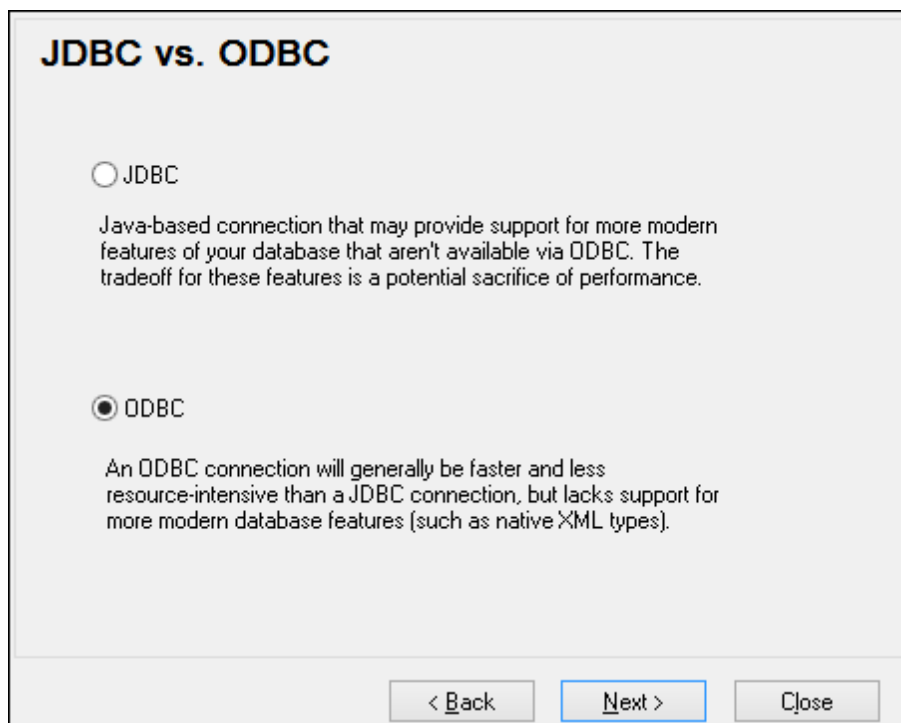
You can add new entries to the **tnsnames.ora** file either by pasting the connection details and saving the file, or by running the *Oracle Net Configuration Assistant* wizard (if available).

To connect to Oracle using ODBC:

1. [Start the database connection wizard.](#)



2. Select **Oracle (ODBC / JDBC)**, and then click **Next**.



JDBC vs. ODBC

☐ JDBC

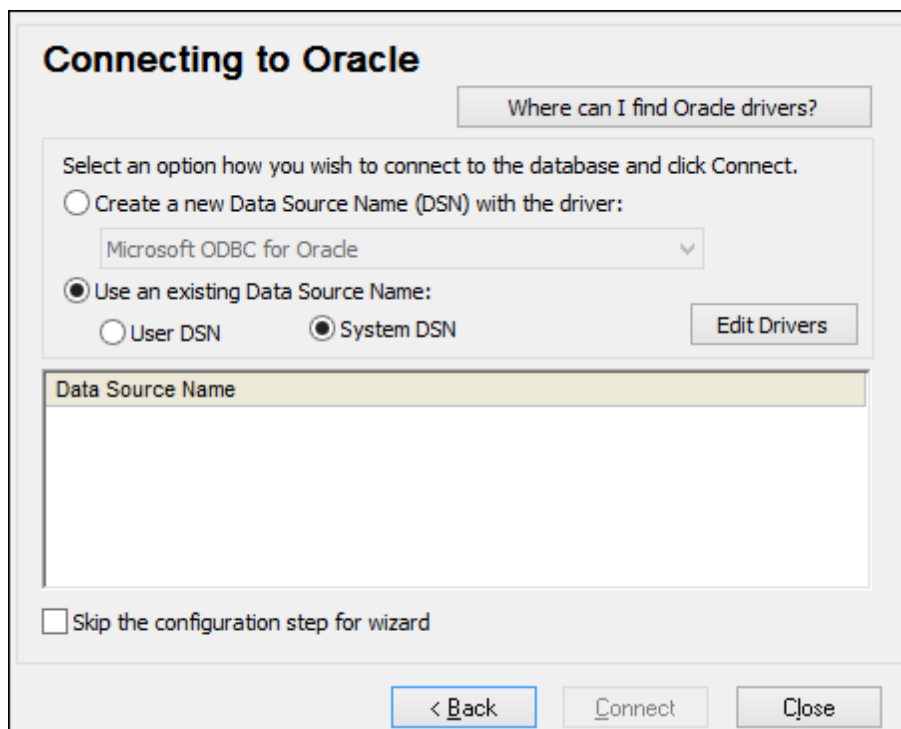
Java-based connection that may provide support for more modern features of your database that aren't available via ODBC. The tradeoff for these features is a potential sacrifice of performance.

☒ ODBC

An ODBC connection will generally be faster and less resource-intensive than a JDBC connection, but lacks support for more modern database features (such as native XML types).

< Back Next > Close

3. Select **ODBC**.



Connecting to Oracle

Where can I find Oracle drivers?

Select an option how you wish to connect to the database and click Connect.

☐ Create a new Data Source Name (DSN) with the driver:

Microsoft ODBC for Oracle

☒ Use an existing Data Source Name:

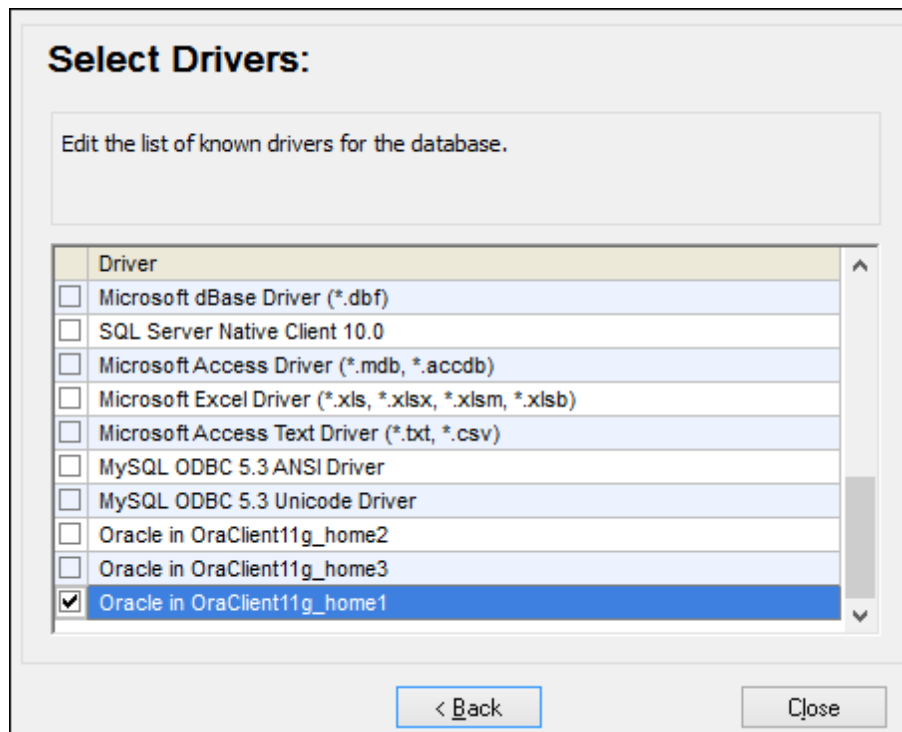
☐ User DSN ☒ System DSN Edit Drivers

Data Source Name

☐ Skip the configuration step for wizard

< Back Connect Close

4. Click **Edit Drivers**.



5. Select the Oracle drivers you wish to use (in this example, **Oracle in OraClient11g_home1**). The list displays the Oracle drivers available on your system after installation of Oracle client.
6. Click **Back**.
7. Select **Create a new data source name (DSN) with the driver**, and then select the Oracle driver chosen in step 4.

Connecting to Oracle

Where can I find Oracle drivers?

Select an option how you wish to connect to the database and click Connect.

☒ Create a new Data Source Name (DSN) with the driver:

Oracle in OraClient11g_home1

☐ Use an existing Data Source Name:

☐ User DSN ☒ System DSN

Edit Drivers

☐ Skip the configuration step for wizard

< Back Connect Close

Avoid using the Microsoft-supplied driver called **Microsoft ODBC for Oracle** driver. Microsoft recommends using the ODBC driver provided by Oracle (see <http://msdn.microsoft.com/en-us/library/ms714756%28v=vs.85%29.aspx>)

8. Click **Connect**.

Oracle ODBC Driver Configuration

Data Source Name: Oracle DSN 1

Description:

TNS Service Name: ORCL

User ID:

Buttons: OK, Cancel, Help, Test Connection

Application: Oracle | Workarounds | SQLServer Migration

Enable Result Sets: ☒ Enable Query Timeout: ☒ Read-Only Connection: ☐

Enable Closing Cursors: ☐ Enable Thread Safety: ☒

Batch Autocommit Mode: Commit only if all statements succeed

Numeric Settings: Use Oracle NLS settings

9. In the Data Source Name text box, enter a name to identify the data source (in this example, **Oracle DSN 1**).
10. In the TNS Service Name box, enter the connection name as it is defined in the **tnsnames.ora** file (see [prerequisites](#)). In this example, the connection name is **ORCL**.
11. Click **OK**.

Service Name: ORCL

User Name: john_doe

Password:

Buttons: OK, Cancel, About...

12. Enter the username and password to the database, and then click OK.

Connecting to PostgreSQL (ODBC)

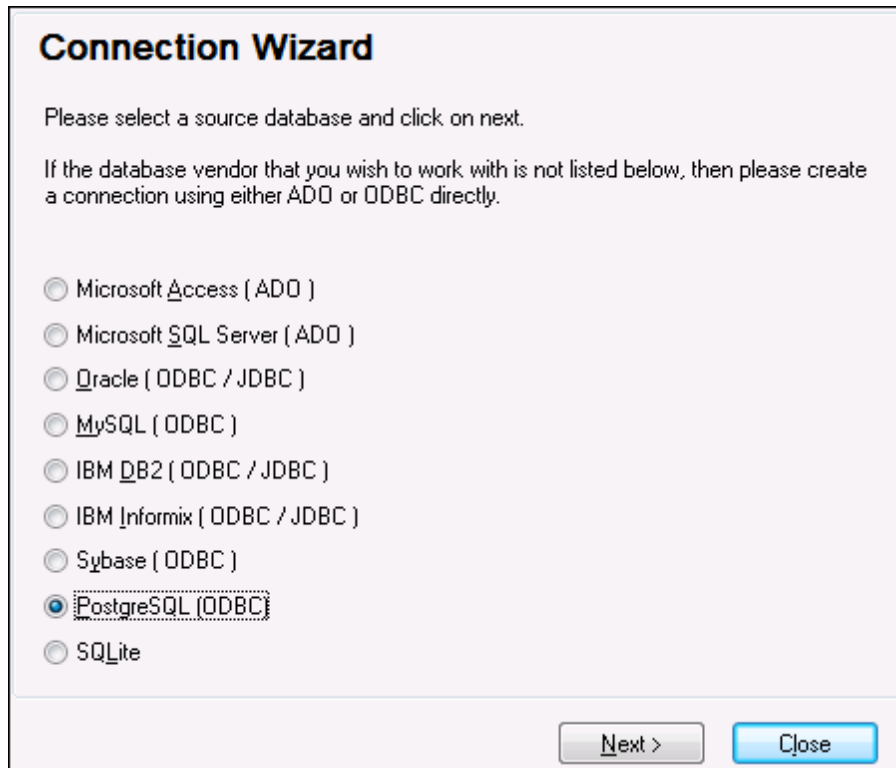
This topic provides sample instructions for connecting to a PostgreSQL database server from a Windows machine through the ODBC driver. The PostgreSQL ODBC driver is not available on Windows, so it must be downloaded and installed separately. This example uses the psqLODBC driver (version 09_03_300-1) downloaded from the official website (see also [Database Drivers Overview](#)).

Prerequisites:

- *psql/ODBC* driver must be installed on your operating system (for installation instructions, check the documentation supplied with the driver).
- You have the following database connection details: server, port, database, user name, and password.

To connect to PostgreSQL using ODBC:

1. [Start the database connection wizard.](#)



2. Select **PostgreSQL (ODBC)**, and then click **Next**.

Connecting to PostgreSQL

[Where can I find PostgreSQL drivers?](#)

Select an option how you wish to connect to the database and click Connect.

☒ Create a new Data Source Name (DSN) with the driver:

PostgreSQL Unicode

☐ Use an existing Data Source Name:

☒ User DSN ☐ System DSN

[Edit Drivers](#)

☐ Skip the configuration step for wizard

< Back Connect Close

3. Select **Create a new Data Source Name (DSN) with the driver**, and select the PostgreSQL driver. If no PostgreSQL driver is available in the list, click **Edit Drivers**, and select any available PostgreSQL drivers (the list contains all ODBC drivers installed on your operating system).
4. Click **Connect**.

Data Source PostgreSQL35w Description

Database SSL Mode disable

Server Port 5432

User Name Password

Options

Datasource Global

Test Save Cancel

5. Fill in the database connection credentials (Database, Server, Port, User Name, Password), and then click **OK**.

Connecting to Sybase (JDBC)

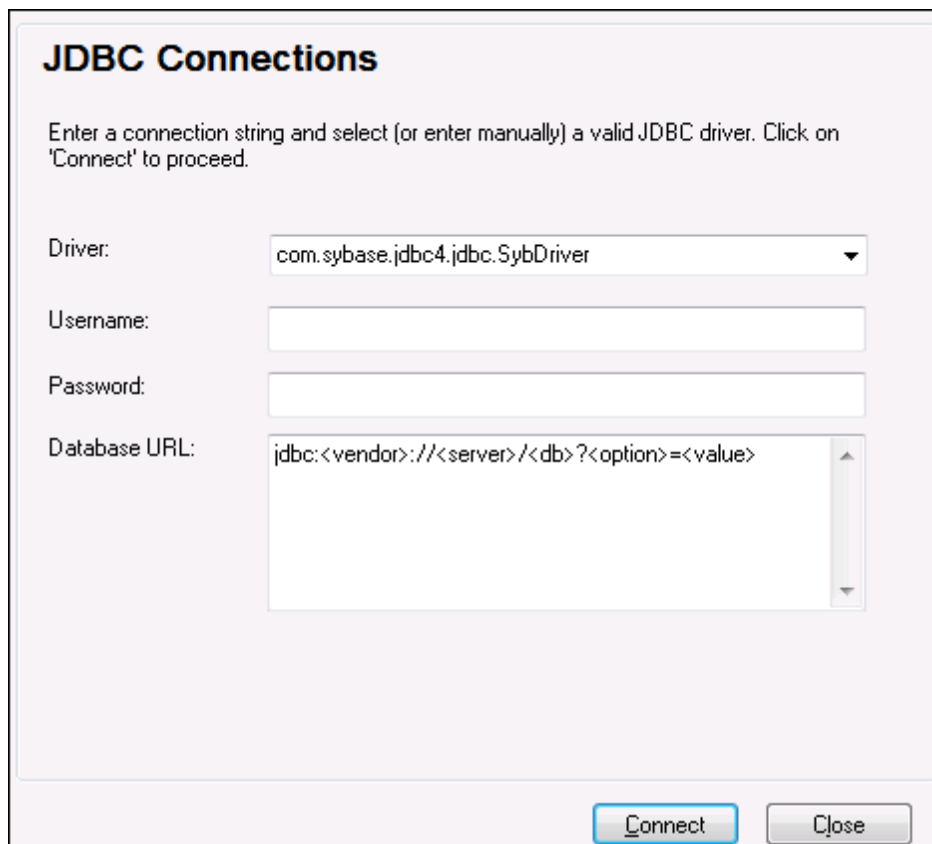
This topic provides sample instructions for connecting to a Sybase database server through JDBC.

Prerequisites:

- Java Runtime Environment (JRE) must be installed on your operating system.
- Sybase *jConnect* component must be installed on your operating system (in this example, *jConnect 7.0* is used, installed as part of the *Sybase Adaptive Server Enterprise PC Client* installation). For the installation instructions of the database client, refer to Sybase documentation.
- The operating system's `CLASSPATH` environment variable includes the path where the Sybase JDBC driver was installed. In this example, the JDBC driver is installed in the directory `C:\Sybase`, and the value of `CLASSPATH` variable was configured to include the path `C:\sybase\jConnect-7_0\classes\jconn4.jar`. For more information, see [Configuring the CLASSPATH](#).
- You have the following database connection details: host, port, database name, username, and password.

To connect to Sybase through JDBC:

1. [Start the database connection wizard](#).
2. Click **JDBC Connections**.
3. Select the Sybase JDBC driver from the list of available JDBC drivers (in this example, **com.sybase.jdbc4.jdbc.SybDriver**). If the list does not contain a Sybase driver, it is either not installed correctly, or not included in the `CLASSPATH` variable (see the list of prerequisites above).



4. Enter the username and password to the database in the corresponding text boxes.
5. Enter the connection string to the database server in the Database URL text box, by

replacing the highlighted values with the ones applicable to your database server.

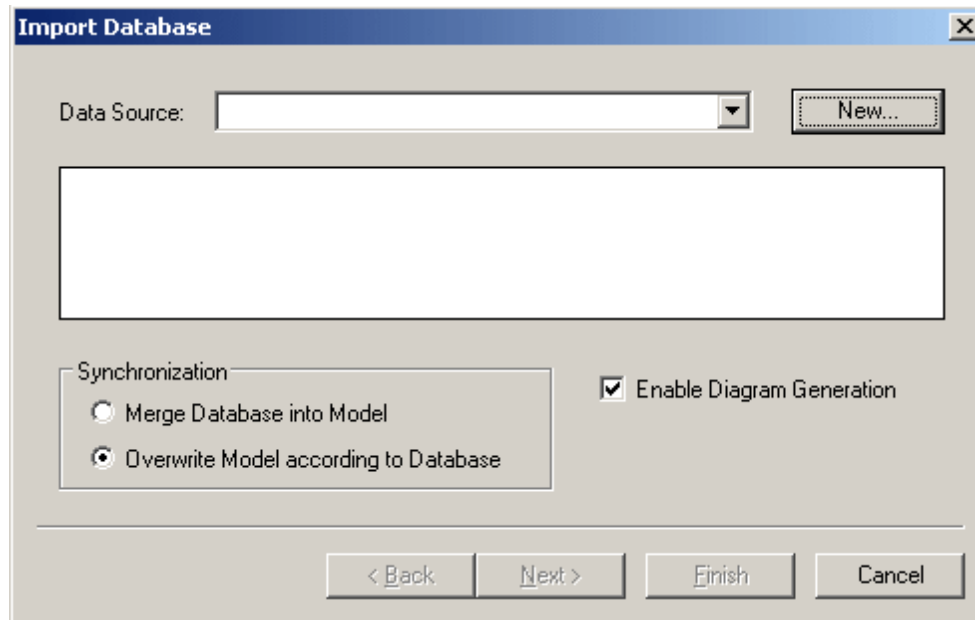
```
jdbc:sybase:Tds:hostName:port/databaseName
```

6. Click **Connect**.

11.2 Importing SQL databases into UModel

To import an SQL database into UModel:

1. Select **Project | Import SQL database**.
2. Click the New... button if this is the first time you are importing a database into UModel.

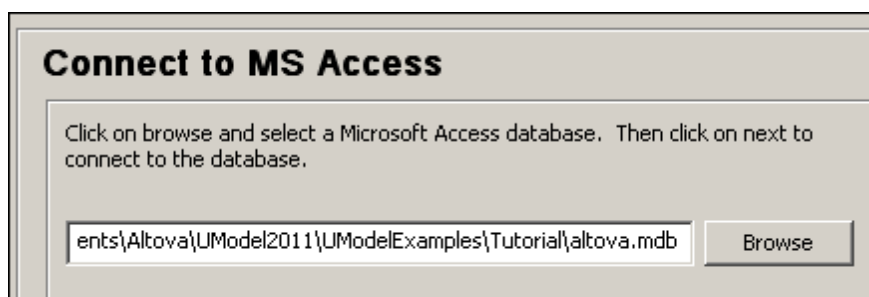


(If you have previously imported a database click the Data Source combo box and select the data source.)

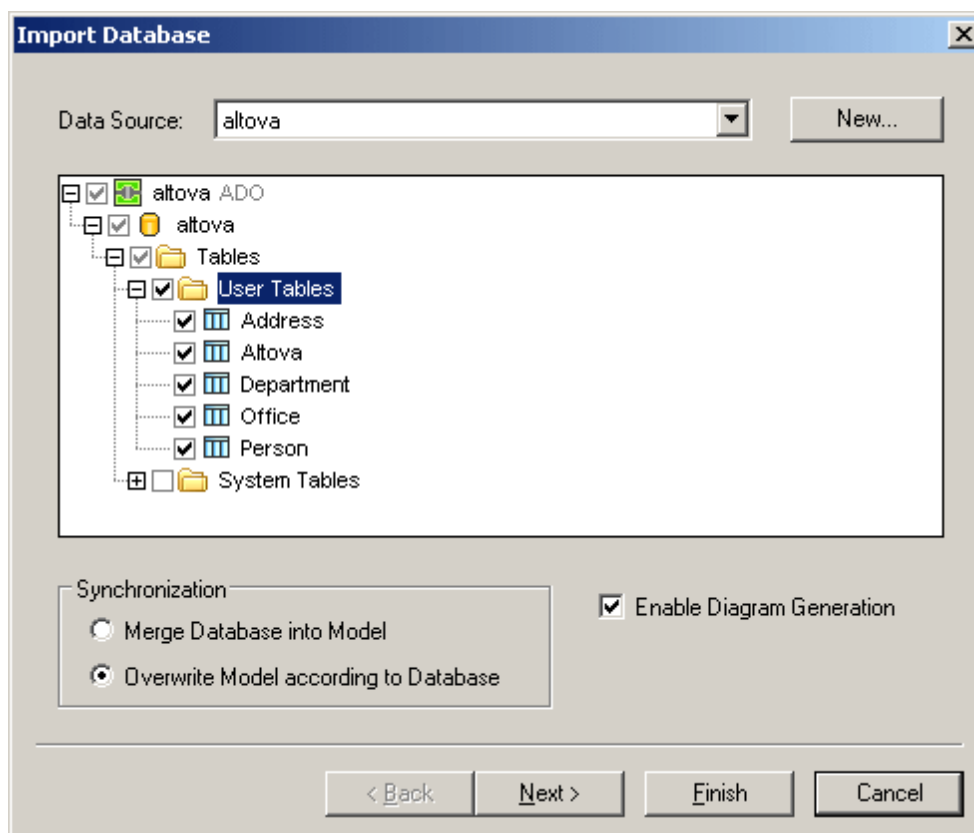
3. Use the [Connection wizard](#) to select the database and the specific tables that you want to insert into UModel.



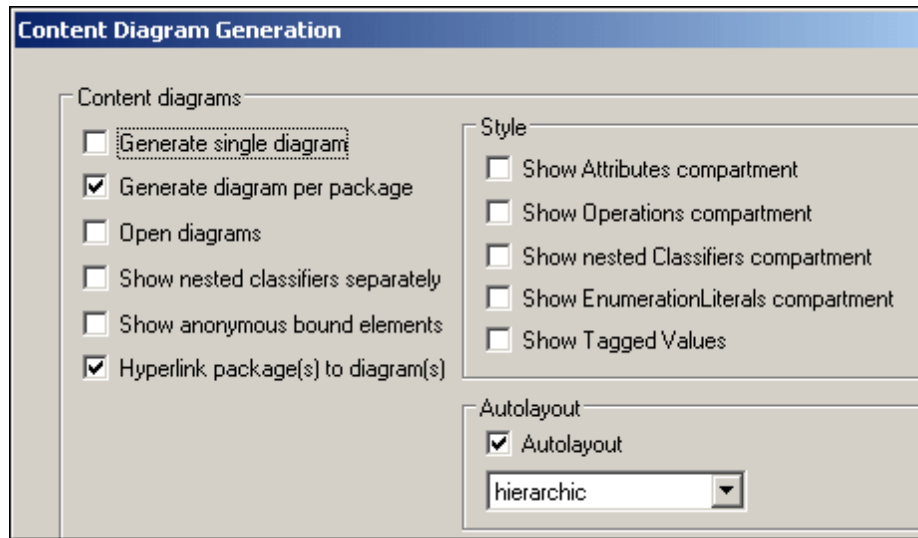
In this example we are going to insert an MS Access database supplied with UModel called *Altova.mdb*. Note that any of the databases listed in the Connection Wizard can be imported into UModel.



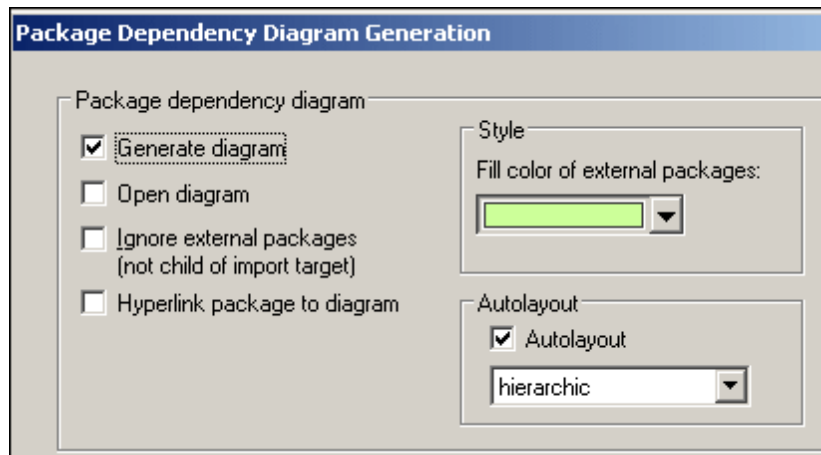
4. Click Connect to connect to the database, and enter a Data Source Name in the prompt that follows e.g. Altova.
5. The following dialog box allows you to select the specific tables that you want to import.
6. Click the User Tables entry to select all non-system tables, then click Next to continue.



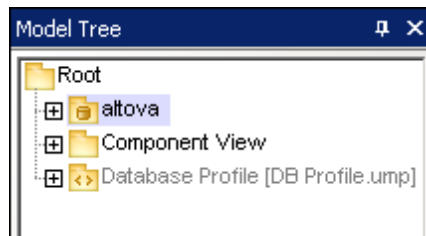
The following dialog box allows you to define the database diagram settings; the default settings are shown.



7. Click Next again, to define the Package Dependency Diagram settings.

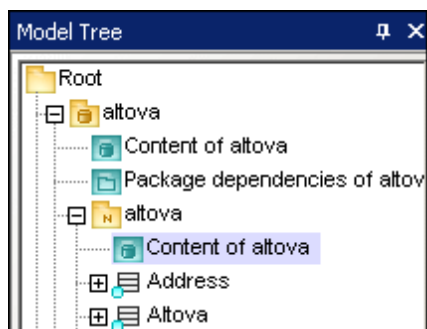


8. Click finish to insert the database into UModel.

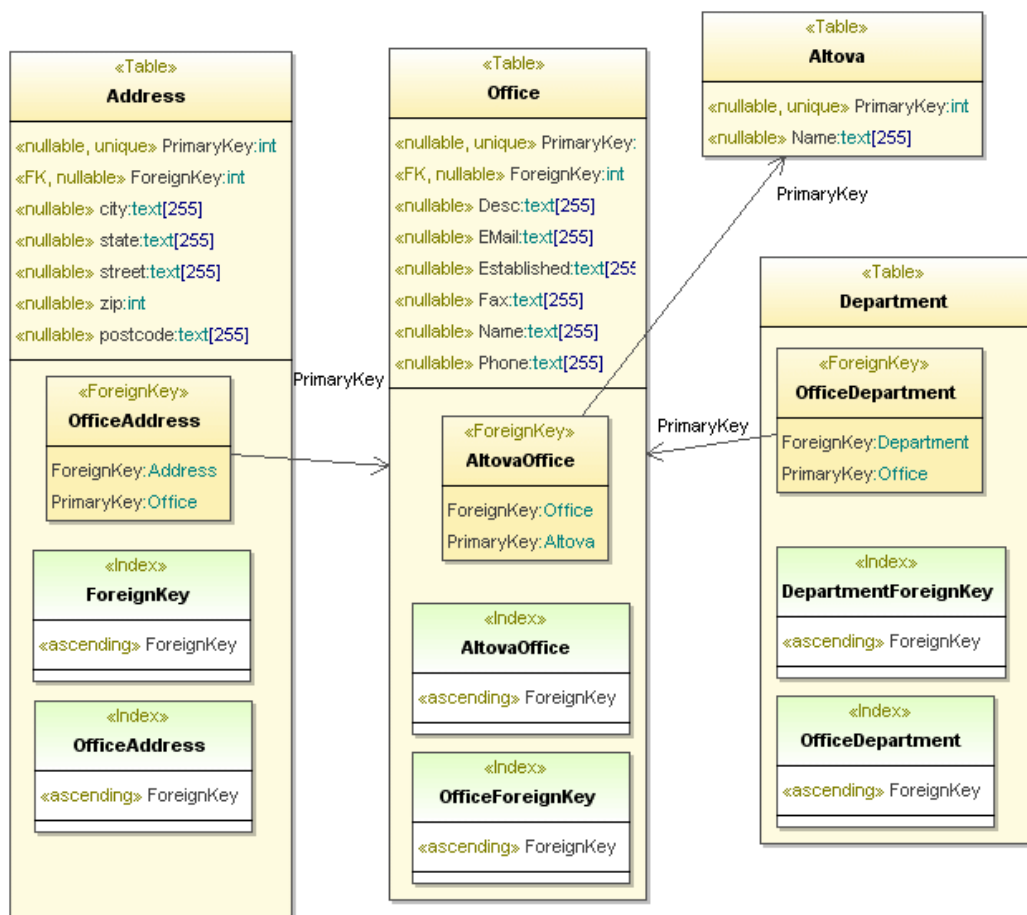


The database has now been imported into two packages. Note that the Database Profile package has also been automatically imported.

9. Expand the altova package until you see the "Content of altova" diagram icon.



10. Double click the "Content of altova" icon to see the database modeling elements.



11.3 Viewing database modeling elements

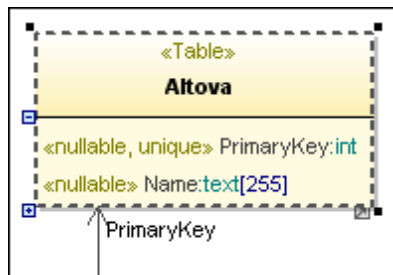
- **Tables** in the UML data profile are shown as classes with the <<Table>> stereotype.
- **Columns** are shown as properties/attributes of the Table class.
- **Behaviour** i.e. operations, are shown as stereotyped classes. E.g. <<index>>, <<Check constraints>>, etc.
- **Relationships** are shown as associations between tables.

To view the various table properties, using e.g. database altova.mdb:

1. Click the top left expand button of the Altova table.

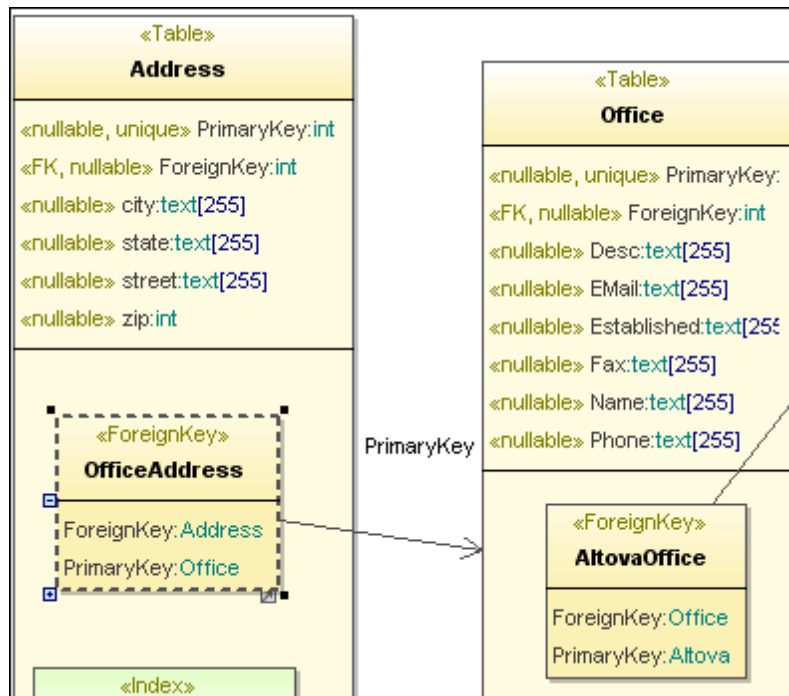


This opens the properties compartment of the Altova Table class and displays the PrimaryKey and Name columns.



ForeignKey classes

1. Click the top expand button of the Address Table.
The first two fields/columns are the Primary and Foreign key entries. The rest are the Address fields.



The **OfficeAddress** <<ForeignKey>> class defines the primary/foreign key relationships between tables.

- The **ForeignKey:Address** entry stipulates that there is a column, named "ForeignKey", in the owner Table "Address", which references a foreign column.
- The **PrimaryKey:Office** entry stipulates that the column "PrimaryKey" in the table "Office", is referenced as a foreign key by the above-mentioned column, (Address) ForeignKey.

Note that an association arrow, labeled PrimaryKey, exists between the Address and Office Tables to show the relationship.

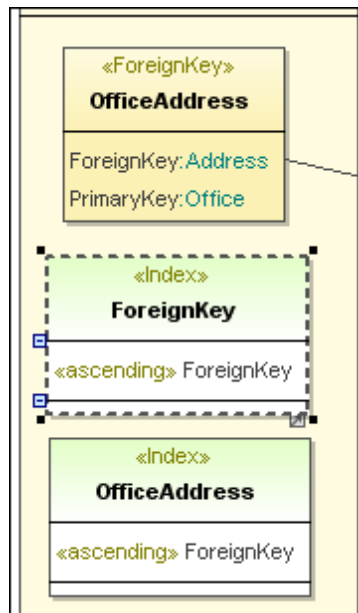
When adding ForeignKey classes make sure that:

- The **first** entry/column refers to the **owner** table, Address in this case
- the **second** entry/column refers to the **foreign** table, Office in this case

Index classes

The Index classes show the currently defined indexes for each table. The Address table has two indexes: ForeignKey and OfficeAddress.

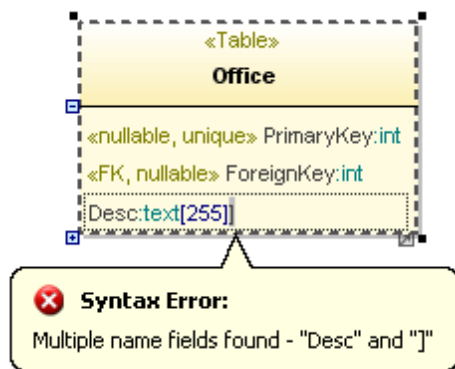
- Index **ForeignKey** is the foreign key of the Address table and has an ascending sort.
- Index **OfficeAddress** is the foreign key of the OfficeAddress class and also has an ascending sort.

**Note:**

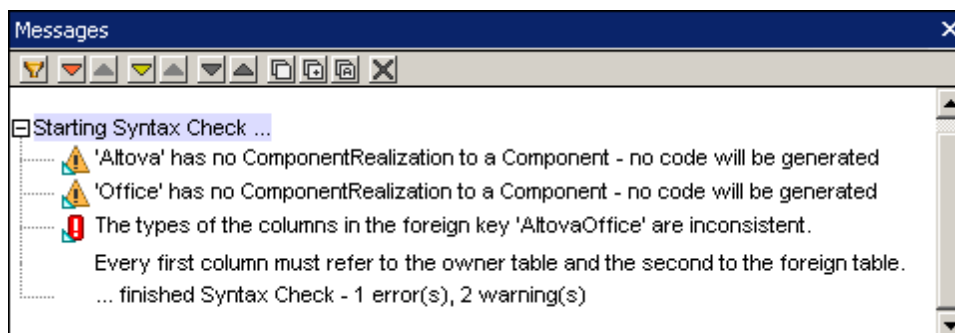
Multiple primary/foreign keys can be defined per ForeignKeys class. Multiple Indexes can also be defined per table, you are not restricted to the two shown in the examples above.

Syntax checks:

Syntax checks are automatically performed when adding structural information, and a Syntax Error pop-up message automatically appears when necessary.



To perform a syntax check on your whole project, select **Project | Check Project Syntax**.



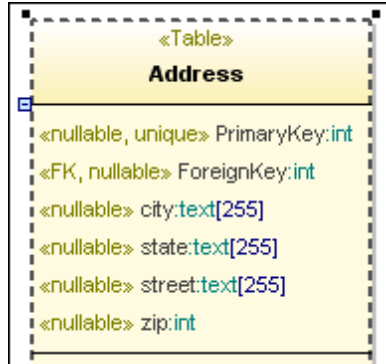
A complete rundown of all project-related errors and warnings is presented in the Messages window.

11.4 Adding columns to a table

Database modeling elements can be added in UModel and then synchronized with a database(s).
E.g. we want to add a postcode field to the Address table.

To add a new column/field to a table:

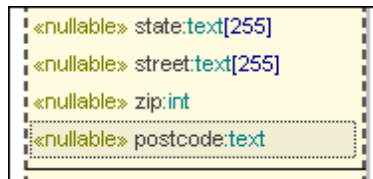
1. Right click the Address table and select **New | Column**.



2. Replace the "Column1" text with the name of the field i.e. postcode, enter a colon and select the data type e.g. text.

«nullable» postcode:text

3. Click between the guillemets (angle brackets), enter "n" and select **nullable** from the autocompletion list.

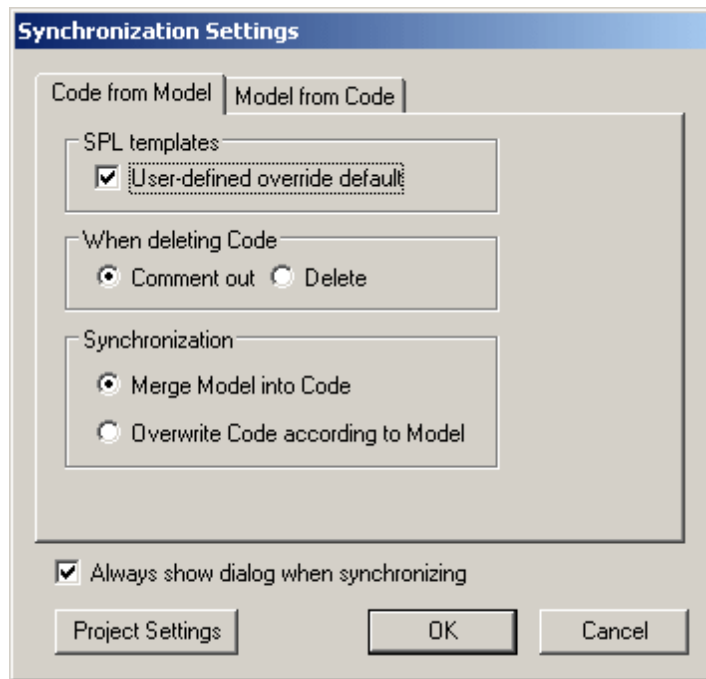


4. Enter a square bracket "]" and select the text length that the text field should have e.g. 255, and press **Enter**.



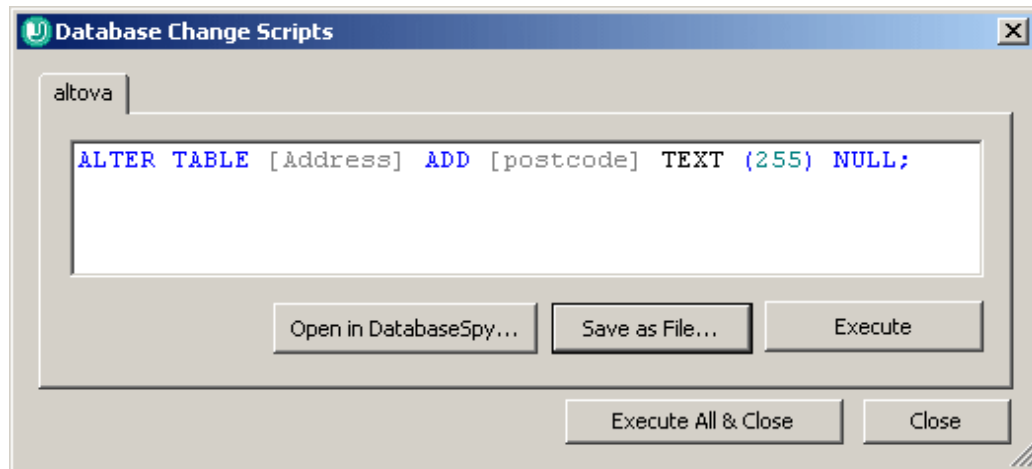
To synchronize the updated model with the database (i.e. code):

1. Select the menu option **Project | Merge Program code from UModel project**.

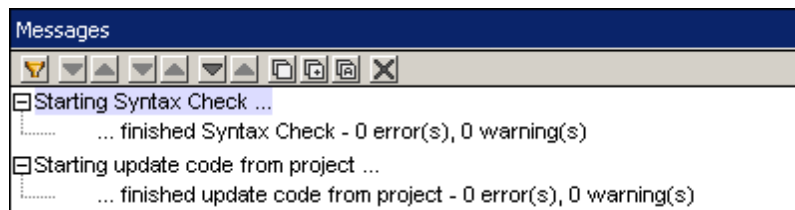


2. If the default synchronization settings shown in the dialog box are fine, click OK to continue.

A Database Change Script dialog box is opened. It shows the SQL change script commands needed to update the database.



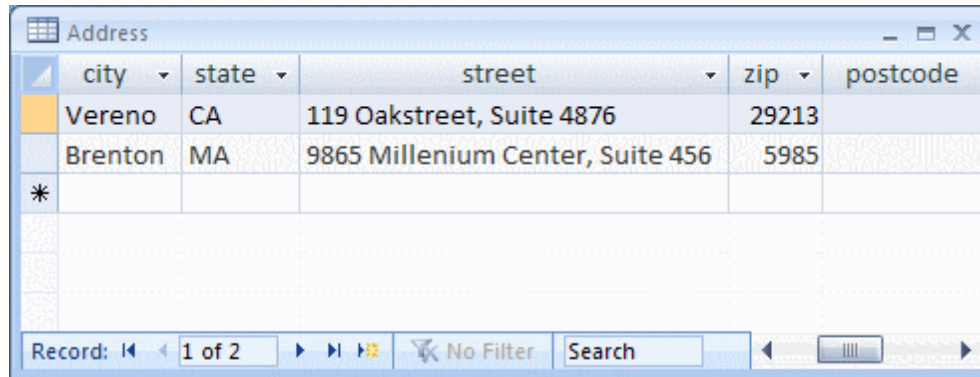
3. To execute the code immediately, click the **Execute** button.
A prompt appears with specific details of the database update execution process.
Please look in the Messages window for syntax check and update code information.



Please note:

Please make sure that you execute the change script to keep the model and database in sync! The next synchronization process will proceed as if the change script has been executed.

Starting the database will show you that the postcode column has been added to the database.



city	state	street	zip	postcode
Vereno	CA	119 Oakstreet, Suite 4876	29213	
Brenton	MA	9865 Millenium Center, Suite 456	5985	
*				

To delete a column:

The method described above can also be used to delete a column of a database.

- Select the operation/column you want to delete, and delete it from the table.
- Synchronize the model with the database.
- The change script generates a DROP COLUMN command.

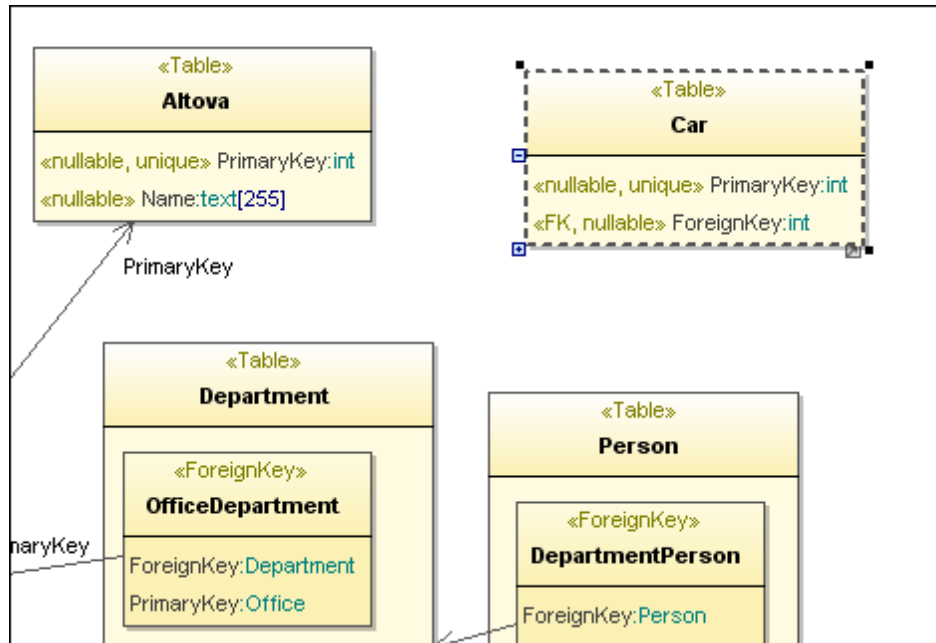
Please note:

Executing this command on an existing database will, of course, delete the column and all the data it contains.

11.5 Adding a new table to model / database

To add a new table to the model/database:

1. Right click in the database diagram and select the **New | Table**, and enter **Car** as the table name.
2. Press the **F7** function key, then press the left arrow key of the keyboard to set the cursor at the far left.
3. Place the cursor between the angle brackets and enter **<<nullable, unique>>**. The autocompletion feature helps you select the entries, press Enter to select one.
4. Move the cursor to Property1 and overwrite it with **PrimaryKey:int** then hit Enter.

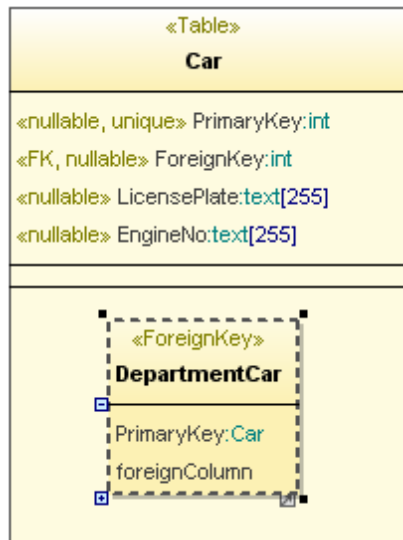


5. Press **F7** and use the same method to enter **<<FK, nullable>>ForeignKey:int**. This ends the key definition for the table.
6. Press **F7** and enter the **LicensePlate** and **EngineNo** columns/fields as shown below. When entering the text length by pressing the square open bracket "[" key, note that the **closing** bracket is automatically supplied. Press Enter after entering the length number, 255.



Adding ForeignKey and Index classes:

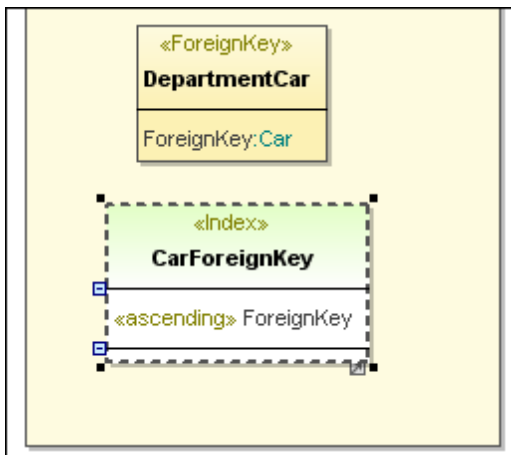
1. Right click the Car table and select **New | ForeignKey** and enter the key name **DepartmentCar**.



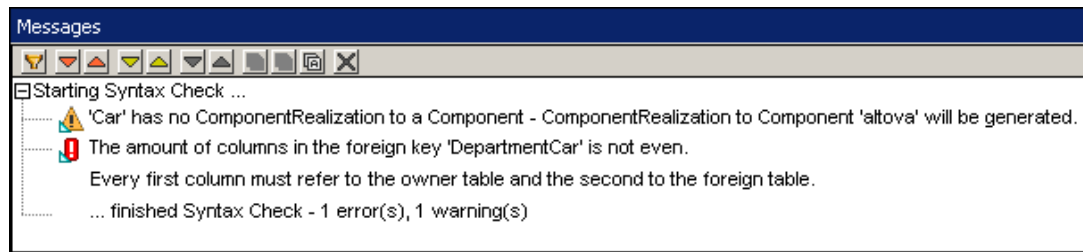
2. Double click the top entry/property (PrimaryKey:Car) of the class and change it to **ForeignKey:Car**.



3. Delete the second entry (foreignColumn) below it.
4. Right click the **Car** table and select **New | Index**, enter the index name **CarForeignKey**.
5. Double click the (automatically inserted) IndexedColumn entry, and change it to **ForeignKey**.
6. Place the cursor between the angle brackets and select <<**ascending**>> from the autocompletion list.



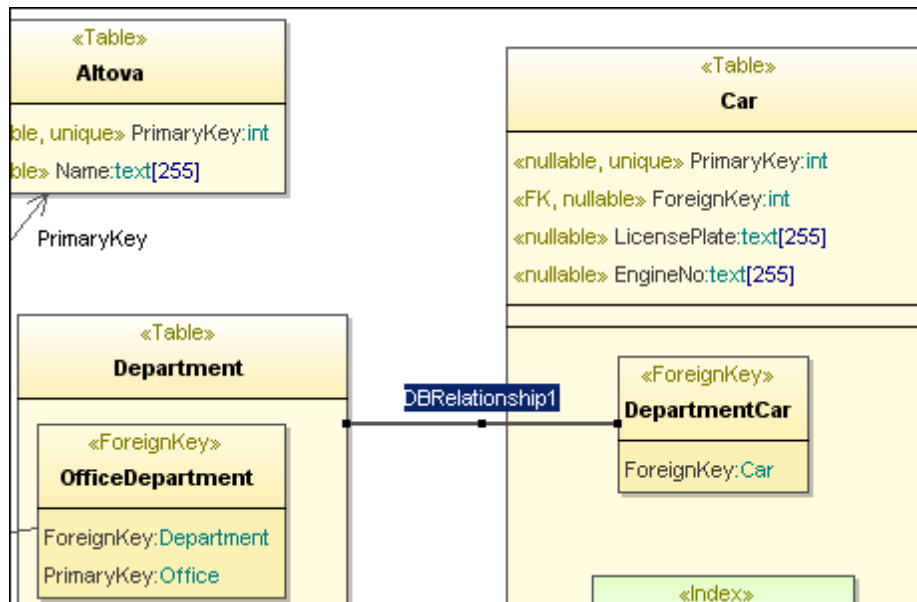
7. Select **Project | Check Project Syntax** to see if all database rules have been adhered to.



The error message refers to the fact that only one operation/column has been created in the DepartmentCar class, while at least two are needed. The next section will add the missing operation/column.

To create relationships between tables:

1. Click the **Database Relationship Association** icon  in the icon bar.
2. Click the **DepartmentCar** class in the **Car** table and drop the line on the **Department** table.

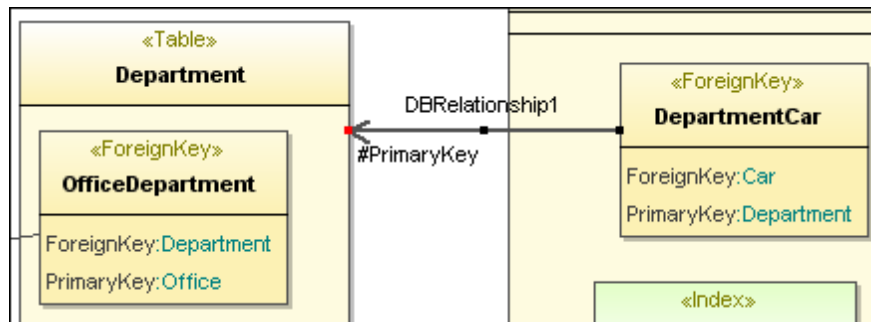


This creates an association between the two tables.

3. With the association selected, click in the **A:name** field of the Properties tab, and enter **PrimaryKey**.
4. Click the **A:memberEndKind** combo box and select **memberEnd**.

name	DBRelationship1
qualified name	altova::altova::DBRelation
element kind	Association
visibility	public
leaf	☐
abstract	☐
isFinalSpecialization	☐
derived	☐
A: name	PrimaryKey
A: aggregation	none
A: memberEndKind	memberEnd
A: multiplicity	
B: name	
B: aggregation	none
B: memberEndKind	ownedEnd
B: multiplicity	

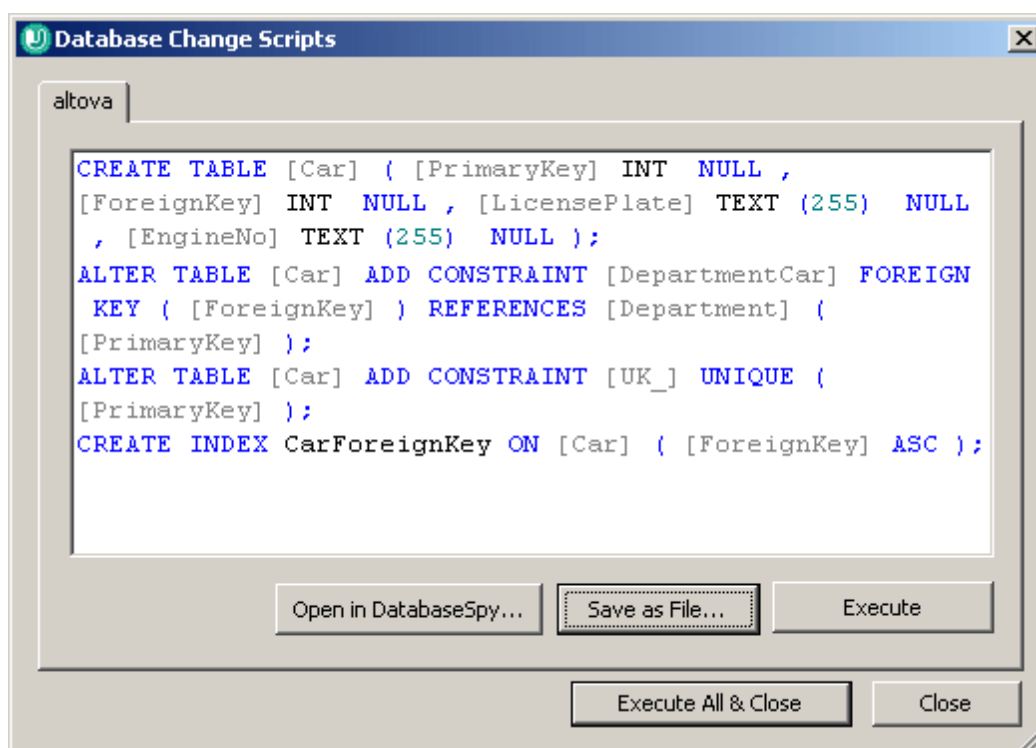
Note that this adds a new property, **PrimaryKey:Department** to the **DepartmentCar** ForeignKey class.



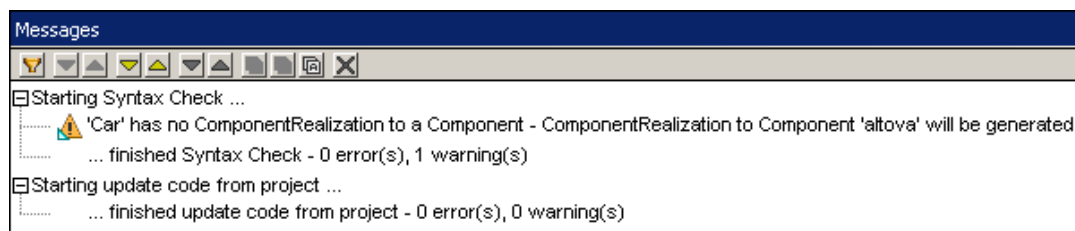
5. Selecting **Project | Check Project Syntax**, shows that only one warning remains.

To synchronize the model with the database:

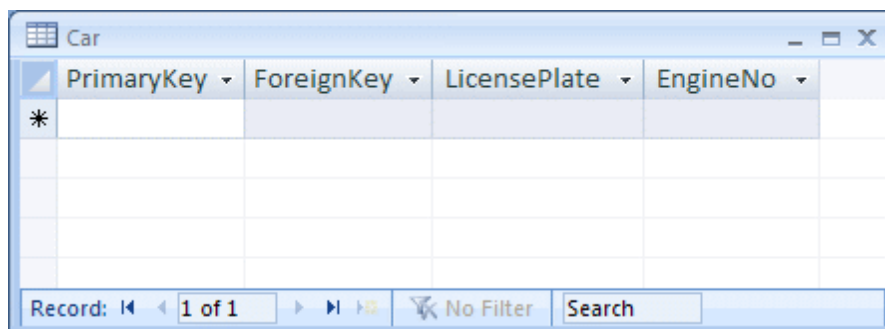
1. Select the menu option **Project | Merge Program code from UModel project**. This opens the Synchronization settings dialog box.
2. Click OK if the settings are correct.
Database Change Script code is generated and is displayed in a dialog box. Note that the change script cannot be edited here. Each **tab** in this dialog box represents a **database**.
3. Click the **Execute All and Close** button.



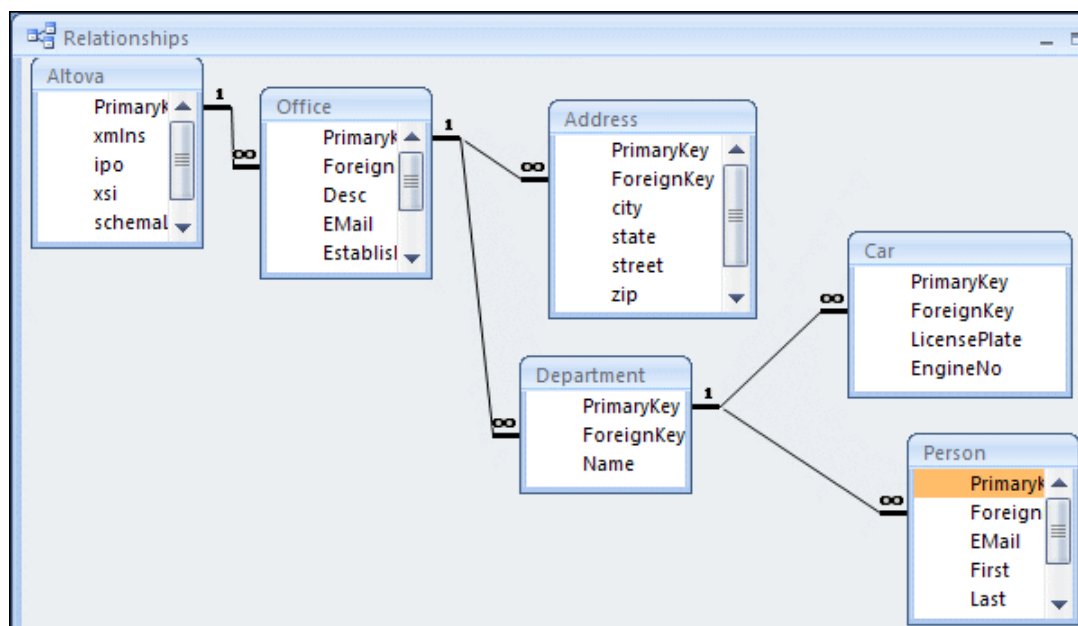
A message box appears displaying any errors or warnings, click OK to continue.
More detailed information on the update process is supplied in the Messages window.



Opening the Altova database shows that the new Car table, and all of its columns, have been added to the database.



The relationship structure can be seen in the Relationships window.



Chapter 12

XMI - XML Metadata Interchange

12 XMI - XML Metadata Interchange

Altova web site:  [Exchanging UModel projects using XMI](#)

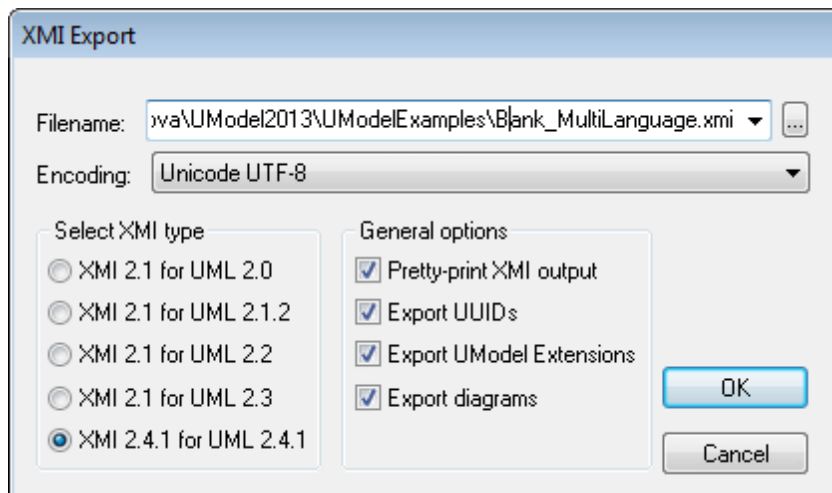
UModel supports the export and import of XMI 2.4 for UML 2.0 / 2.1 / 2.1.1 and 2.1.2, 2.2, 2.3, 2.4. Do not use the Export to XMI function to archive your UModel projects, please archive the *.ump project files instead.

Select the menu item **File | Export to XMI File** to generate an XMI file from the UModel project, and **File | Import from XMI File**, to import a previously generated XMI file.

The XMI Export dialog box allows you to select the specific XMI format you want to output, XMI for UML 2.0/2.1.1. During the export process included files, even those defined as "[include by reference](#)" are also exported.

Please note:

If you intend to **re-import** generated XMI code into UModel, please make sure that you activate the "Export UModel Extensions" check box.



Pretty-print XMI output

This option outputs the XMI file with XML appropriate tag indentation and carriage returns/line feeds.

Export UUIDs

XMI defines three versions of element identification: IDs, UUIDs and labels.

- IDs are unique within the XMI document, and are supported by most UML tools. UModel exports these type of IDs by default, i.e. none of the check boxes need activated.
- UUID are Universally Unique Identifiers, and provide a mechanism to assign each element a global unique identification, GUID. These IDs are globally unique, i.e. they are not restricted to the specific XMI document. UUIDs are generated by selecting the "Export UUIDs" check box.
- UUIDs are stored in the standard canonical UUID/GUID format (e.g "6B29FC40-CA47-1067-B31D-00DD010662DA", "550e8400-e29b-41d4-a716-446655440000",...)
- Labels are not supported by UModel.

Please note:

The XMI import process automatically supports both types of IDs.

Export UModel Extensions

XMI defines an "extension mechanism" which allows each application to export its tool-specific extensions to the UML specification. Other UML tools will, however, only be able to import the standard UML data (ignoring the UModel extensions). This UModel extension data will be available when importing into UModel.

Data such as the file names of classes, or element colors, are not part of the UML specification and thus have to be deleted in XMI, or be saved in "Extensions". If they have been exported as extensions and re-imported, all file names and colors will be imported as defined. If extensions are not used for the export process, then these UModel-specific data will be lost.

When importing an XMI document, the format is automatically detected and the model generated.

Export diagrams

Exports UModel diagrams as "Extensions" in the XMI file. The option "Export UModel Extensions" must be active to be able to save the diagrams as extensions.

Chapter 13

Teamwork support for UModel projects

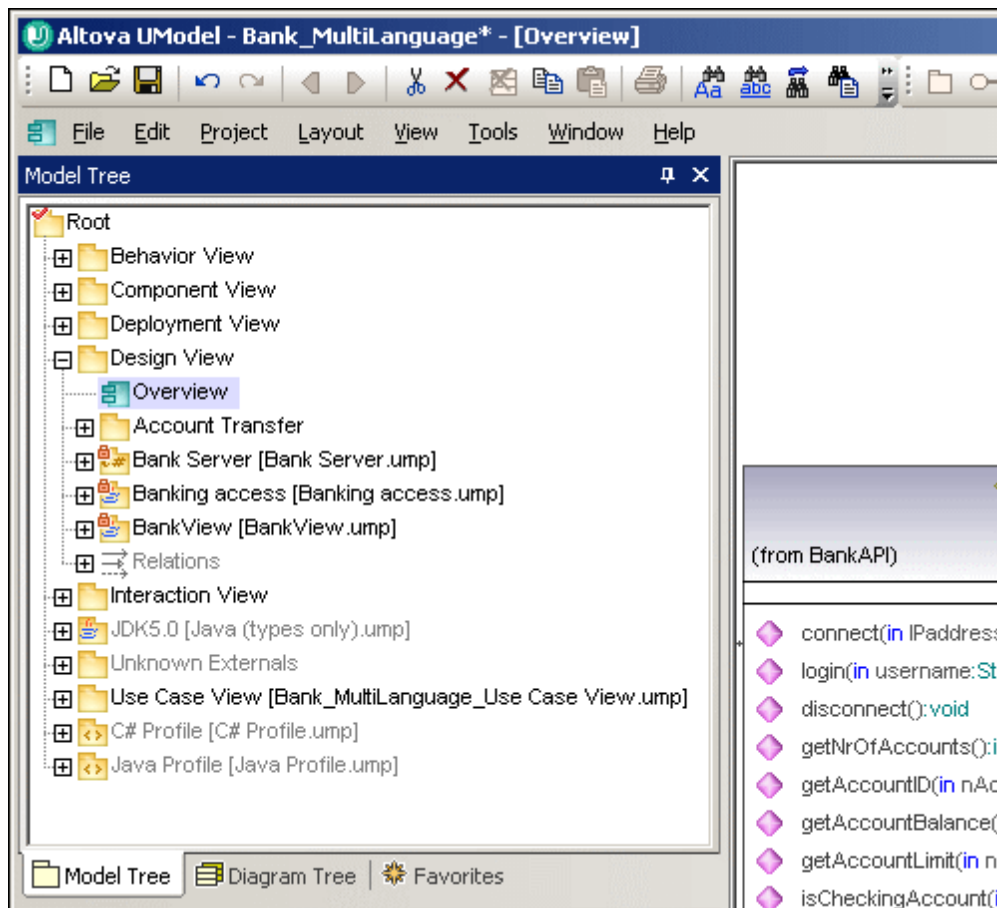
13 Teamwork support for UModel projects

UModel projects can be split up into multiple subprojects allowing several developers to simultaneously edit different parts of a single project. Each individual subproject can be added to a source control system.

The top-level project i.e. the project from which the subprojects can be included, is called the **main** project. Subprojects are created on the package level as individual UModel project files and have a *.ump extension.

Subprojects can be created/included in two different ways:

- As **editable** from within the **main** project, and at the subproject level
- As read-only from within the main project, i.e. only editable at the subproject level.



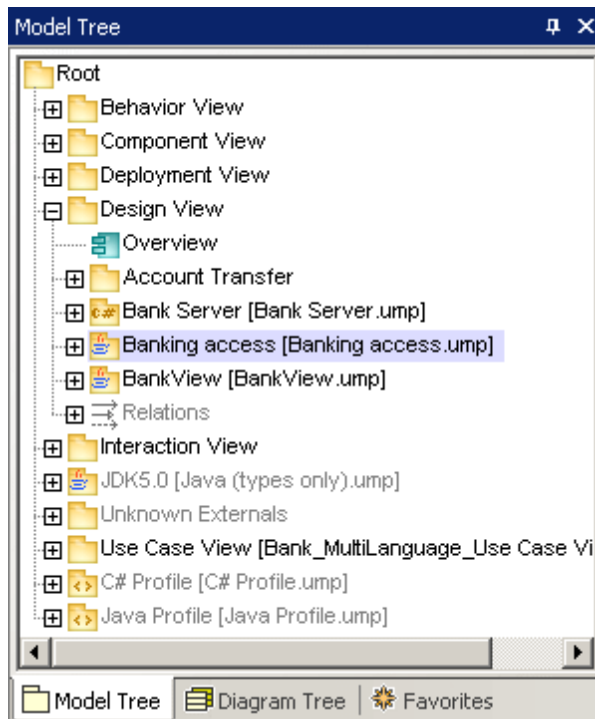
Subprojects can be structured in any way that you wish, in a flat or hierarchical structure, or a combination of both. This makes it theoretically possible to split off every package of a main project, into read-only/editable subproject files.

During the code-engineering process all subordinate components of a subproject are considered, there is no difference between a single project file or one that consists of multiple editable subprojects.

This also applies to UML diagrams, they can also be edited at the main, or subproject, level.

13.1 Creating and editing subproject files

The example discussed here shows how the **Bank_MultiLanguage.ump** project was split up into multiple subprojects.



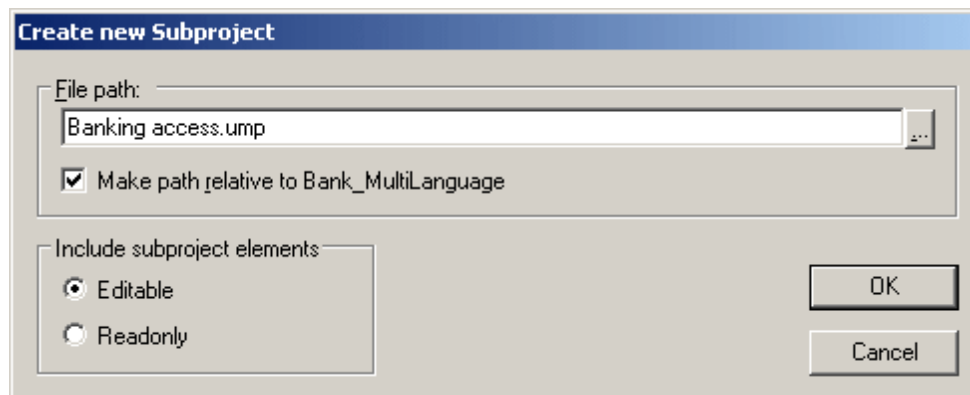
Please note:

UModel allows you to share packages and UML diagrams they might contain, between different projects. Packages can be included in other projects by reference, or as a copy. Please see [Sharing Packages and Diagrams](#) for more information.

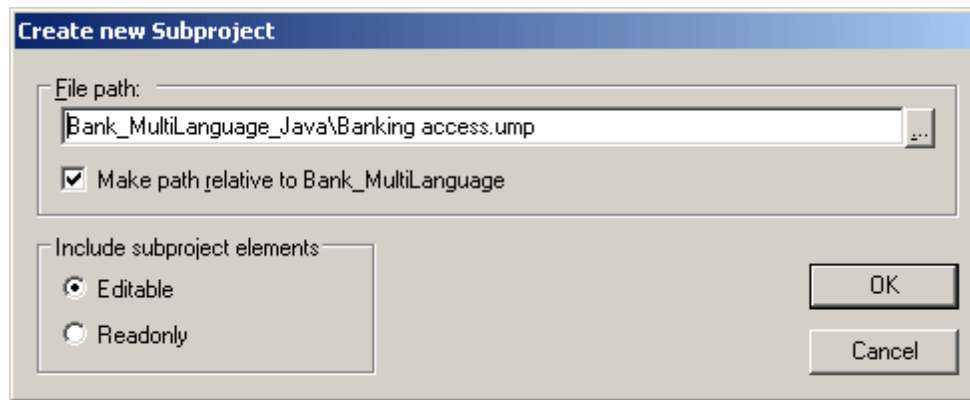
To create a subproject file

Subproject files can be split off a main, or subproject, file at any time.

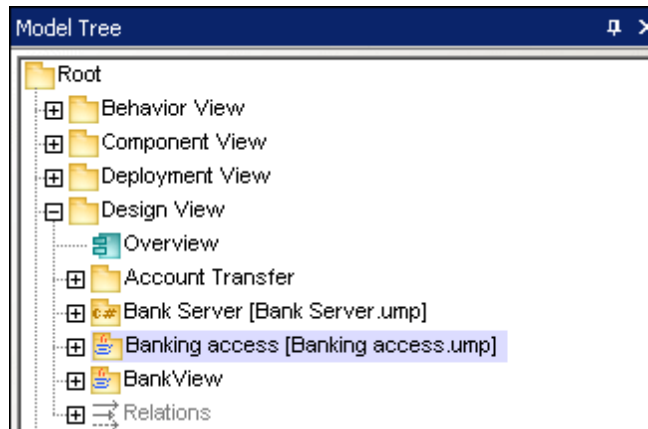
1. Right click a package(s) e.g. **Banking access** and select **Subproject | Create new Subproject**.



2. Click the Browse button and select the subdirectory **\\Bank_MultiLanguage_Java**.

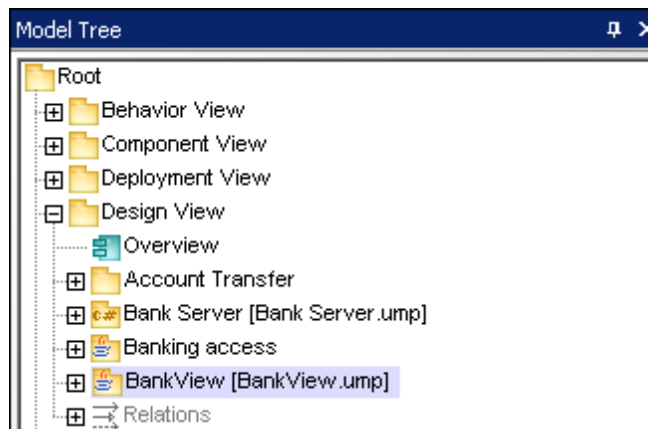


3. Select "Editable" to be able to edit the subproject from the main project. (Selecting Read-only makes it uneditable in the main project.) Then click OK.



The name of the subproject file is shown in square brackets next to the package name and the file **Banking access.ump** file is placed in the **UModelExamples\Bank_MultiLanguage_Java** folder.

The same method is used to create a subproject of the BankView folder. The **BankView.ump** file is placed in the **...UModelExamples\Bank_MultiLanguage_Java** folder.



Please note:

The file path of the subproject can be changed at any time by right clicking the subproject

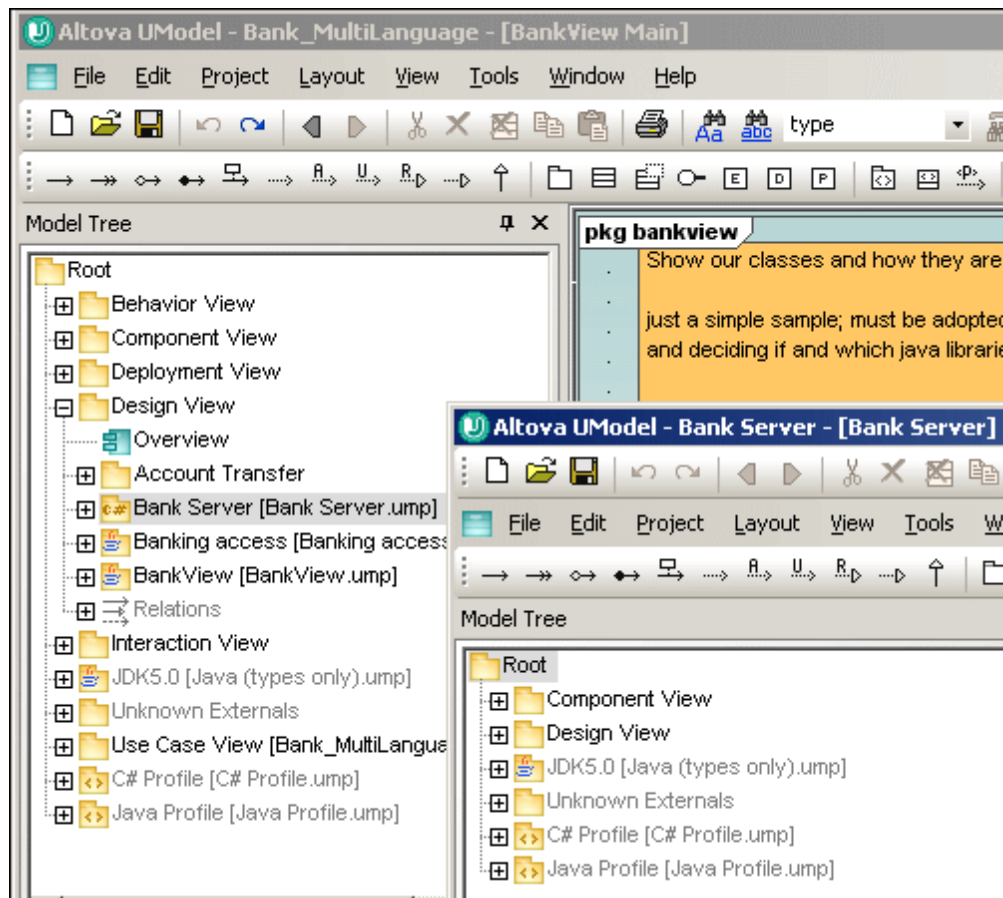
and selecting **Subproject | Edit File Path**.

Opening and editing subproject files

Having created a subproject file, the resulting *.ump file can then be opened, and edited, as a "main" UModel project file. For this to be successful there should not be any unresolved references to other elements. UModel automatically performs checks when creating a subproject from the "main" project, and whenever a file is saved.

1. Right click the subproject package (e.g. Bank Server.ump) in the main project and select **Project | Open as Project**.

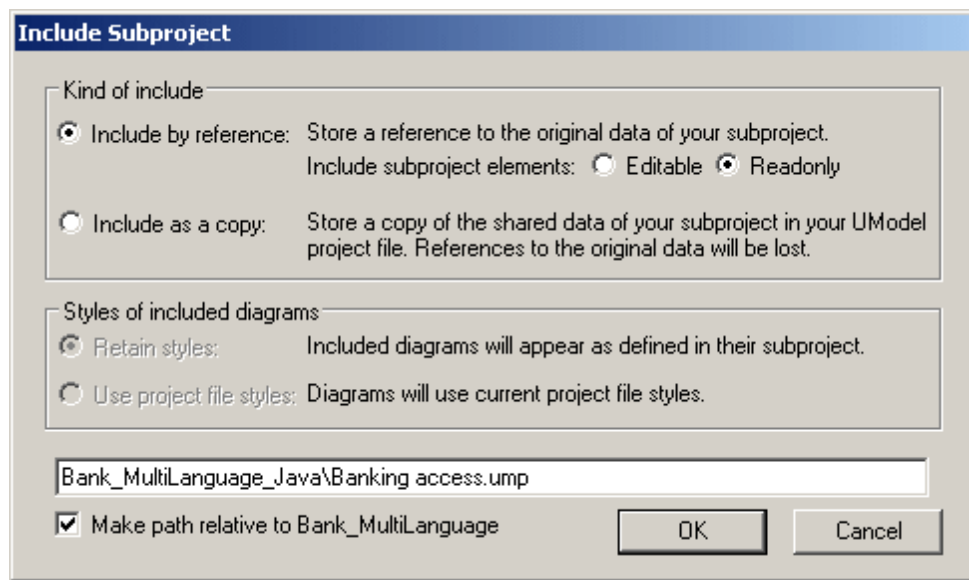
This starts another instance of UModel and opens the subproject as a "main" project. Any unresolved references are shown in the Messages window.



Reusing subprojects

Subprojects that have been split off from a main project can be used in any other main project(s).

1. Right click a project and select **Project | Include Subproject**.
2. Click the Browse button and select the *.ump file that you want to include, e.g. Banking access.ump.



3. Choose how the file is to be included; by reference for as copy.

Saving projects

When saving the main project file, all editable subproject files are also saved. I.e. all data contained in the shared packages of the subproject files are saved.

You should therefore not create/add data (components) outside of the shared/subproject structure, if the subproject is defined as "editable" in a main project file. If data exists outside of the subproject structure, a warning message will be displayed in the Messages window.

Saving subproject files

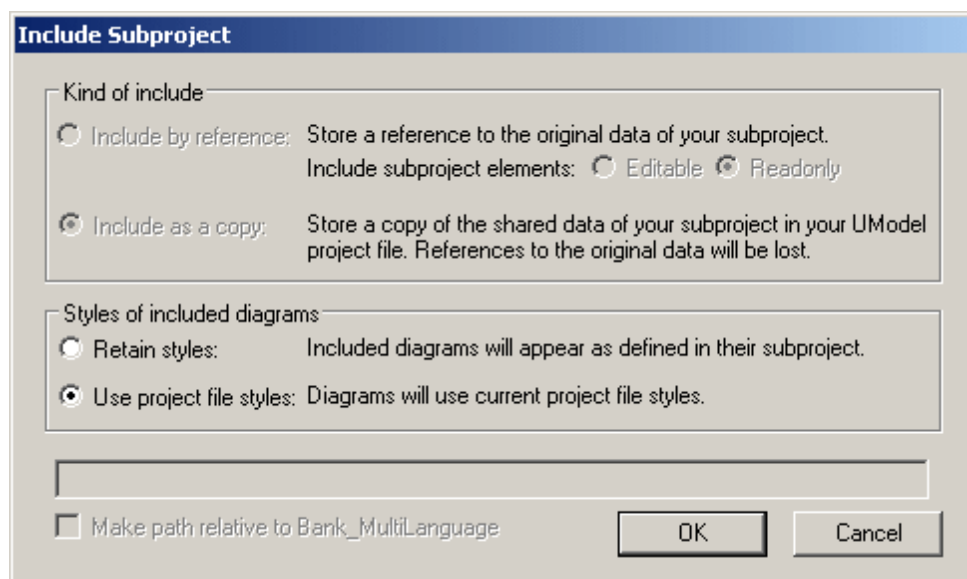
When saving subprojects (from the main project level) all references to sibling, as well as child subprojects, are considered and saved. E.g. if two sibling subprojects, sub1 and sub2, exist and sub1 uses elements from sub2, then sub1 is saved in such a way that it automatically saves references to sub2 as well.

If sub1 was opened as a "main" project, then it is considered as a self contained project and can be edited without any reference to the actual main project.

Reintegrating subprojects into the main project

Previously defined subprojects can be copied back into the main project again. If the subproject does not contain any diagrams then the reintegration will be immediate. If diagrams exist a dialog box will open.

1. Right click the subproject and select **Subproject | Include as Copy**.
This opens the "Include Subproject" dialog box, which allows you to define the diagrams styles you want to use when including the subproject.



2. Select the style option that you want to use, then click OK to include the project again.

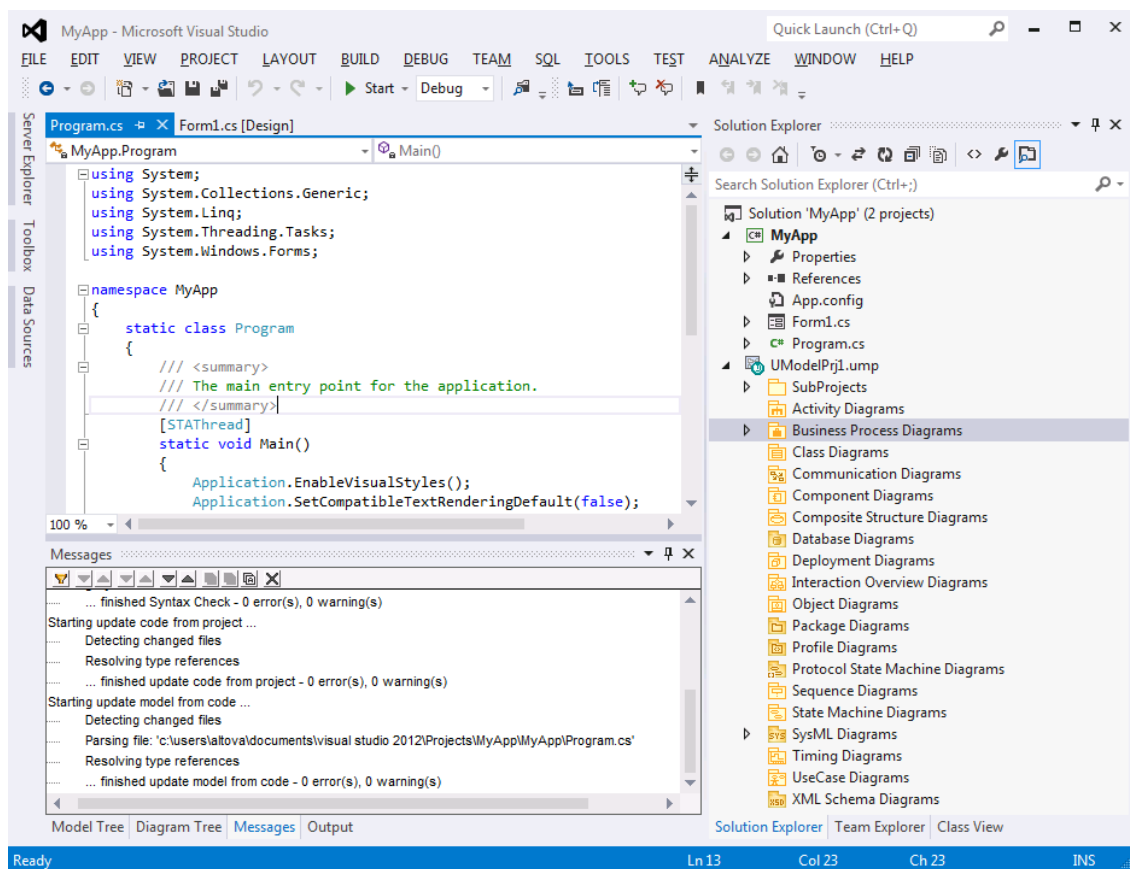
Chapter 14

UModel Plug-in for Visual Studio

14 UModel Plug-in for Visual Studio

You can integrate UModel 2016 into the Microsoft Visual Studio versions 2005/2008/2010/2012/2013/2015. This unifies the best of both worlds, combining the modeling capabilities of UModel with the development environment of Visual Studio.

One of the main benefits to using UModel as a Visual Studio plug-in is automatic synchronization between the C# or VB.NET code and the UML model. This means that, if you make changes to your code in Visual Studio, these are automatically propagated to the model. Likewise, if you make changes to the model (for example, by editing class diagrams), these would be propagated to the code. If necessary, you can disable automatic synchronization, and synchronize the code and the model manually (in either direction).



Sample Visual Studio 2012 project with UModel Plug-in support

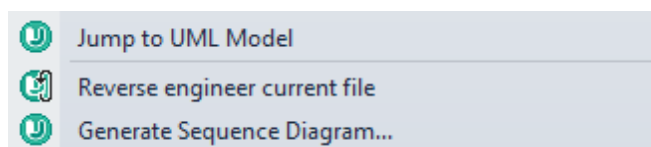
Compared to the standalone edition of UModel, the UModel plug-in for Visual Studio has the following behavior:

- Automatic synchronization between the UModel model and the project code is available, in either direction (see [Synchronizing the Model and Code](#)).
- The UModel functionality is accessible from the following menus:

File	Contains menu entries from both UModel and Visual Studio.

Edit	Contains menu entries from both UModel and Visual Studio.
View	The UModel-specific commands are grouped under View UModel .
Project	The UModel-specific commands are grouped under Project UModel .
Layout	Same as in the standalone edition of UModel.
Tools	Contains menu entries from both UModel and Visual Studio. The UModel options are available under Tools UModel options .
Help	The UModel help is available under Help UModel Help .

- When the cursor is in the Visual Studio code editor, the following new context menu items are available (in contexts where these commands are meaningful):
 - Jump to UML Model
 - Reverse engineer current file
 - Generate Sequence Diagram...



On the other hand, when the cursor is inside an element in the Model Tree window, the **Jump to Code** context menu item is available (in contexts where this command is meaningful).

- When UModel runs as a Visual Studio plug-in, you can use the version control functionality available in Visual Studio. The source control commands from the standalone edition of UModel available through the Microsoft Source Control Plug-in API are not supported.
- The dialogs triggered by the commands **UModel | Import Source Directory** and **UModel | Import Source Project** do not have the option to select "C#" and "Visual Basic" in the Language combo box. Import of existing projects is done through Visual Studio commands (for example, **File | Add | Existing Project**).
- The Scripting Editor (**Tools | Scripting Editor**) and the menu option **Tools | Restore Toolbars and Windows** are not supported.

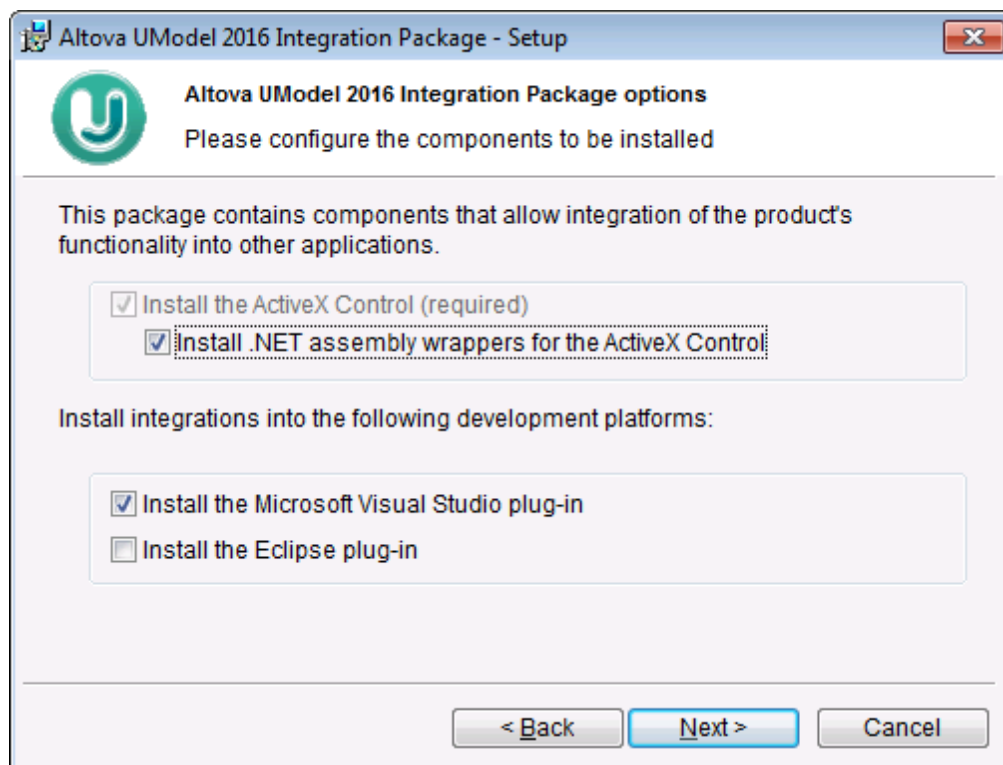
14.1 Installing the UModel Plug-in for Visual Studio

Prerequisites:

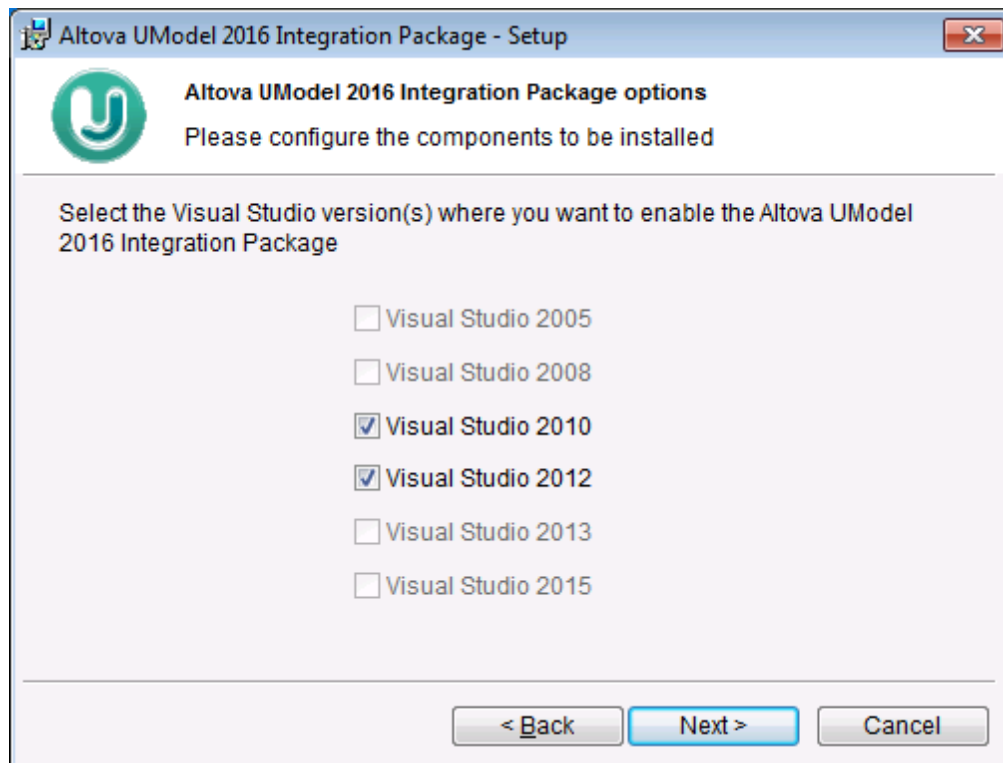
- Microsoft Visual Studio 2005/2008/2010/2012/2013/2015
- UModel (Enterprise or Professional Edition)
- UModel Integration Package, available for download at <http://www.altova.com/download/umodel.html>.

Note: To use UModel as a Visual Studio plug-in, install the 32-bit version of both UModel and the UModel integration package, since there is currently no support for 64-bit plug-ins in Visual Studio.

To enable the UModel plug-in for Visual Studio, download and run the UModel Integration Package and follow the on-screen installation instructions. During installation, ensure that the **Install the Microsoft Visual studio plug-in** option is selected.



When prompted during installation, select the Visual Studio version(s) where the plug-in should be enabled, for example:



Note: Only the Visual Studio versions installed on your operating system are available for selection.

Once the integration package has been installed, the UModel functionality becomes available in Visual Studio.

Enabling the UModel plug-in manually

It is possible that the plug-in was not automatically enabled during the installation process. To enable it, do the following:

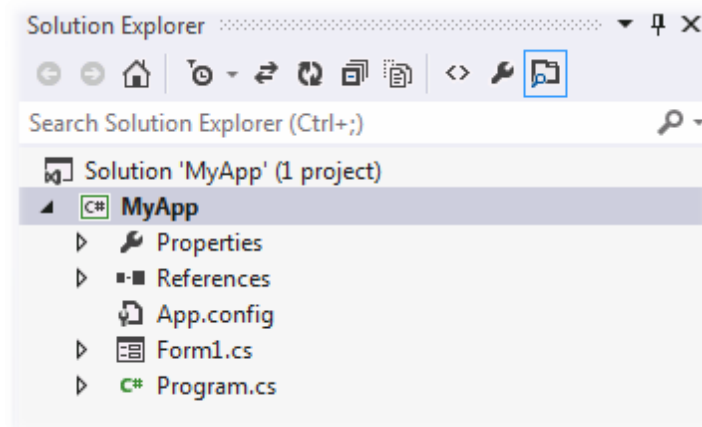
1. Navigate to the directory where the Visual Studio IDE executable is installed (for example, `c:\Program Files\Microsoft Visual Studio 8\Common7\IDE`).
2. Run the command prompt as administrator and enter `devenv.exe /setup`.
3. Wait for the process to terminate normally before starting to use the application within Visual Studio.

14.2 Adding UModel Support to Visual Studio Projects

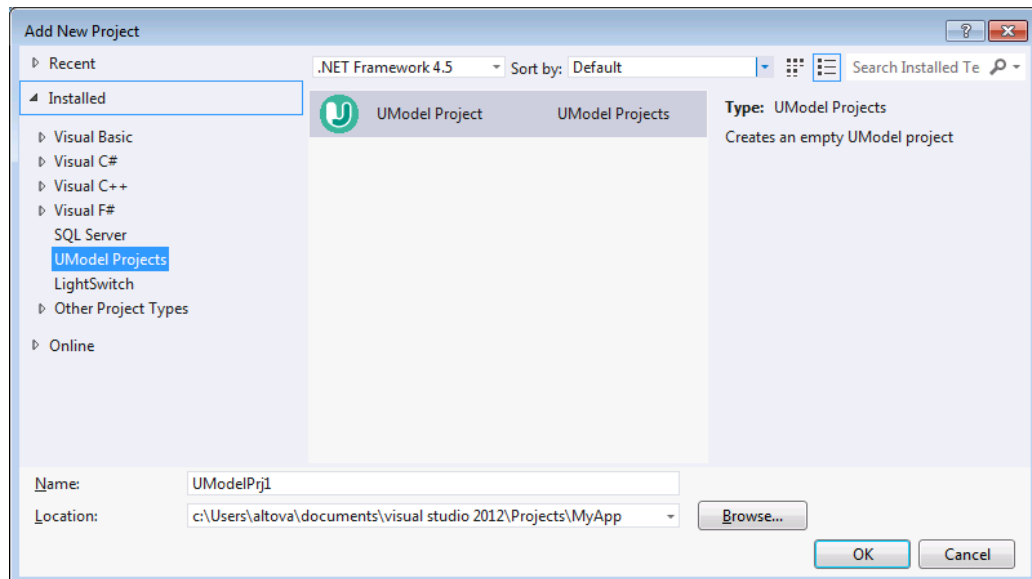
Adding UModel support to new or existing Visual Studio projects enables you to set up automatic synchronization between your Visual Studio project and the UModel model. A Visual Studio solution can contain one UModel project (not more).

To add UModel support to a Visual Studio project:

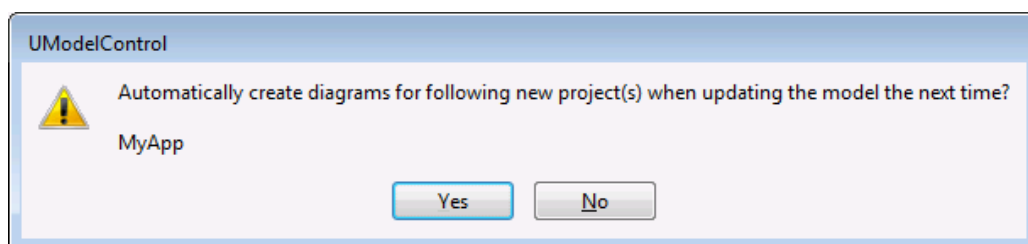
1. Create a new Visual Studio project, or open an existing one. (In this example, a new C# project called "MyApp" has been created with Visual Studio 2012).



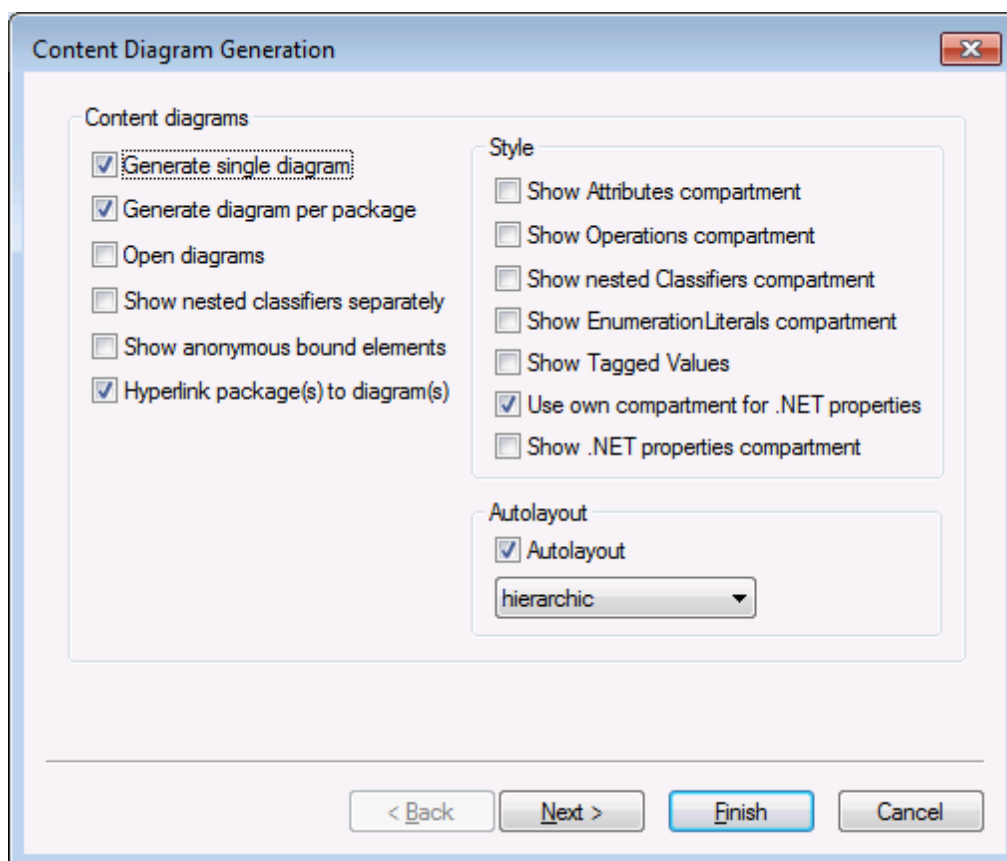
2. On the **File** menu, click **Add**, and then click **New Project**.
3. Select **UModel Projects**, and click **OK**.



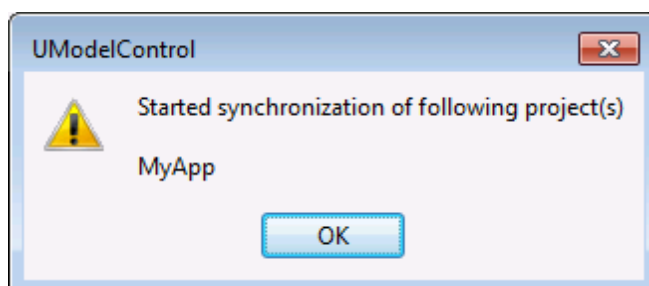
4. If you want diagrams to be created automatically in the model based on the code, click **Yes** when prompted (this is the recommended option).



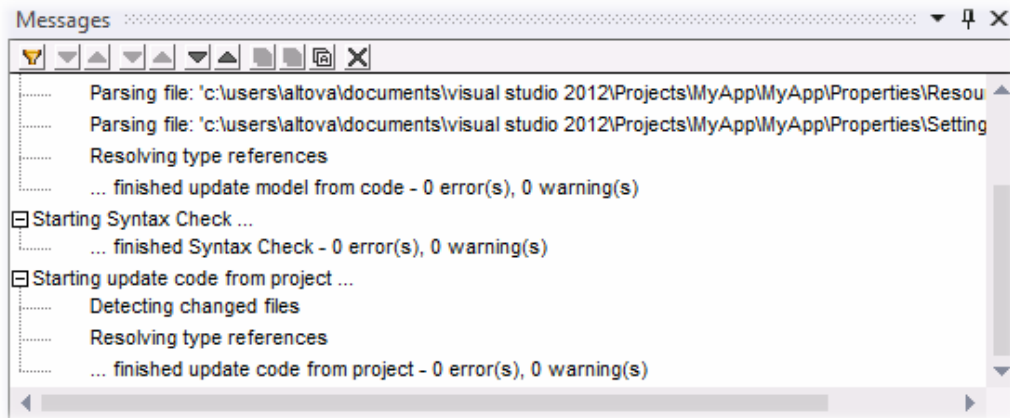
5. When prompted to select the diagrams generation options, choose your preferences as you go through the wizard steps, and click **Finish**. These steps are the same as in the standalone edition of UModel.



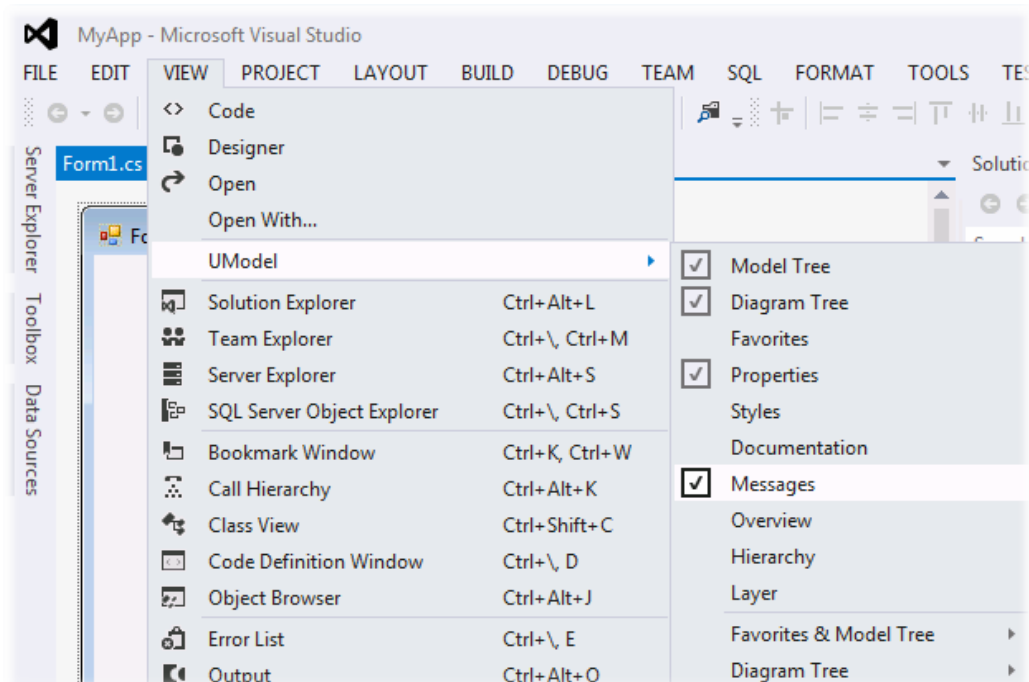
When you click **Finish**, UModel starts the synchronization process and notifies you of this. Click **OK** to close this message box.



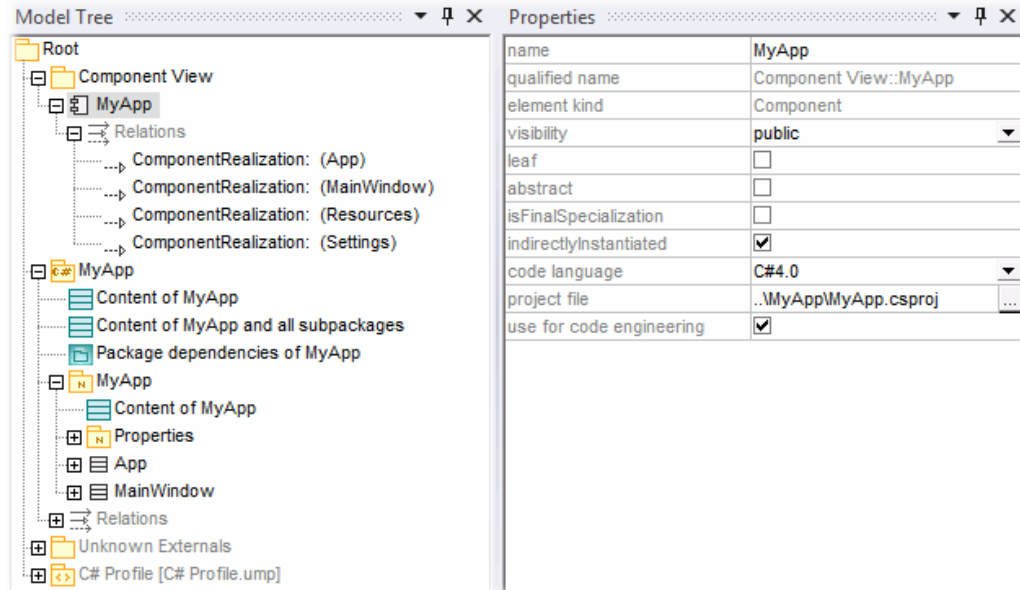
The synchronization details are displayed in the **Messages** window.



Note that the **Messages** window might not be visible by default in Visual Studio. You can display this window (and all other UModel-specific windows) by selecting the menu command **View | UModel | [Name of the window]**.



When you add a new UModel project to a Visual Studio solution, the settings required for code engineering (such as the component realization, and the C# or VB.NET profile) are defined automatically. To view these settings, open the **Model Tree** and the **Properties** windows.



14.3 Loading/Unloading UModel Projects

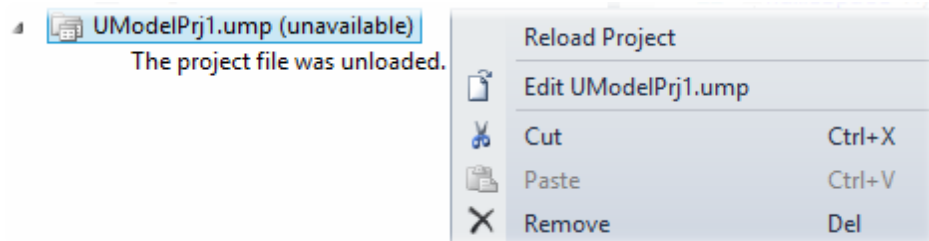
After you add an UModel project to a Visual Studio solution, it appears in the **Solution Explorer** of Visual Studio along with any other projects that are part of the solution. If necessary, you can temporarily unload the UModel project from the solution. When an UModel project is unloaded from the solution, its files remain on the disk, and in the **Solution Explorer**. This way, you can reload the project later back into the solution at a later time.

To unload an UModel project from a Visual Studio solution:

- On the **Project** menu, click **Unload project**.

To reload the UModel project back into the solution:

- Right-click the project in Solution Explorer, and click **Reload Project**.



To remove the UModel project from the Visual Studio solution:

- Unload the project, as shown above.
- Right-click the project in Solution Explorer, and click **Remove**.

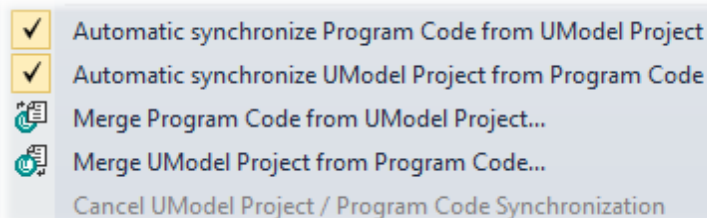
14.4 Synchronizing the Model and Code

The synchronization process between the UModel .ump file (the model) and the C# or VB.NET code can be manual or automatic.

Automatic synchronization takes place once you add UModel support to your Visual Studio project (see [Adding UModel Support to Visual Studio Projects](#)). Automatic synchronization means that, whenever you edit the code, the UModel Plug-in for Visual Studio parses the code and updates the model. Likewise, if you make changes to the model (for example, by editing a diagram), the code is updated accordingly. Manual synchronization, on the other hand, is initiated on demand, as shown below.

Both the automatic and the manual synchronization update changes in bulk, for the entire project. The option to merge or update a single class is not available in the Model Tree of Visual Studio.

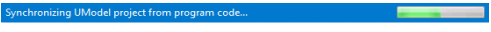
The commands which control automatic or manual synchronization are available in the **Project | UModel** menu:



Automatic synchronize Program Code from UModel Project	This menu option is switched on by default, meaning that synchronization from model to code is set to take place automatically. To enable or disable automatic synchronization, click the menu item.
Automatic synchronize UModel Project from Program Code	Same as above, in the opposite direction (from code to model).
Merge Program Code from UModel Project	Updates the program code with changes made in the UModel project (same functionality as in the standalone version). The name of this command changes to Overwrite Program Code from UModel Project, if you have set this option from Project UModel Project Synchronization Settings.
Merge UModel Project from Program Code	Updates the UModel project with changes made in the program code (same functionality as in the standalone version). The name of this command changes to Overwrite UModel Project from Program Code, if you have set this option from Project UModel Project

	Synchronization Settings.
Cancel UModel Project / Program Code Synchronization	Enables you to cancel a synchronization operation which is in progress. When no synchronization operation is in progress, this option is disabled.

During synchronization, the progress of the operation appears in the Visual Studio status bar, for example:

Visual Studio 2010	
Visual Studio 2012	

Code synchronization between code and mode cannot take place in the following cases:

- Code is not parseable
- The last reverse engineering or forward engineering process encountered an error.
- The syntax check throws an error in UModel.

In such cases, the error details are displayed in the **Messages** window. To open the source file which contains the error, click the corresponding line in the **Messages** window. The cursor will be positioned on the line containing the error.

Automatic synchronization limitations

Some C# and VB.NET code modifications in Visual Studio do not trigger an internal Visual Studio event and are thus not automatically updated in UModel. In such cases, you can either perform a forced synchronization manually, or make a different modification which triggers a source file update. Manual synchronization is necessary when adding or changing the following entities:

- Default values for attributes
- Default values for operation parameters
- TemplateParameters
- TemplateBindings
- Summary section for all elements
- Remark section for all elements
- All changes in method bodies

Note that if you change any of the above-mentioned modeling elements in the model, automatic code synchronization will take place normally. There are no limitations when automatic synchronization is from model to code.

To perform a forced manual synchronization from code to model, right-click the source code file in the code editor and select **Reverse engineer current file** from the context menu.

If your UModel project contains the language profile for Java, then automatic synchronization is automatically disabled for that project in Visual Studio, and a message box informs you of this. Such projects must be synchronized manually (using the menu commands **UModel | Merge Program Code from UModel Project**, and **UModel | Merge UModel Project from Program Code**). Alternatively, consider using the UModel Plug-in for Eclipse (see [UModel Plug-in for Eclipse](#)).

Chapter 15

UModel Plug-in for Eclipse

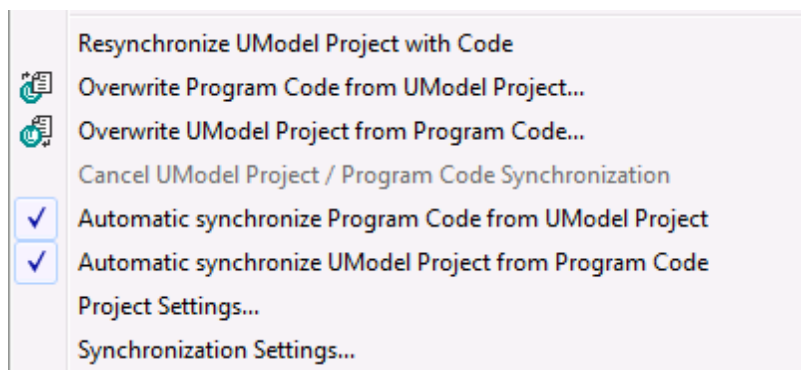
15 UModel Plug-in for Eclipse

Eclipse is an open source framework that integrates different types of applications delivered in form of plug-ins. The UModel plug-in for the Eclipse Platform allows you to access UModel functionality directly from Eclipse (versions 4.3 / 4.4 / 4.5), while also exposing some Eclipse-specific behavior discussed in this chapter.

One of the main benefits to using UModel as an Eclipse plug-in is automatic synchronization between the Java code and the UModel model. This means that, if you make changes to your Java code in Eclipse, these are automatically propagated to the model. Conversely, if you make changes to the model (for example, by editing class diagrams), these would be propagated to the code. If necessary, you can disable automatic synchronization, and synchronize the code and the model manually (in either direction).

Compared to the standalone version of UModel, the UModel plug-in for Eclipse has the following behavior:

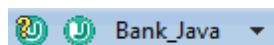
- In Eclipse, several graphical user interface elements conform to the specifics of the Eclipse development environment (see [The UModel Perspective](#)). Like in the standalone version, some user interface elements may be disabled or not available if the context is not relevant. For example, the UModel toolbar buttons are shown based on the kind of diagram active in the main editor.
- In Eclipse, a **UModel** menu is available—it corresponds to the **Project** menu in the standalone version of UModel. While most of the commands in this menu are not different to the standalone version, there are several new commands that enable you to control automatic synchronization:



Resynchronize UModel Project with Code	Enables you to explicitly initiate the synchronization between the UModel project and the program code (this may be the case when last automatic synchronization has failed due to any reason).
Merge Program Code from UModel Project	Updates the program code with changes made in the UModel project (same functionality as in the standalone version).
Merge UModel Project from Program Code	Updates the UModel project with changes made in the program code (same functionality as in the standalone version).

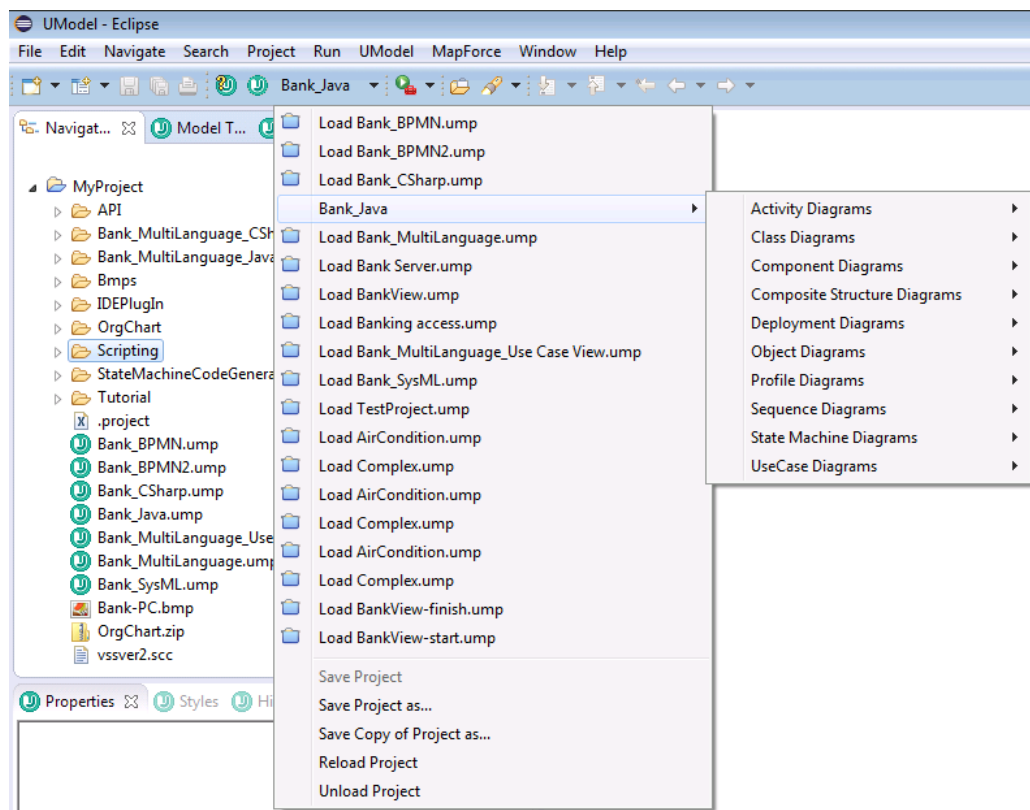
Cancel UModel Project / Program Code Synchronization	Enables you to cancel a synchronization operation which is in progress. When no synchronization operation is in progress, this option is disabled.
Automatic synchronize Program Code from UModel Project	This menu option is switched on by default, meaning that synchronization from model to code is set to take place automatically. To disable automatic synchronization, switch it off.
Automatic synchronize UModel Project from Program Code	Same as above, in the opposite direction (from code to model).

- The version control commands available in the standalone version of UModel through the Microsoft Source Control Plug-in API are not supported in Eclipse. Instead, you have the flexibility to use third-party version control systems that can integrate with Eclipse.
- The dialogs triggered by the commands **UModel | Import Source Directory** and **UModel | Import Source Project** do not have the option to select "Java" in the Language combo box. To import Java source code into an Eclipse project, use the standard Eclipse commands (for example, **File | Import**).
- In Eclipse, a new toolbar is available—the UModel toolbar, which contains some general as well as project-related commands.



The  toolbar button opens the help file. The  toolbar button displays the current status of the code engineering process (when it turns red this indicates an error, and you can view the details in the Messages view). Finally, the drop-down list in the toolbar has several functions:

- It enables you to quickly load or unload in Eclipse a particular UModel project (.ump) file. Your Eclipse project must include at least one UModel project (.ump) file; otherwise, the drop-down list is disabled.
- When a UModel project is loaded, it provides several contextual commands, including quick access to any of the diagrams of the loaded project:



Click to enlarge image

- The Scripting Editor (**Tools | Scripting Editor**) and the menu option **Tools | Restore Toolbars and Windows** are not supported.
- The UModel Help, Support Center, Check for Updates and About menus are available in the **Help | UModel Help** menu of Eclipse. The version information of the UModel Plug-in for Eclipse is also available from the Eclipse menu (select **Help | About Eclipse**, and then click the UModel icon).

15.1 Installing the UModel Plug-in for Eclipse

The prerequisites before installing the UModel Plug-in for Eclipse are as follows:

- UModel Enterprise or Professional Edition.
- Java Runtime Environment (JRE) 6.0 or later. Install a 32-bit or 64-bit JRE to match your version of UModel (32-bit or 64-bit).
- Eclipse Platform 4.3 / 4.4 / 4.5. Install a 32-bit or 64-bit Eclipse to match your version of UModel (32-bit or 64-bit).

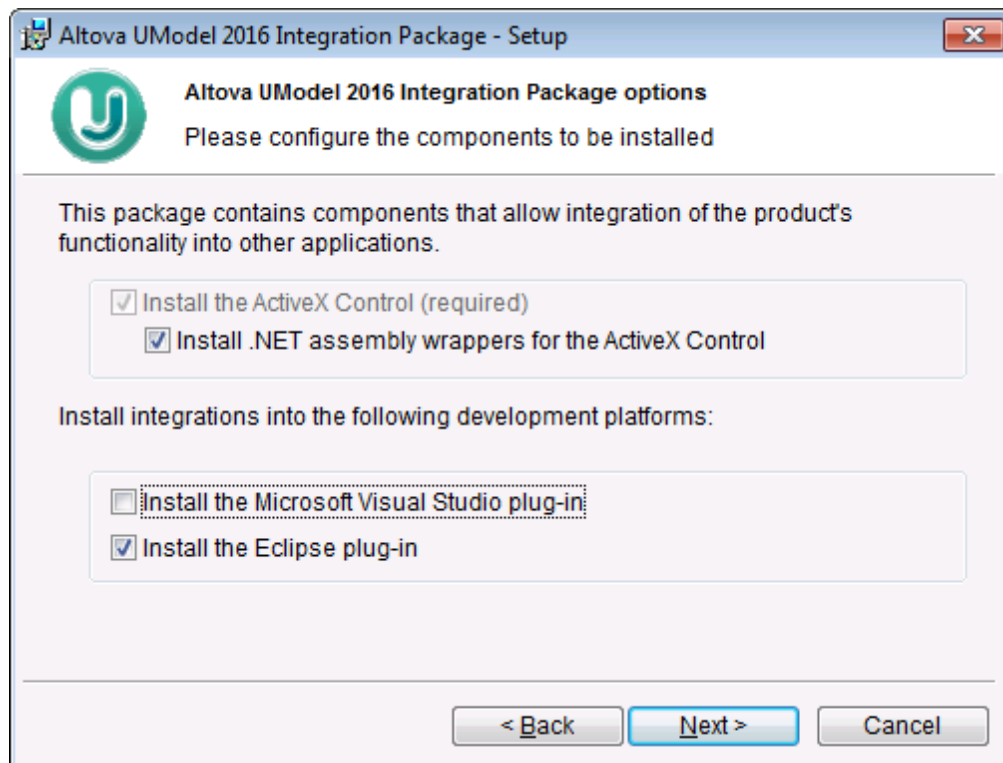
If you installed Eclipse 4.5 using the Eclipse installer, it is not possible to run on the same machine both the 32-bit and 64-bit versions of the UModel plug-in. This limitation originates in the Eclipse installer and does not apply if you install manually both versions of Eclipse (32-bit and 64-bit).

Installing the UModel Plug-in for Eclipse

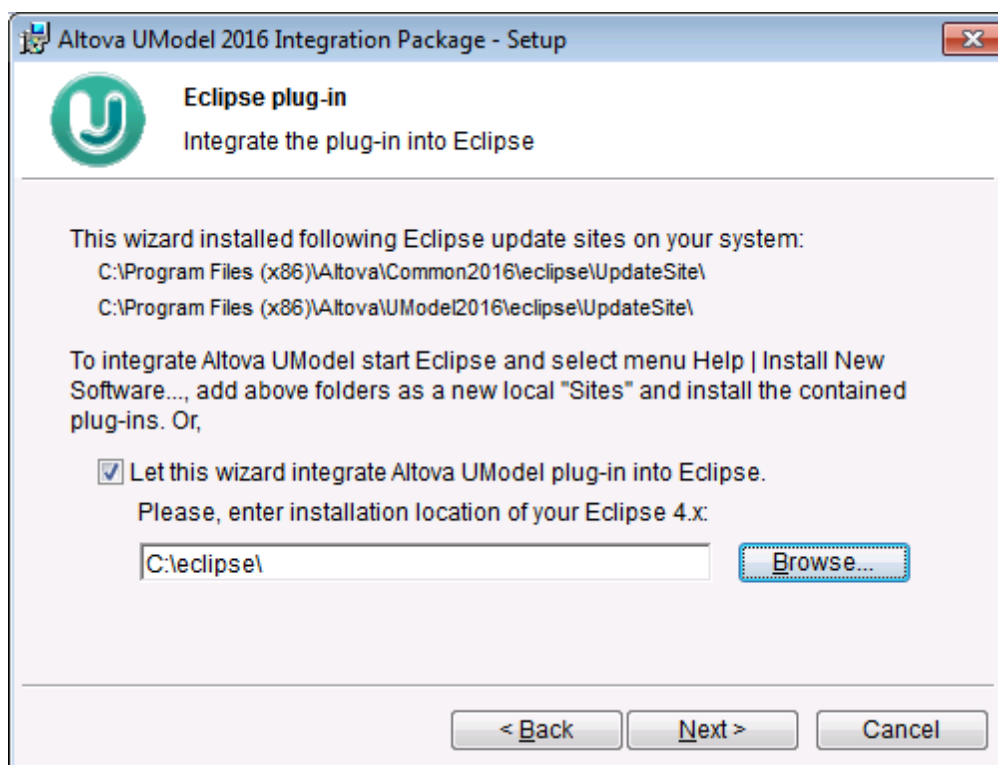
1. Download the UModel Integration Package from the Altova download page (http://www.altova.com/download_components.html).
2. Ensure that Eclipse is not running, and run the downloaded package.

Eclipse must be closed while you install or uninstall the UModel Integration Package.

3. During installation, select the **Install the Eclipse plug-in** option, and then click **Next**.



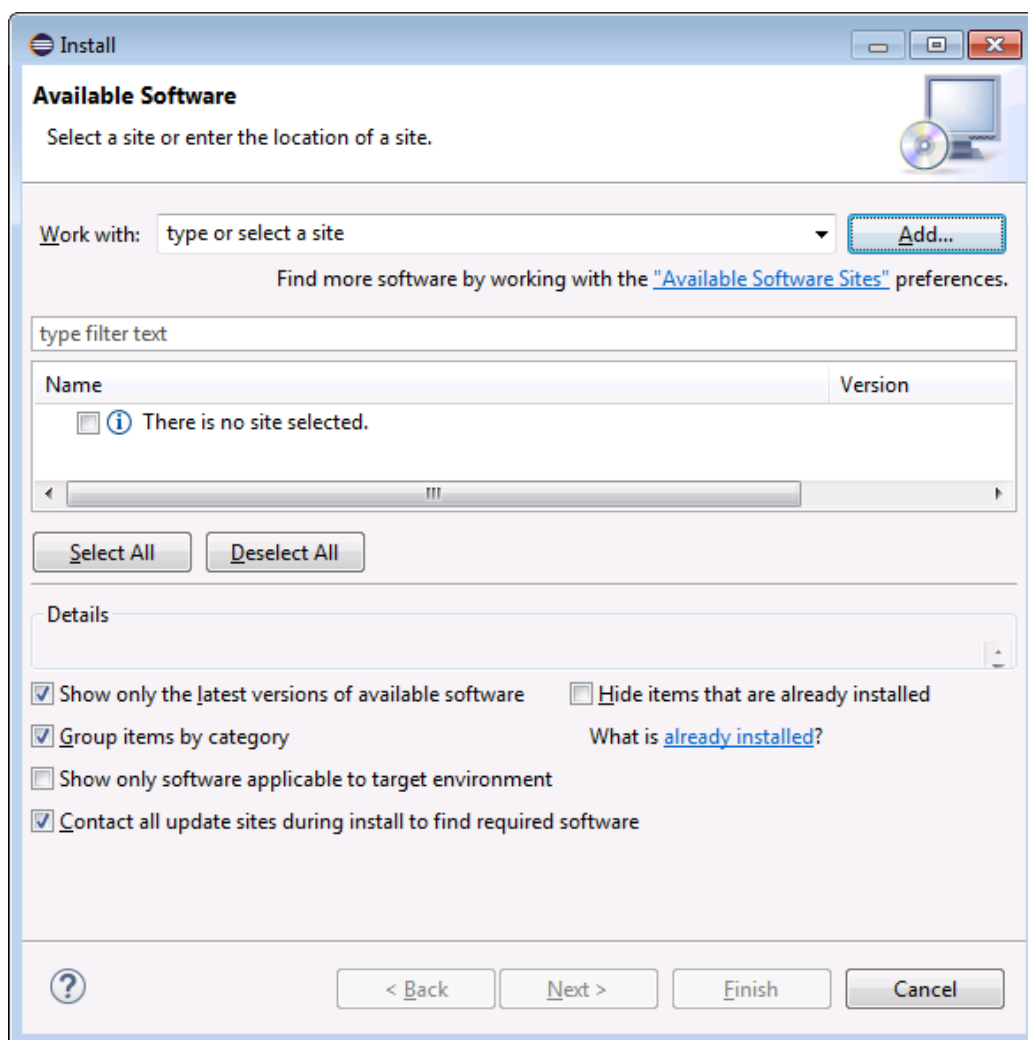
4. When prompted to choose how the plug-in should be integrated into Eclipse (automatically or manually), do one of the following:
 - a. To complete the plug-in installation automatically (this is the recommended option), select **Let this wizard integrate Altova UModel plug-in into Eclipse**, and browse for the directory where the Eclipse executable (`eclipse.exe`) is located.
 - b. To complete the plug-in installation separately in Eclipse, click to clear the **Let this wizard...** check box (see instructions below).



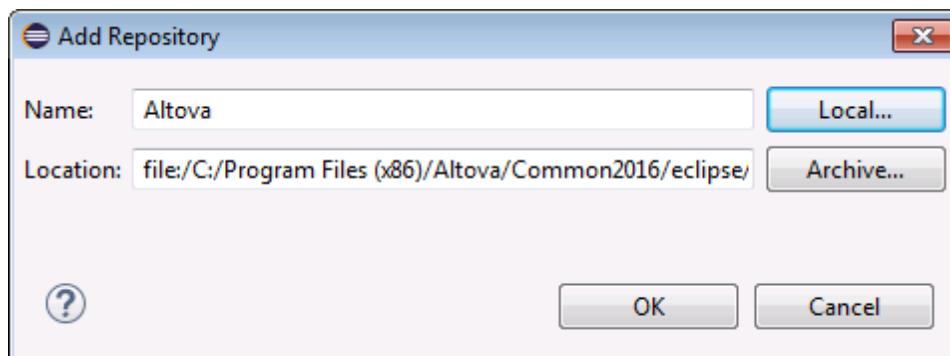
5. Click **Next**, and complete the installation. If you chose the automatic integration, the UModel perspective and menus become available in Eclipse next time when you start Eclipse.

Integrating the UModel plug-in manually

1. In Eclipse, click the menu command **Help | Install New Software**.
2. In the Install dialog box, click the **Add** button.

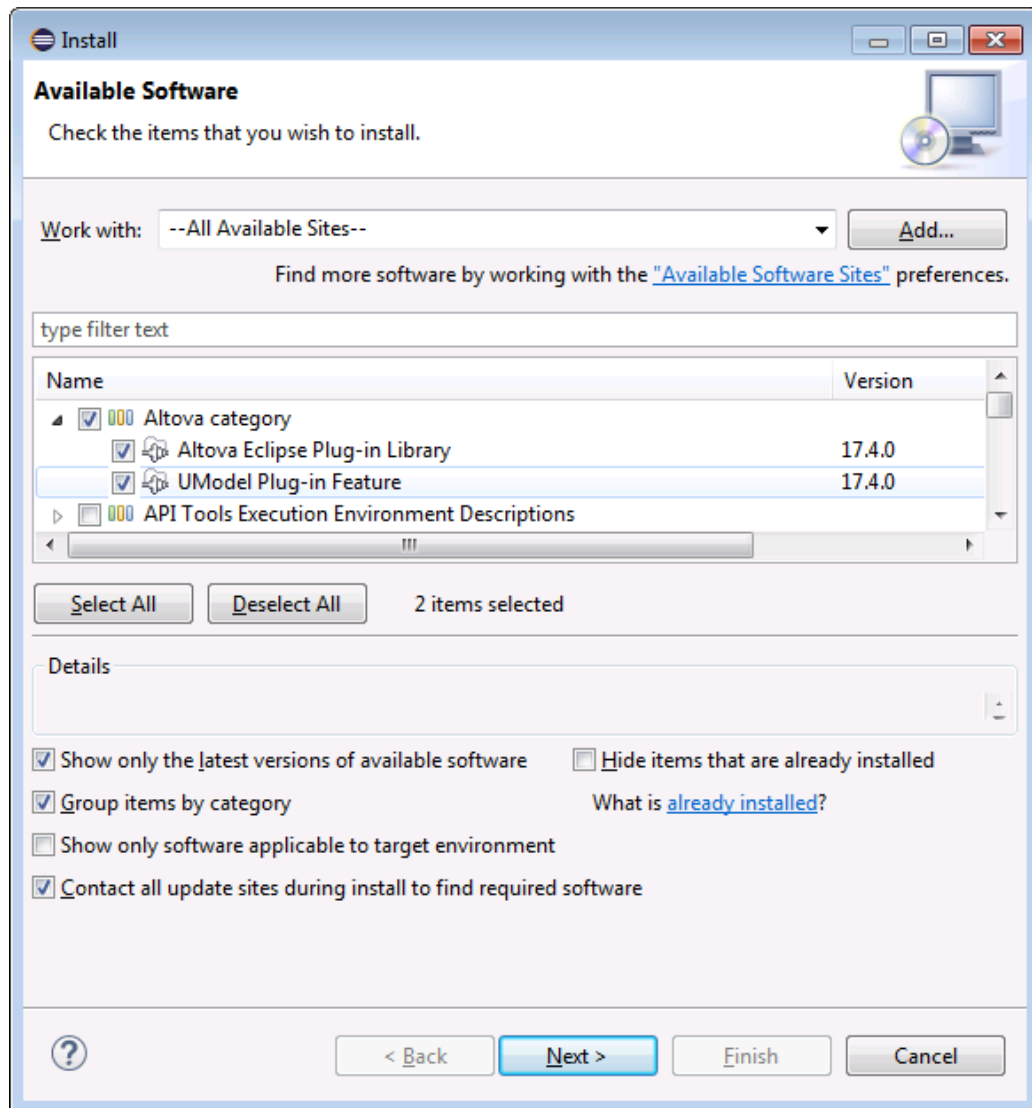


3. In the Add Repository dialog box, click the **Local** button.
4. Browse for the folder `c:\Program Files\Altova\Common2016\eclipse\UpdateSite`, and select it. Provide a name for the site (such as 'Altova'), and click **OK**.



5. Repeat Steps 2 to 4, this time selecting the folder `c:\Program Files\Altova\UModel2016\eclipse\UpdateSite`, and providing a name such as 'Altova UModel'.
6. In the *Work With* combo box of the Install dialog, select the option -- *All Available Sites* --

(see screenshot below). This causes all available plug-ins to be displayed in the pane below. Check the top-level check box of the *Altova* category folder (see screenshot below). Then click the **Next** button.

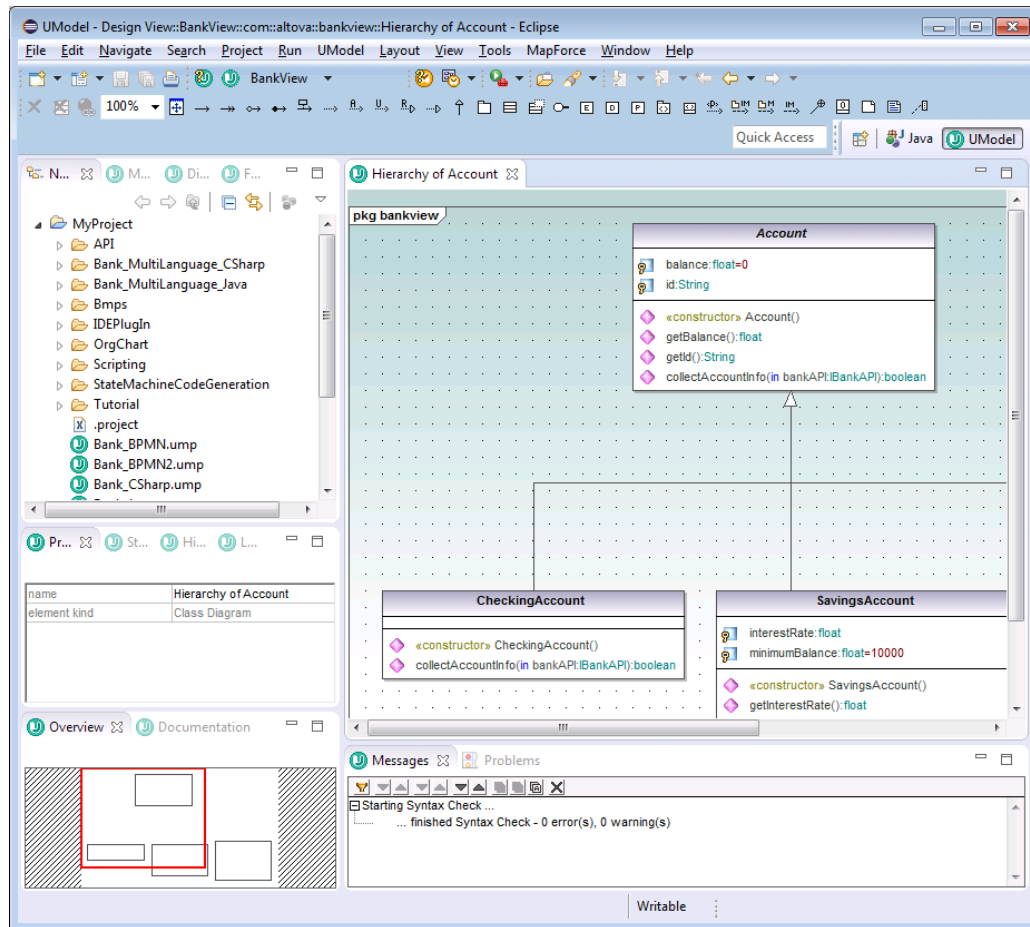


7. An *Install Details* screen allows you to review the items to be installed. Click **Next** to proceed.
8. In the *Review Licenses* screen that appears, select *I accept the terms of the license agreement*. No license agreement additional to your UModel Enterprise or Professional Edition license is required for the UModel plug-in. Then click **Finish** to complete the installation.

Note: If there are problems with the plug-in (missing icons, for example), start Eclipse from the command line with the `-clean` flag.

15.2 The UModel Perspective

After you install the UModel plug-in for Eclipse, a new perspective ("UModel") becomes available in Eclipse. By default, the UModel perspective resembles to some extent the graphical user interface of the standalone version of UModel. To switch to the UModel perspective, click **Window | Open Perspective | Other**, and choose UModel from the list. The image below illustrates a sample UModel project (BankView.ump) loaded into Eclipse, with the UModel perspective switched on.



The UModel perspective in Eclipse is organized as follows:

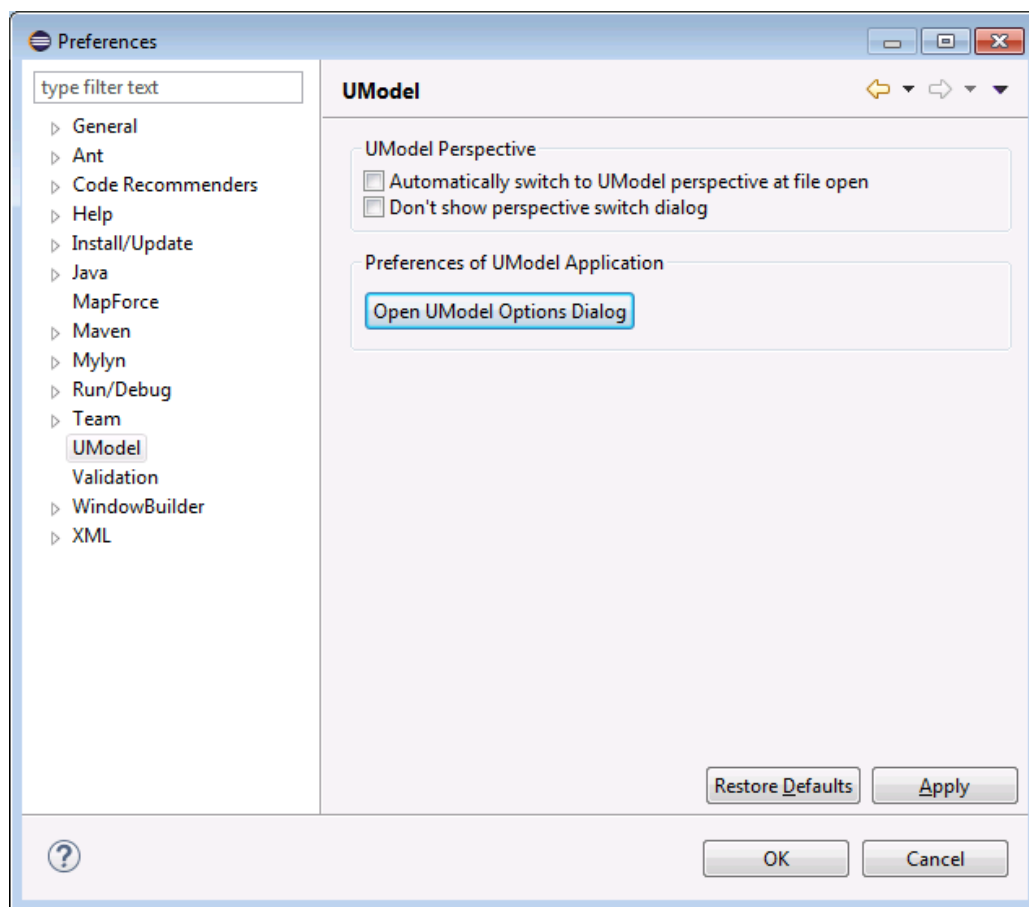
- The Diagram window is available as an Eclipse editor. Like in the standalone version, when there are multiple diagrams open, they are shown in individual editors.
- All of the following UModel windows are available as Eclipse views (by default, to the left of the main editor):
 - Diagram Tree
 - Favorites
 - Properties
 - Styles
 - Hierarchy
 - Overview
 - Documentation

- Layer
- Finally, the Messages window is also available as an Eclipse view (by default, under the main editor).

The UModel perspective behaves just like any other Eclipse perspective—you can switch to it whenever required using the menu command **Window | Navigation | Next Perspective**.

To configure the settings applicable to the UModel perspective:

1. On the **Window** menu, click **Preferences**.
2. On the Preferences dialog box, select **UModel**.



Click to enlarge image

To customize the appearance of the UModel perspective (toolbar visibility, menu visibility, and so on), switch to the UModel perspective, and then select the menu command **Window | Customize Perspective**. To revert to the default settings, select **Window | Reset Perspective**.

To display a particular view in the UModel perspective, switch to the UModel perspective, and then select the required view from the **Window | Show View** menu.

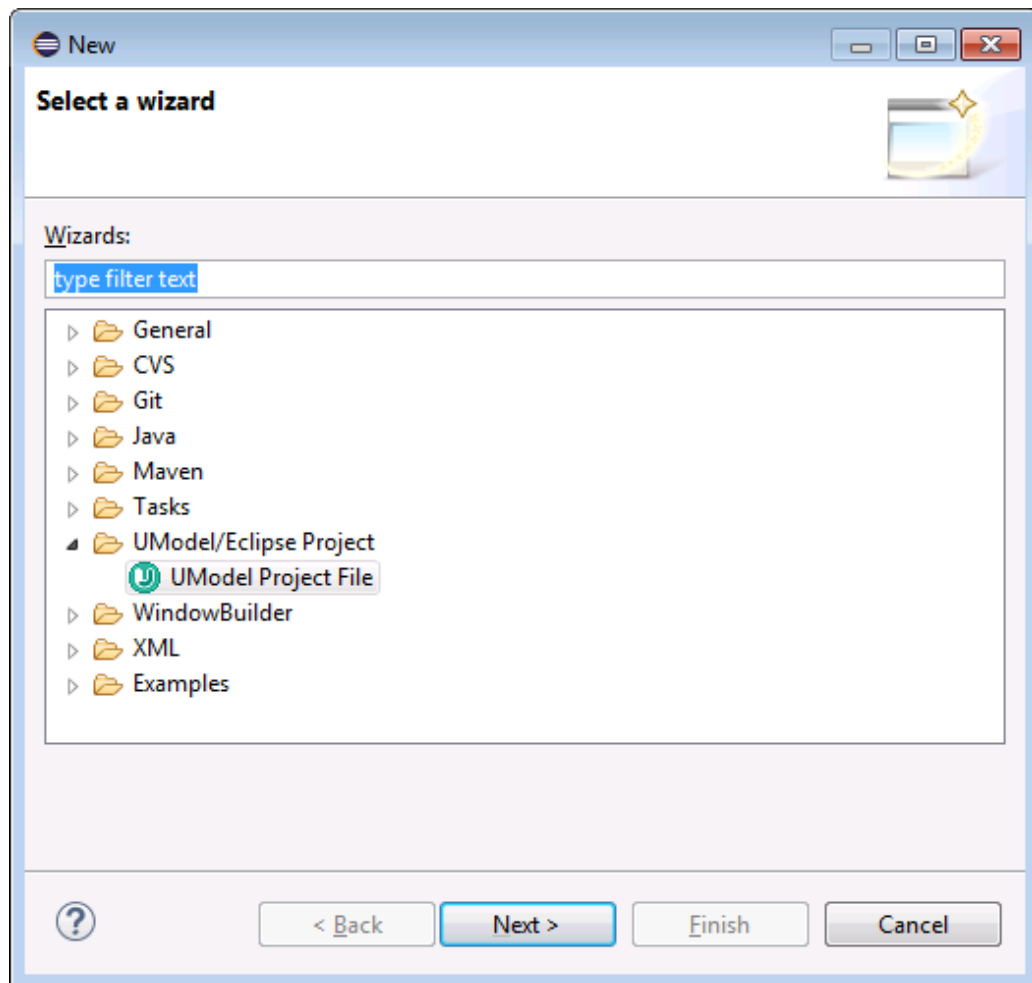
For general information about Eclipse perspectives, refer to the Eclipse documentation.

15.3 Adding UModel Support to Eclipse Projects

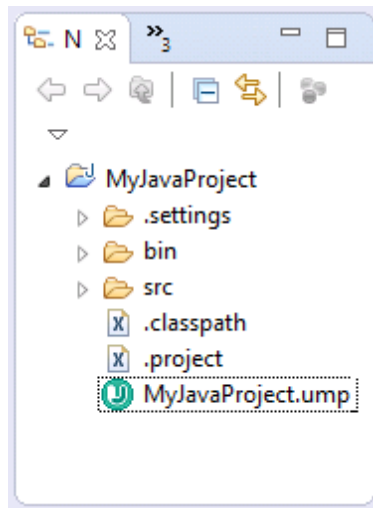
Before you can work with UModel projects (.ump file) in the Eclipse environment, make sure to create or open an Eclipse project first (this can be, for example, a new or existing Java project to which you would like to add UML support). This topic shows you how to create a new UModel project within an Eclipse project. For instructions on how to import an existing UModel project into an Eclipse project, see [Importing Existing UModel Projects](#).

To add a UModel project to an Eclipse project:

1. Create a new (or open an existing) Eclipse project, by using the standard Eclipse commands (**File | New | Project**, or **File | Open File**)
2. On the **File** menu, click **New | Other**, and then select the **UModel Project File** type from the dialog box.



3. Click **Next**.
4. When prompted, select a parent folder for the new UModel project, and click **Finish**. The new UModel project becomes available in the Navigator view, under the parent folder you specified.

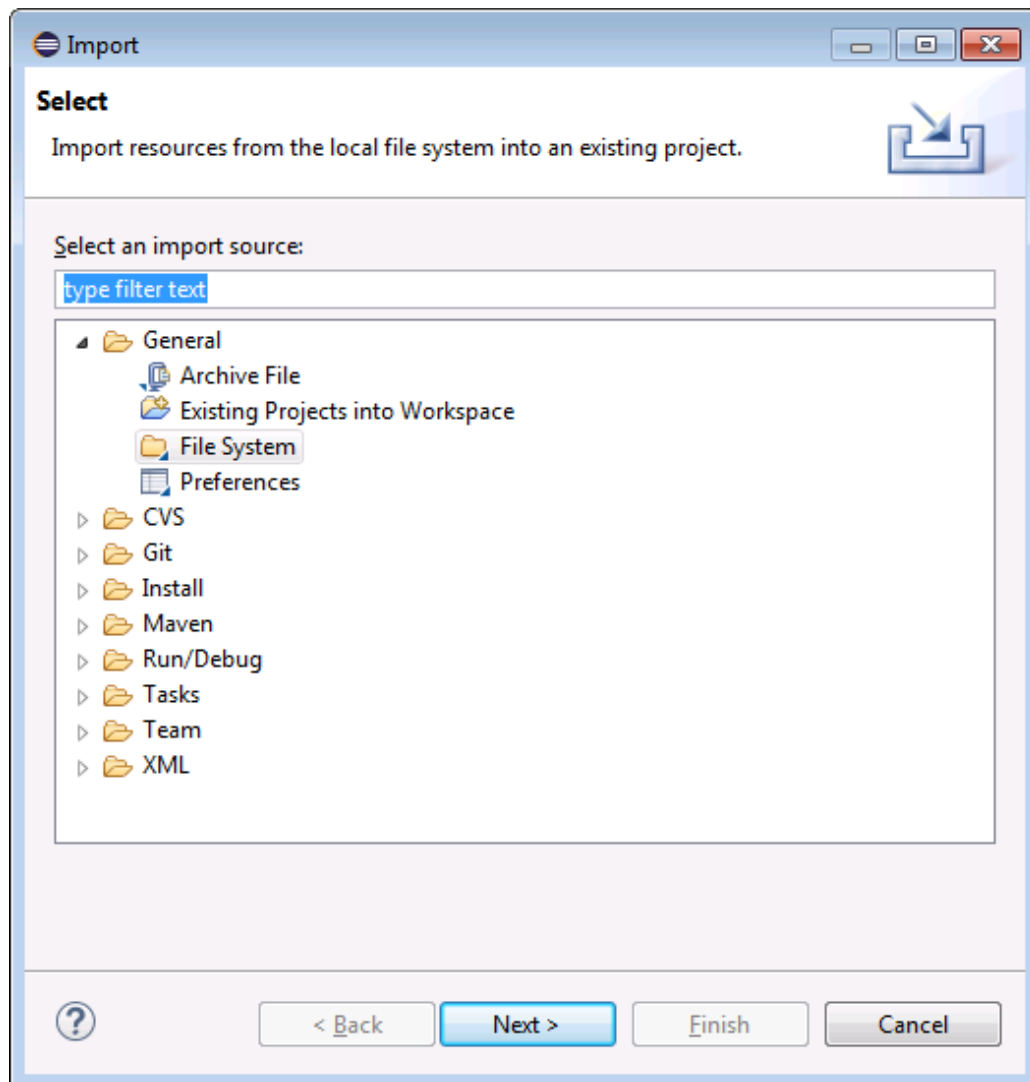


UModel projects cannot be opened in an editor. To take actions against the project (such as saving or loading its contents into Eclipse), right-click the .ump file, and select the required command.

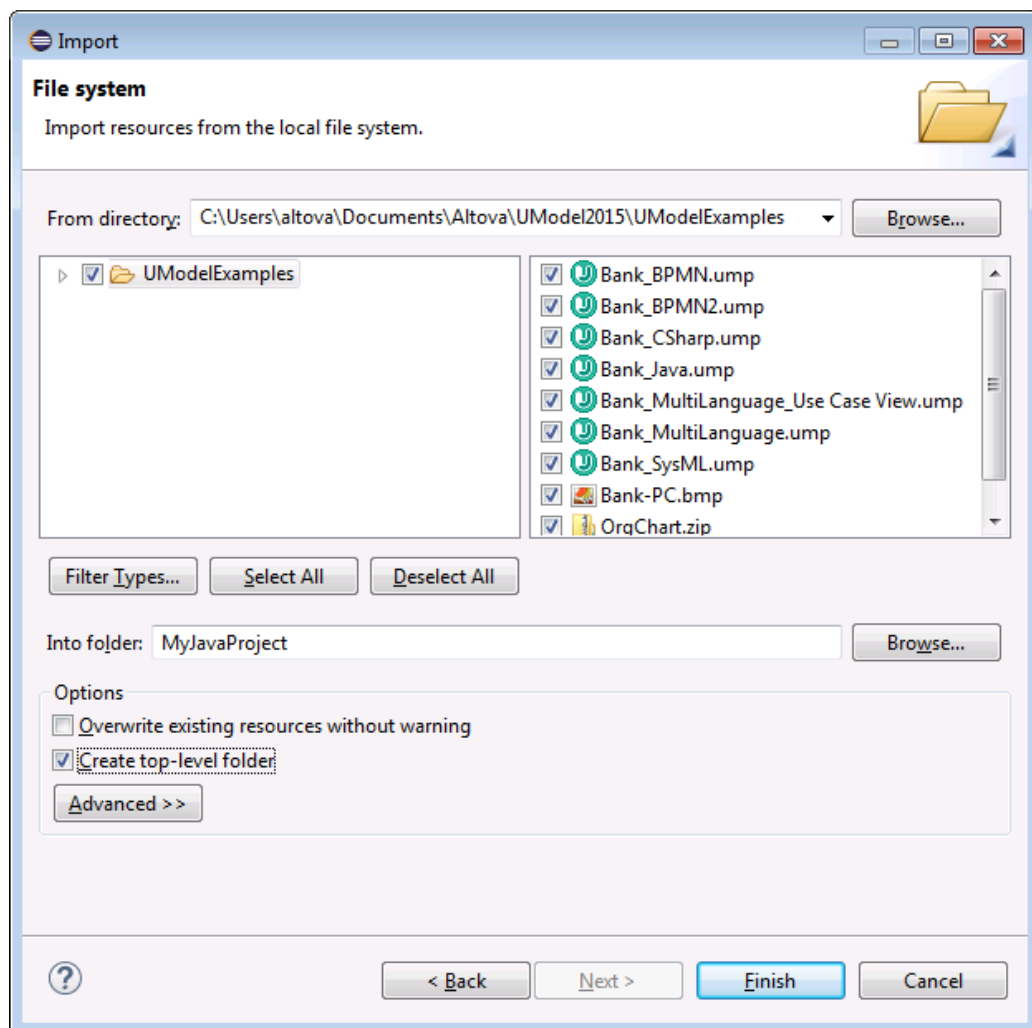
15.4 Importing Existing UModel Projects

To import existing UModel projects into Eclipse:

1. Create a new (or open an existing) Eclipse project.
2. On the **File** menu, click **Import**.
3. Select **General | File System**.



4. Click **Next**.
5. Click **Browse** and select the UModel project folders you want to import (for example, the UModel Examples folder).



6. Click **Finish**.

15.5 Loading/Unloading UModel Projects

After you have created or imported one or more UModel project files, they appear in the Navigator view of Eclipse. Even though an Eclipse project can contain multiple UModel project files, only one UModel project can be active (loaded) at the same time in Eclipse. You can load a specific project as follows:

- Right-click the file in the Navigator view, select **UModel | Load**.
- In the UModel toolbar, select **Load YourProjectName.ump**.

To unload a project:

- Right-click the file in the Navigator view, select **UModel | Unload**.
- In the UModel toolbar, select **Unload project**.

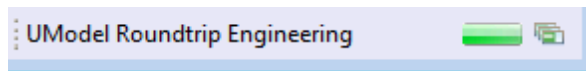
15.6 How Automatic Synchronization Works

Automatic synchronization takes place after you add UModel support to a Java project (see [Adding UModel Support to Eclipse Projects](#)). Automatic synchronization means that, whenever you edit the code in the Eclipse environment, the UModel Plug-in for Eclipse parses the code and updates the model. Likewise, if you make changes to a diagram in the model, the code is updated accordingly.

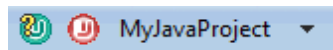
If your UModel project contains the language profile for C# or Visual Basic, then automatic synchronization is automatically disabled for that project, and a message box informs you of this. Such projects must be synchronized manually (using the menu commands **UModel | Merge Program Code from UModel Project**, and **UModel | Merge UModel Project from Program Code**).

Automatic or manual synchronization updates changes in bulk, for the entire project. The option to merge or update a single class is not available in the Model Tree.

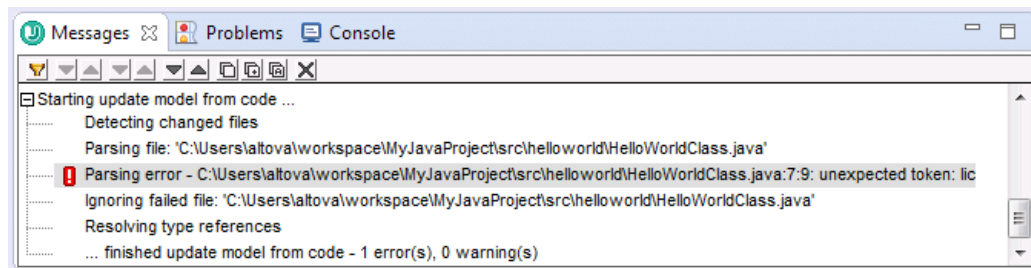
During synchronization, the progress of the operation appears in the Eclipse status bar.



If code is not parseable then the Code Engineering Status tool bar button turns red. This also happens if the last reverse engineering or forward engineering process encountered an error. The same is true if the syntax check throws an error in UModel.



The Messages view displays the error details.



To open the source file which contains the error, click the corresponding line in the Messages view. The cursor will be positioned on the line containing the error

15.7 Example: Setting up Automatic Synchronization

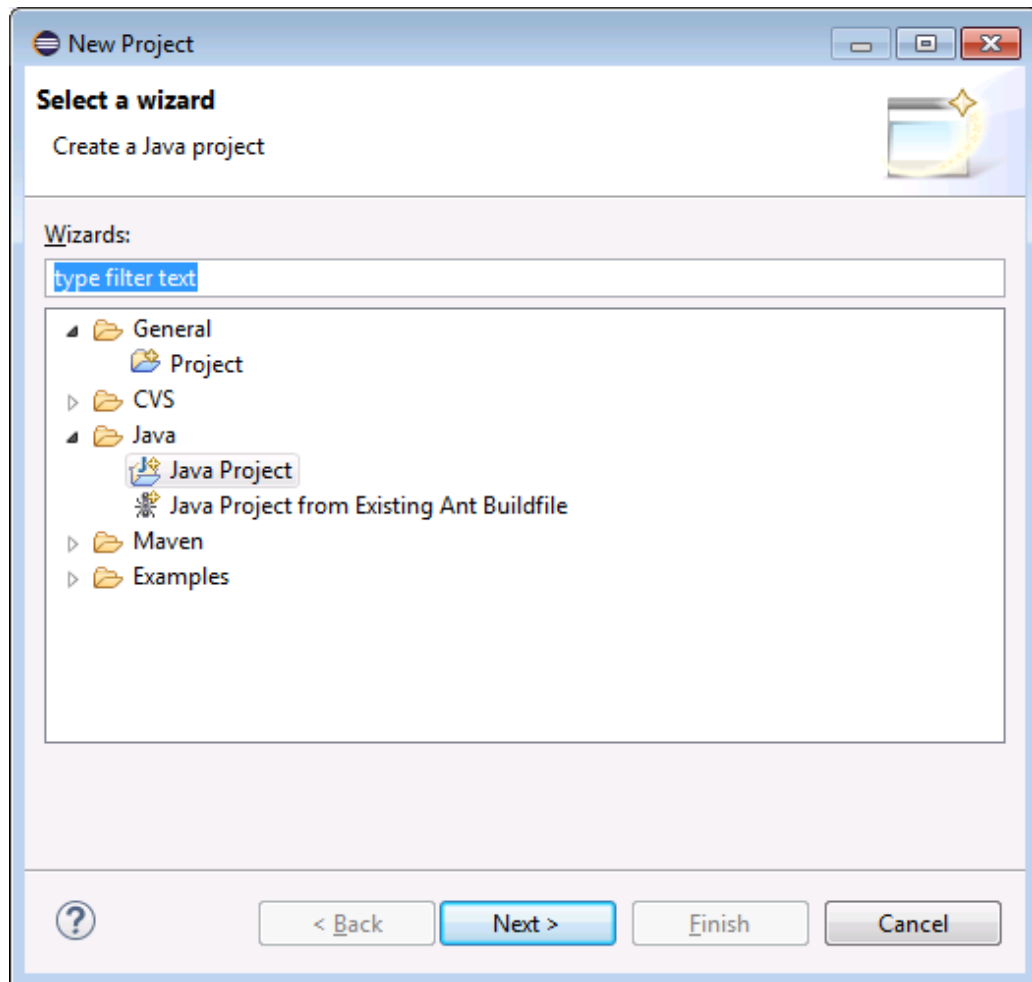
This tutorial illustrates how to set up automatic synchronization between a Java project and its corresponding UML model. Before you proceed, make sure that you have already installed the UModel plug-in for Eclipse, and the Java Development Kit (not just the Java Runtime Environment) required by Eclipse.

Step 1: Create a new Java project

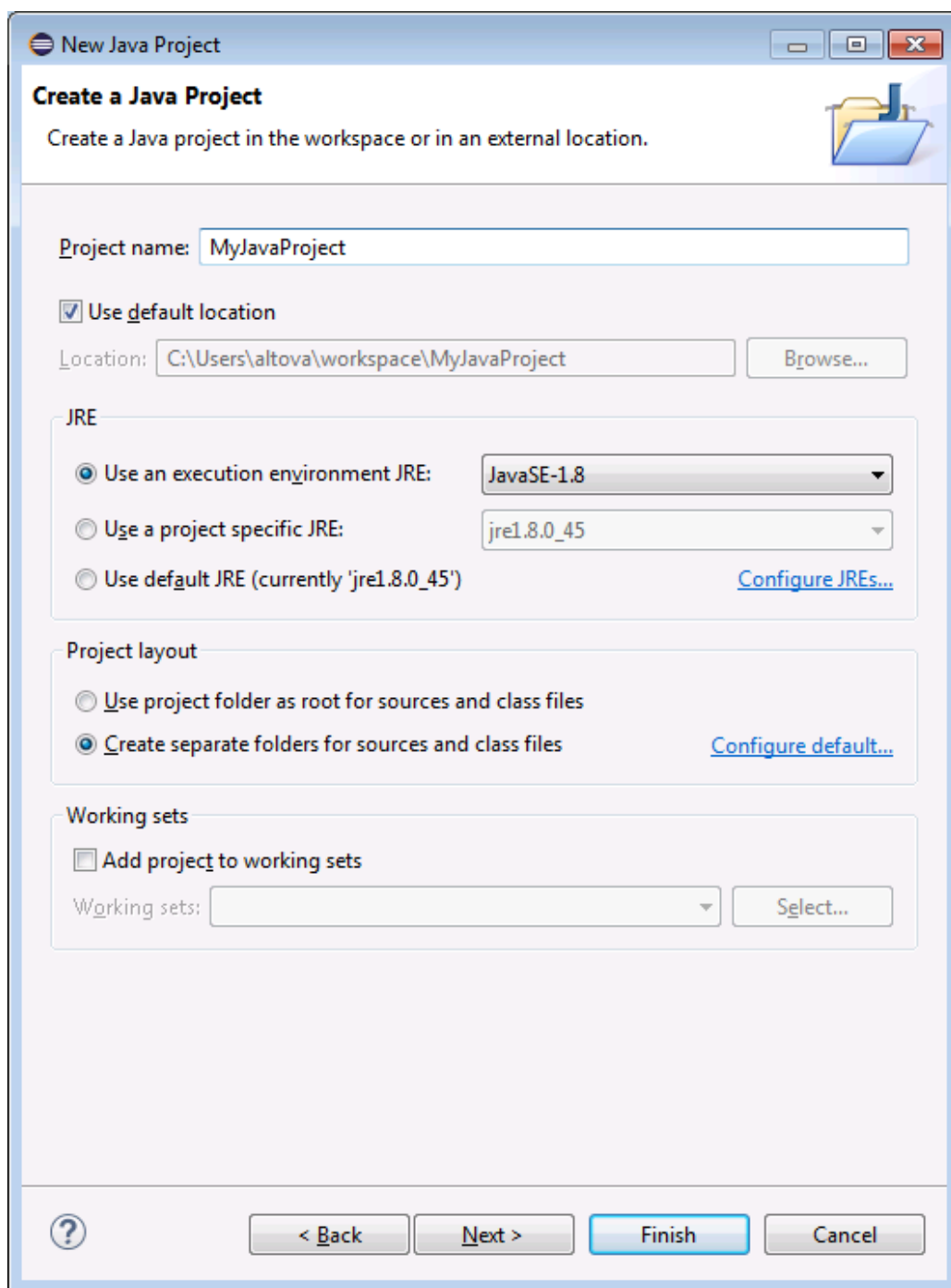
We will begin by creating a new Java project in Eclipse. For the scope of this example, this will be a simple application that displays the text "Hello, World" when it is run.

To create the "Hello, World" application:

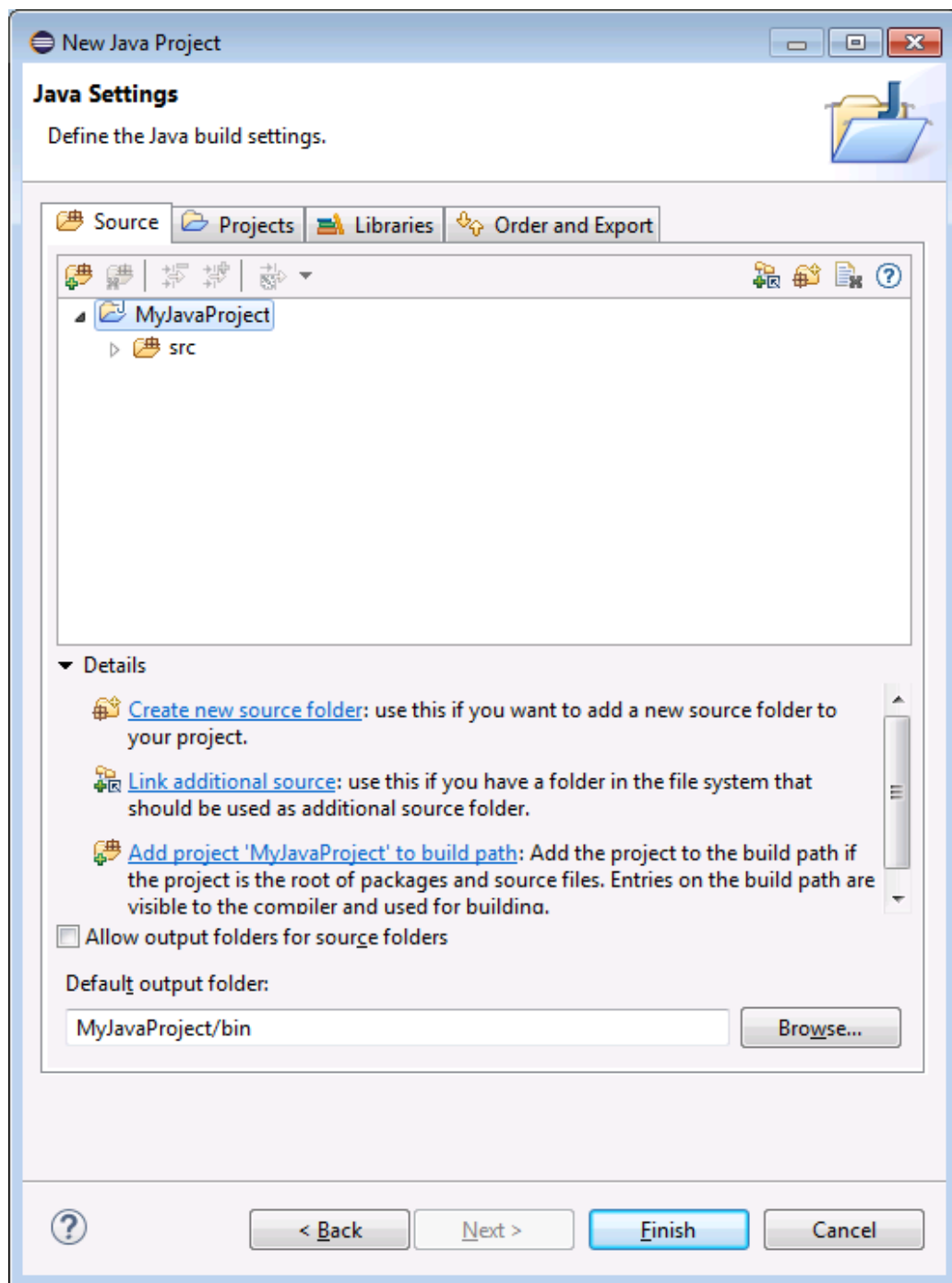
1. Start Eclipse and switch to the Java perspective.
2. On the **File** menu, click **New | Project**.



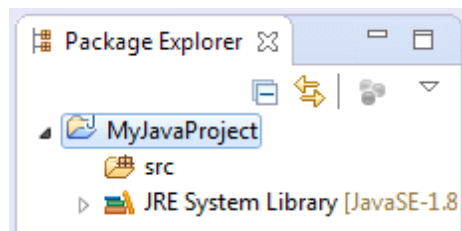
3. Select **Java | Java Project**, and then click **Next**.



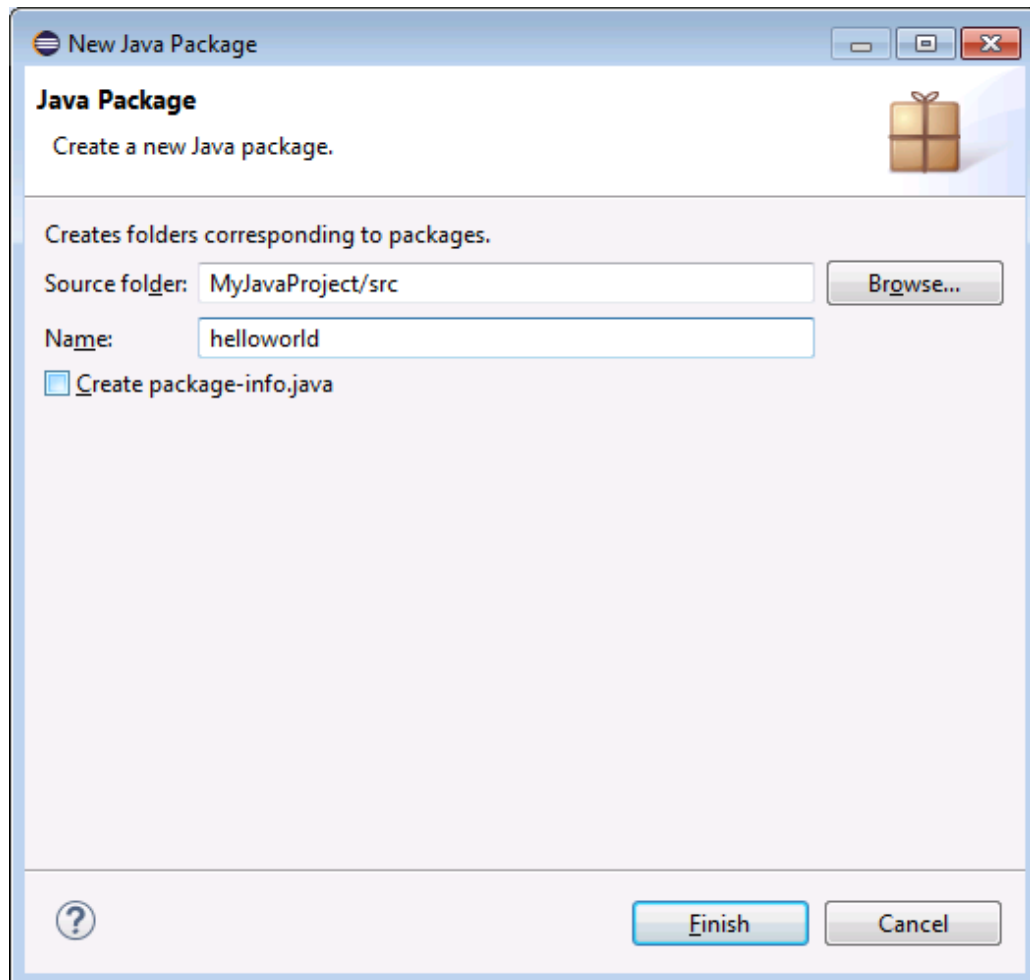
4. Enter "MyJavaProject" as project name, and then click **Next**.



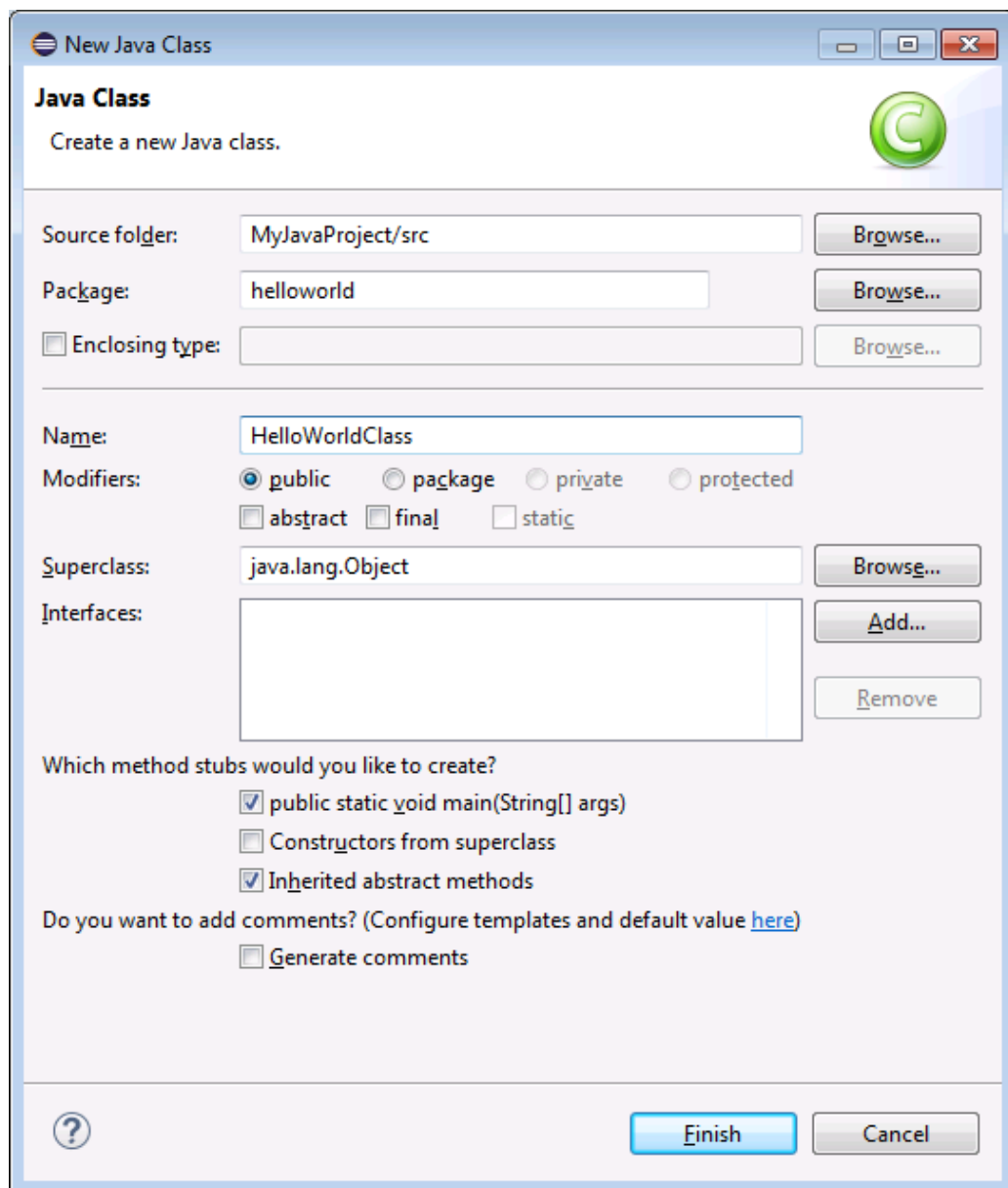
5. Leave the default settings as is, and click **Finish**. Your project now appears in the Package Explorer.



6. On the **File** menu, click **New | Package**.



7. Enter "helloworld" as package name, and click **Finish**.
8. On the **File** menu, click **New | Class**. Enter "HelloWorldClass" as class name, and make sure to select the **public static void main(String[] args)** option.



9. Open the class file, and add the following text to the body of the class:

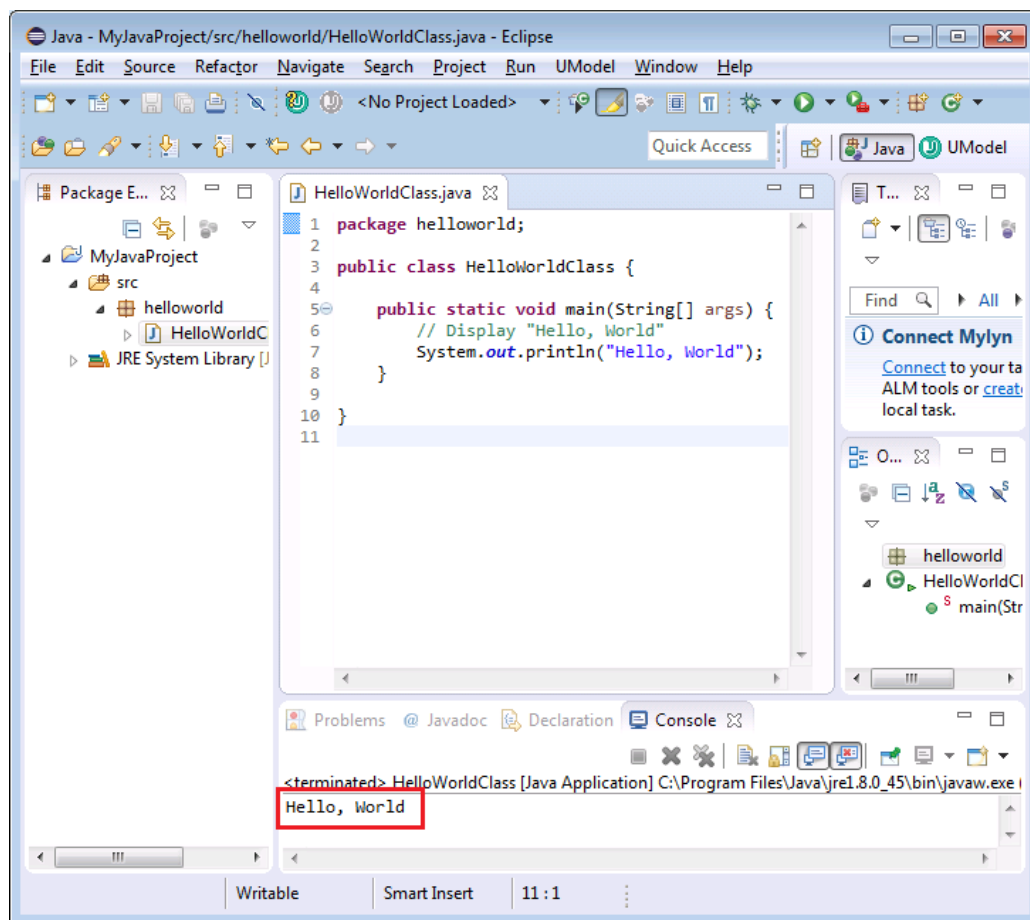
```
package helloworld;

public class HelloWorldClass {

    public static void main(String[] args) {
        // Display "Hello, World"
        System.out.println("Hello, World");
    }

}
```

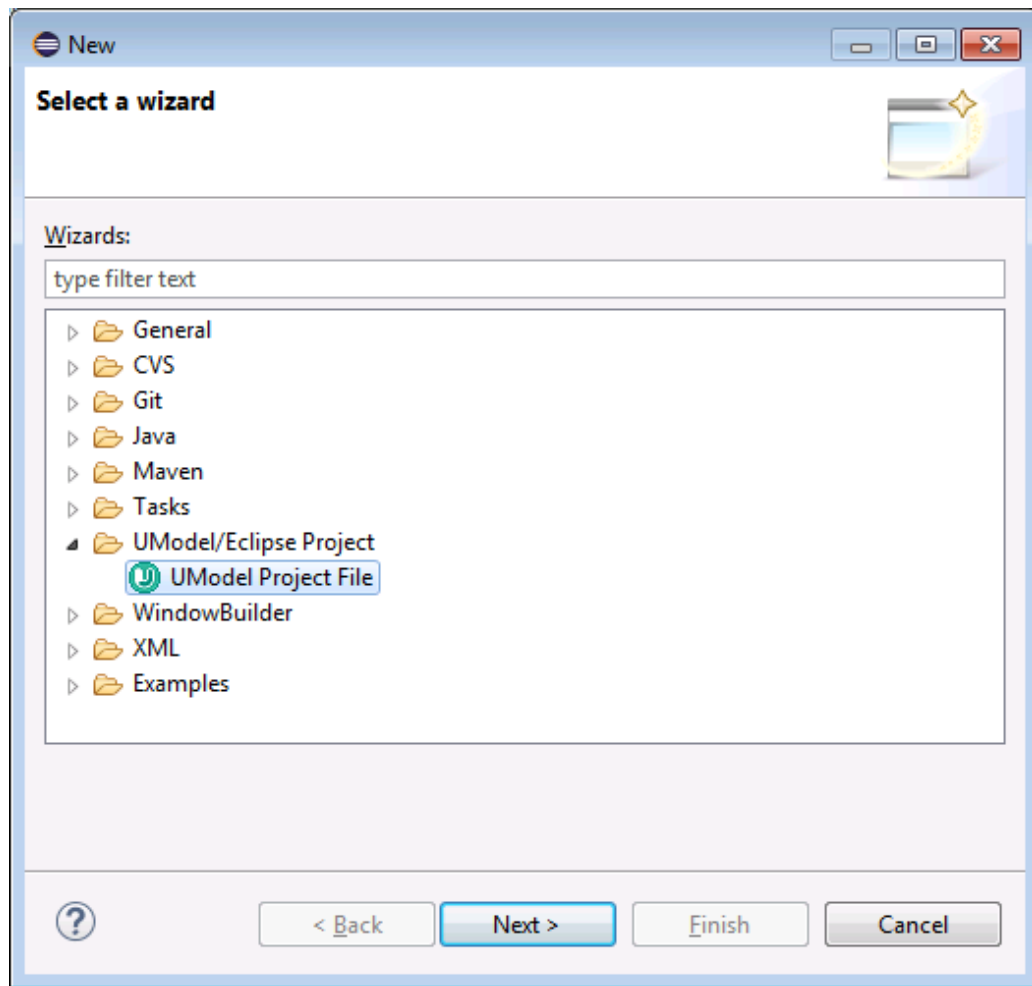
10. Run the application. The Console view displays the text "Hello, World", as shown below.



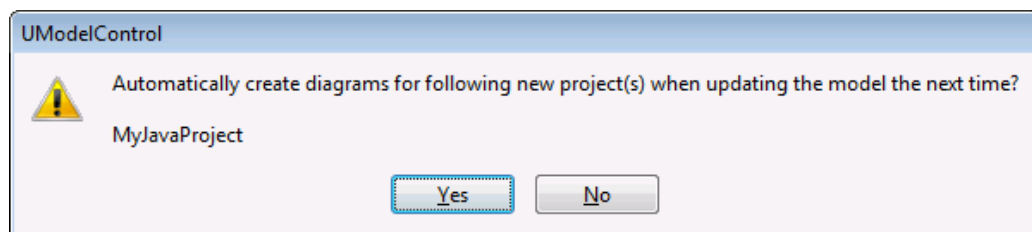
Step 2: Add the UModel project to the Java project

It is now time to add the UModel project file to the Eclipse project. This will create a synchronization relationship between the model and the code.

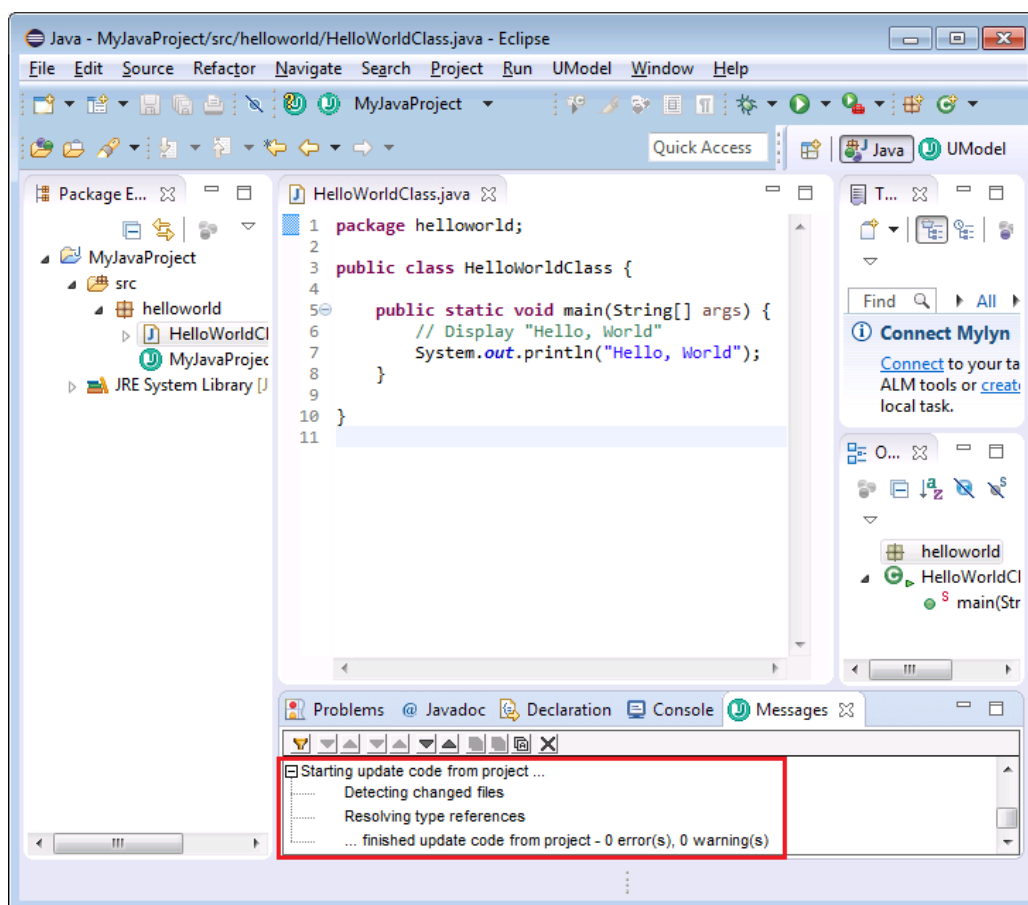
1. On the **File** menu, click **New | Other**, and select **UModel Project File**.



2. Click **Next**. When prompted to specify a location for the new UModel project, leave the default settings as is, and then click **Finish**.
3. When prompted by UModel to create diagrams for the project, click **Yes**.



4. Go through the wizard steps, leaving the default settings as is. When you click **Finish**, the new UModel project is added to the Eclipse project, and synchronization of the code with the model takes place automatically. Notice the messages displayed in the Messages view of UModel.

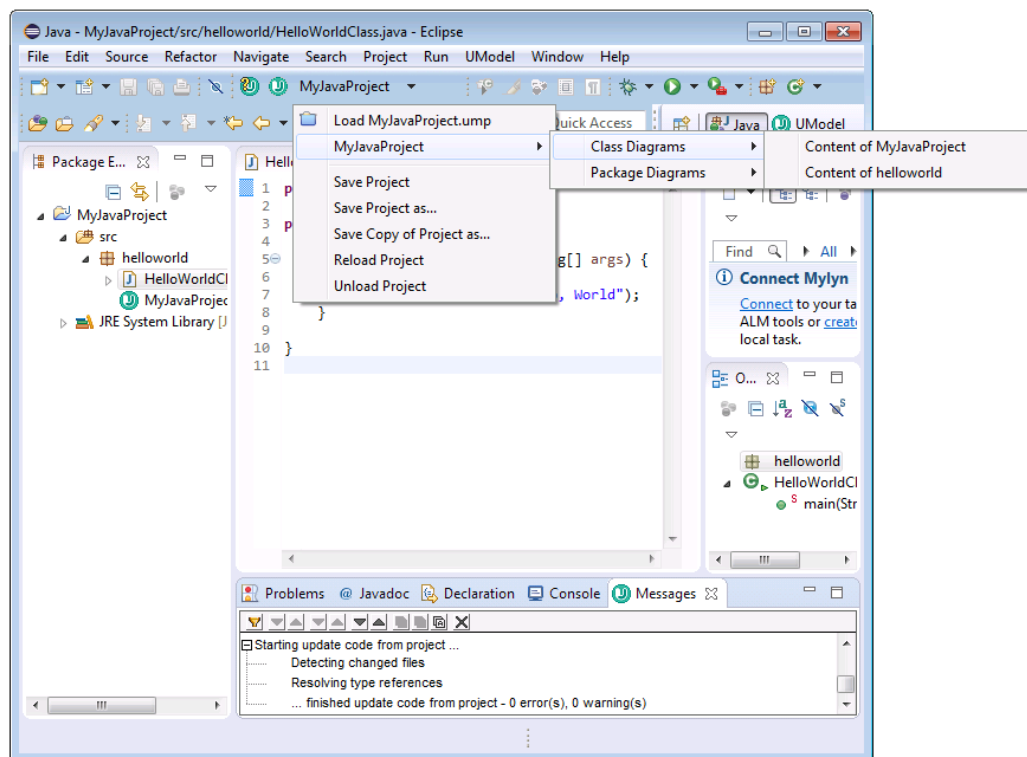


Step 3: Trigger automatic synchronization from model to code

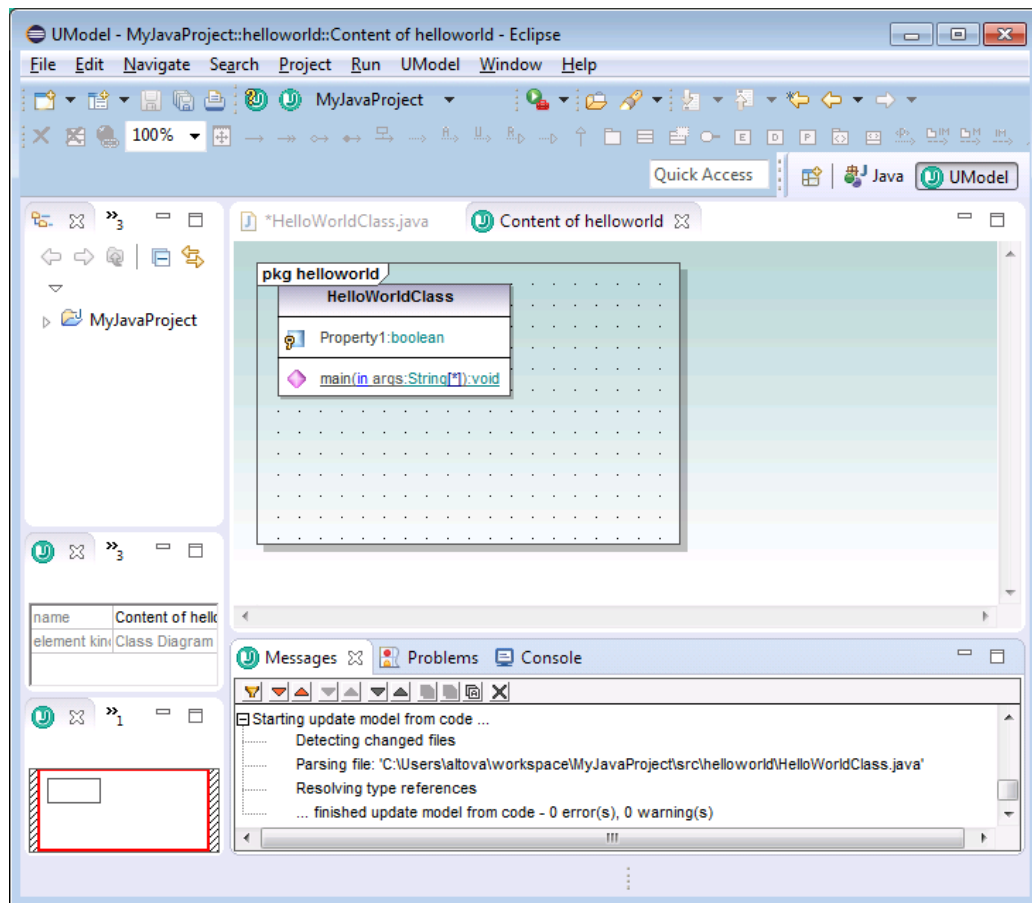
To trigger automatic synchronization from model to code, we will make some changes to the class diagram in the model. Namely, we will add to the class a new property called "Property1" of type "Boolean".

To add the property to the class:

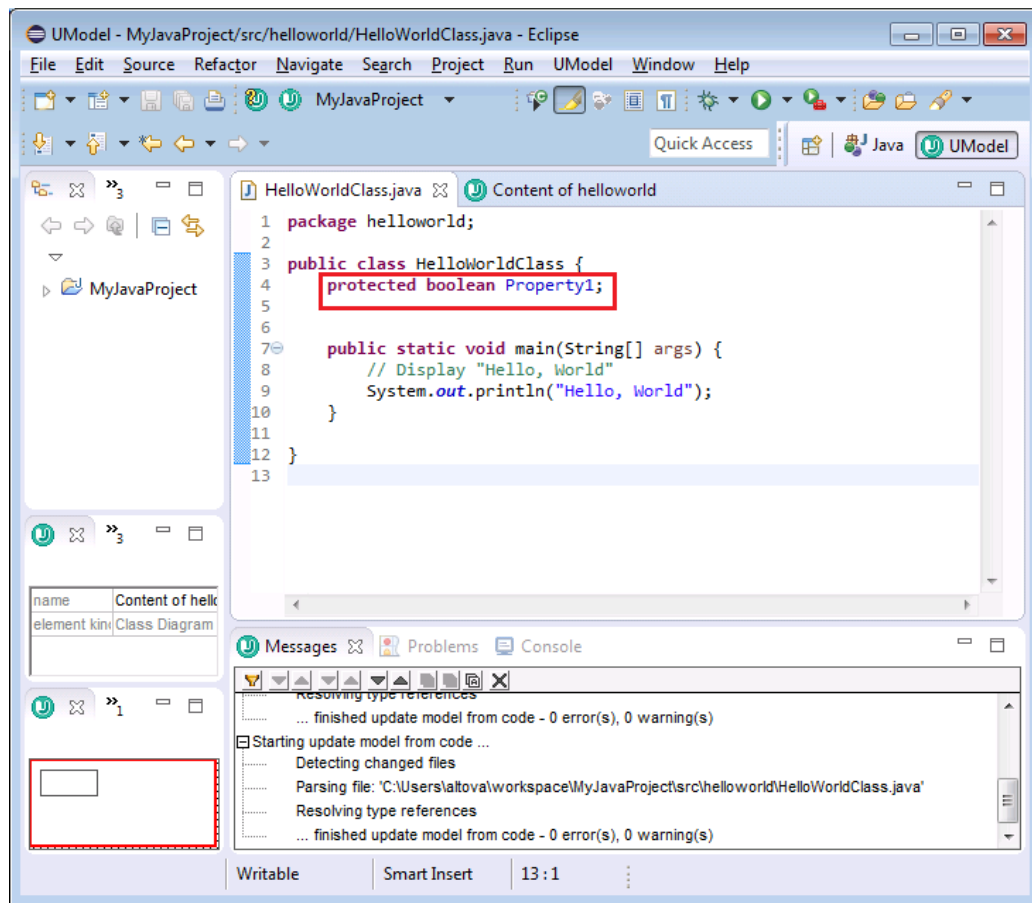
1. In the UModel toolbar, expand the project drop-down list, and open the generated "Content of helloworld" class diagram.



2. Right-click the class, and select **New | Property** from the context menu.
3. Type the property name ("Property1"), followed by the colon character (:), followed by the type ("boolean").

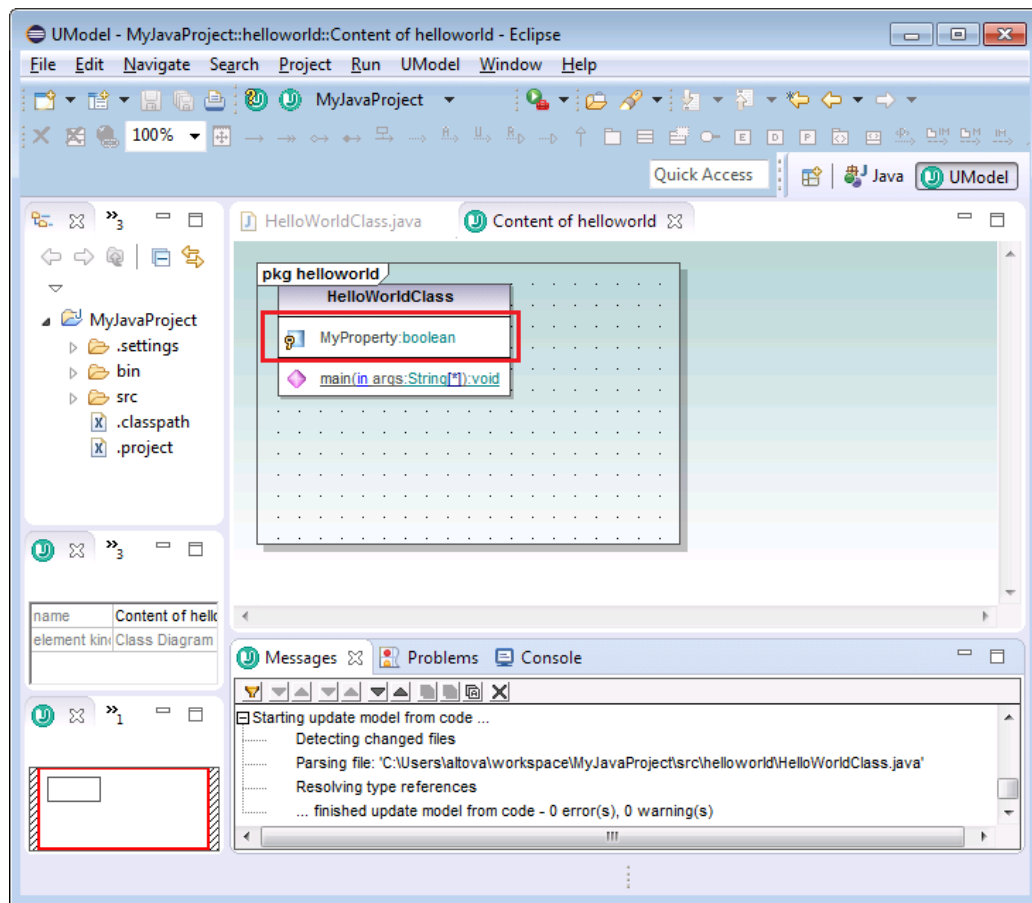


4. Switch back to the code editor. Notice that the newly added property is now reflected in the code.



Step 4. Trigger automatic synchronization from code to model

Let's now trigger automatic synchronization of changes in the opposite direction (from code to model). To do this, change in the code the name of the "Property1" property to "MyProperty", and then save the project. Notice that the changes are now reflected in the diagram.



Chapter 16

Source Control

16 Source Control

The source control support in UModel is available through the Microsoft Source Control Plug-in API (formerly known as the MSSCCI API), versions 1.1, 1.2 and 1.3. This enables you to run source control commands such as "Check in" or "Check out" directly from UModel to virtually any source control system that lets native or third-party clients connect to it through the Microsoft Source Control Plug-in API.

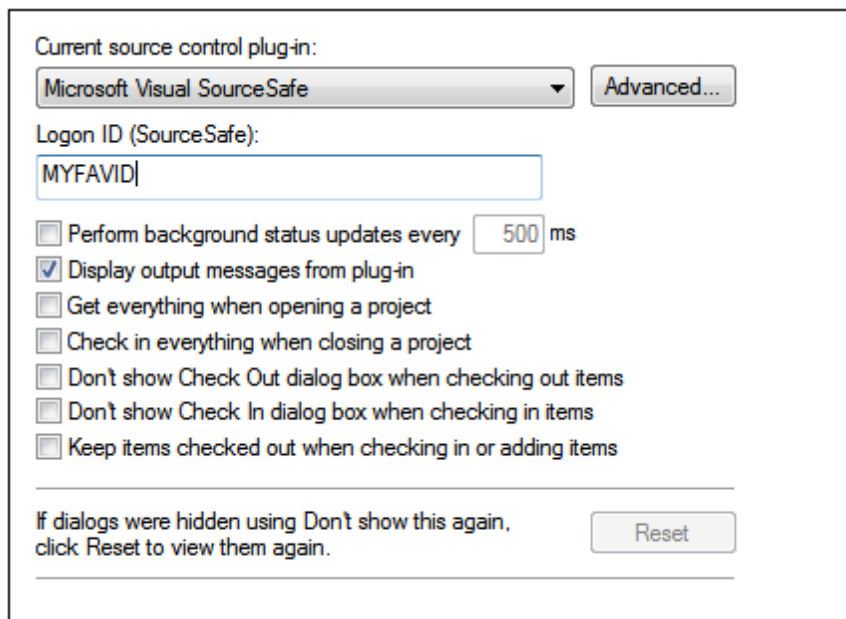
You can use as your source control provider any commercial or non-commercial plug-in that supports the Microsoft Source Control Plug-in API, and can connect to a compatible version control system. For the list of source control systems and plug-ins tested by Altova, see [Supported Source Control Systems](#).

Installing and configuring the source control provider

To view the source control providers available on your system, do the following:

1. On the **Tools** menu, click **Options**.
2. Click the **Source Control** tab.

Any source control plug-ins compatible with the Microsoft Source Code Control Plug-in API are displayed in the **Current source control plug-in** drop-down list.



The screenshot shows the 'Source Control' options dialog. At the top, 'Current source control plug-in:' is set to 'Microsoft Visual SourceSafe' with an 'Advanced...' button. Below, 'Logon ID (SourceSafe):' is 'MYFAVID'. A list of checkboxes includes: 'Perform background status updates every 500 ms' (unchecked), 'Display output messages from plug-in' (checked), 'Get everything when opening a project' (unchecked), 'Check in everything when closing a project' (unchecked), 'Don't show Check Out dialog box when checking out items' (unchecked), 'Don't show Check In dialog box when checking in items' (unchecked), and 'Keep items checked out when checking in or adding items' (unchecked). At the bottom, a 'Reset' button is next to the text 'If dialogs were hidden using Don't show this again, click Reset to view them again.'

If a compatible plug-in cannot be found on your system, the following message is displayed:

"Registration of installed source control providers could not be found or is incomplete."

Some source control systems might not install the source control plug-in automatically, in which case you will need to install it separately. For further instructions, refer to the documentation of the respective source control system. A plug-in (provider) compatible with the Microsoft Source Code Control Plug-in API is expected to be registered under the following registry entry on your

operating system:

```
HKEY_LOCAL_MACHINE\SOFTWARE\SourceCodeControlProvider\InstalledSCCProviders
```

Upon correct installation, the plug-in becomes available automatically in the list of plug-ins available to UModel.

Accessing the source control commands

The commands related to source control are available in the **Project | Source Control** menu.

Resource / Speed issues

Very large source control databases might be introducing a speed/resource penalty when automatically performing background status updates.

You might be able to speed up your system by disabling (or increasing the interval of) the **Perform background status updates every ... seconds** option in the **Source Control** tab accessed through **Tools | Options**.

Note: The **64-bit** version of your Altova application automatically supports any of the supported 32-bit source control programs listed in this documentation. When using a 64-bit Altova application with a 32-bit source control program, the **Perform background status updates every ... seconds** option is automatically grayed-out and cannot be selected.

Differencing with Altova DiffDog

You can configure many source control systems (including Git and TortoiseSVN) so that they use Altova DiffDog as their differencing tool. For more information about DiffDog, see <http://www.altova.com/diffdog.html>. For DiffDog documentation, see <http://www.altova.com/documentation.html>.

16.1 Setting Up Source Control

The mechanism for setting up source control and placing files in a UModel project under source control is as follows:

1. If this hasn't been done already, install the source control system (see [Supported Source Control Systems](#)) and set up the source control database (repository) to which you wish to save your work.
2. Create a local workspace folder that will contain the working files that you wish to place under source control. The folder that contains all your workspace folders and files is called the local folder, and the path to the local folder is referred to as the local path. This local folder will be bound to a particular folder in the repository.
3. In your Altova application, create an application project folder to which you must add the files you wish to place under source control. This organization of files in an application project is abstract. The files in a project reference physical files saved locally, preferably in one folder (with sub-folders if required) for each project.
4. In the source control system's database (also referred to as source control or repository), a folder is created that is bound to the local folder. This folder (called the bound folder) will replicate the structure of the local folder so that all files to be placed under source control are correctly located hierarchically within the bound folder. The bound folder is usually created when you add a file or an application project to source control for the first time.

16.2 Supported Source Control Systems

The list below shows the Source Control Servers (SCSs) supported by UModel, together with their respective Source Control Clients (SCCs). The list is organized alphabetically by SCS. Note the following:

- Altova has implemented the Microsoft Source Control Plug-in API (versions 1.1, 1.2, and 1.3) in UModel, and has tested support for the listed drivers and revision control systems. It is expected that UModel will continue to support these products if, and when, they are updated.
- Source Code Control clients not listed below, but which implement the Microsoft Source Control Plug-in API, should also work with UModel.

Source Control System	Source Code Control Clients
AccuRev 4.7.0 Windows	AccuBridge for Microsoft SCC 2008.2
Bazaar 1.9 Windows	Aigenta Unified SCC 1.0.6
Borland StarTeam 2008	Borland StarTeam Cross-Platform Client 2008 R2
Codice Software Plastic SCM Professional 2.7.127.10 (Server)	Codice Software Plastic SCM Professional 2.7.127.10 (SCC Plugin)
Collabnet Subversion 1.5.4	• Aigenta Unified SCC 1.0.6 • PushOK SVN SCC 1.5.1.1 • PushOK SVN SCC x64 version 1.6.3.1 • TamTam SVN SCC 1.2.24
ComponentSoftware CS-RCS (PRO) 5.1	ComponentSoftware CS-RCS (PRO) 5.1
Dynamsoft SourceAnywhere for VSS 5.3.2 Standard/Professional Server	Dynamsoft SourceAnywhere for VSS 5.3.2 Client
Dynamsoft SourceAnywhere Hosted	Dynamsoft SourceAnywhere Hosted Client (22252)
Dynamsoft SourceAnywhere Standalone 2.2 Server	Dynamsoft SourceAnywhere Standalone 2.2 Client
Git	PushOK GIT SCC plug-in (see Source Control with Git)
IBM Rational ClearCase 7.0.1 (LT)	IBM Rational ClearCase 7.0.1 (LT)
March-Hare CVSNT 2.5 (2.5.03.2382)	Aigenta Unified SCC 1.0.6
March-Hare CVS Suite 2008	• Jalindi Igloo 1.0.3 • March-Hare CVS Suite Client 2008 (3321) • PushOK CVS SCC NT 2.1.2.5 • PushOK CVS SCC x64 version 2.2.0.4 • TamTam CVS SCC 1.2.40
Mercurial 1.0.2 for Windows	Sergey Antonov HgSCC 1.0.1
Microsoft SourceSafe 2005 with CTP	Microsoft SourceSafe 2005 with CTP

Source Control System	Source Code Control Clients
Microsoft Visual Studio Team System 2008/2010 Team Foundation Server	Microsoft Team Foundation Server 2008/2010 MSSCCI Provider
Perforce 2008 P4S 2008.1	Perforce P4V 2008.1
PureCM Server 2008/3a	PureCM Client 2008/3a
QSC Team Coherence Server 7.2.1.35	QSC Team Coherence Client 7.2.1.35
Reliable Software Code Co-Op 5.1a	Reliable Software Code Co-Op 5.1a
Seapine Surround SCM Client/Server for Windows 2009.0.0	Seapine Surround SCM Client 2009.0.0
Serena Dimensions Express/CM 10.1.3 for Win32 Server	Serena Dimensions 10.1.3 for Win32 Client
Softimage Alienbrain Server 8.1.0.7300	Softimage Alienbrain Essentials/Advanced Client 8.1.0.7300
SourceGear Fortress 1.1.4 Server	SourceGear Fortress 1.1.4 Client
SourceGear SourceOffsite Server 4.2.0	SourceGear SourceOffsite Client 4.2.0 (Windows)
SourceGear Vault 4.1.4 Server	SourceGear Vault 4.1.4 Client
VisualSVN Server 1.6	• Aigenta Unified SCC 1.0.6 • PushOK SVN SCC 1.5.1.1 • PushOK SVN SCC x64 version 1.6.3.1 • TamTam SVN SCC 1.2.24

16.3 Source Control Commands

The following sections use Visual SourceSafe to show the source control features of UModel. The examples in this section use the **Bank_CSharp.ump** UModel project (and associated code files) available in the C:\Documents and Settings\<username>\My Documents\Altova\UModel2016\UModelExamples\ folder. Note that a Source Control project is not the same as a UModel project. Source Control projects are directory dependent, whereas UModel projects are logical constructions without direct directory dependence.

To access the Source Control commands, do one of the following:

- Use the menu command **Project | Source Control**
- Use the **context** menu in the Model Tree
- Click the source control toolbar buttons in the Source Control toolbar. Use **Tools | Customize | Toolbars** to activate the toolbar.

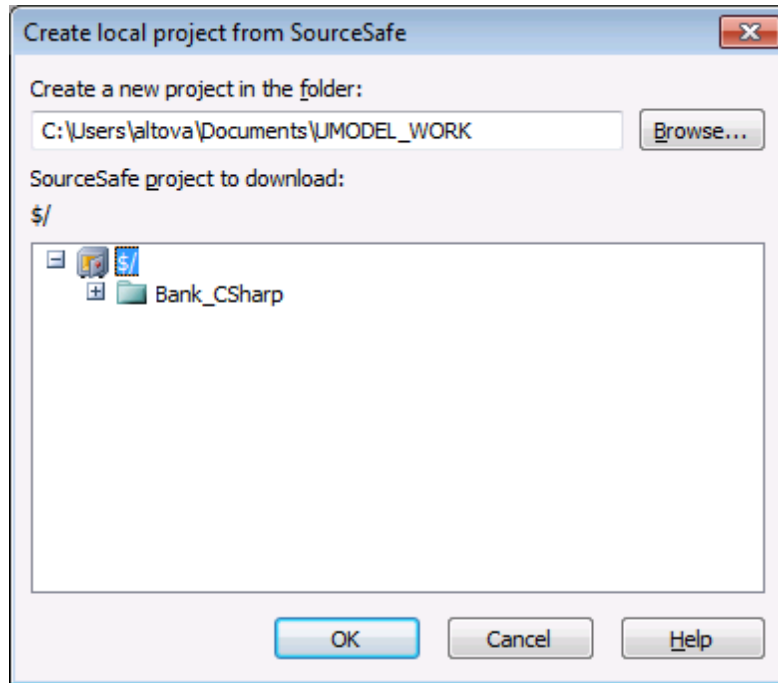
The description of the version control commands that follow apply to the standalone version of UModel. The Visual Studio and Eclipse versions of UModel use the version control functionality and menu items available in those IDEs.

[Open from Source Control](#)
[Enable Source Control](#)
[Get Latest Version](#)
[Get](#)
[Get Folder\(s\)](#)
[Check Out](#)
[Check In](#)
[Undo Check Out...](#)
[Add to Source Control](#)
[Remove from Source Control](#)
[Share from Source Control](#)
[Show History](#)
[Show Differences](#)
[Show Properties](#)
[Refresh Status](#)
[Source Control Manager](#)
[Change Source Control](#)

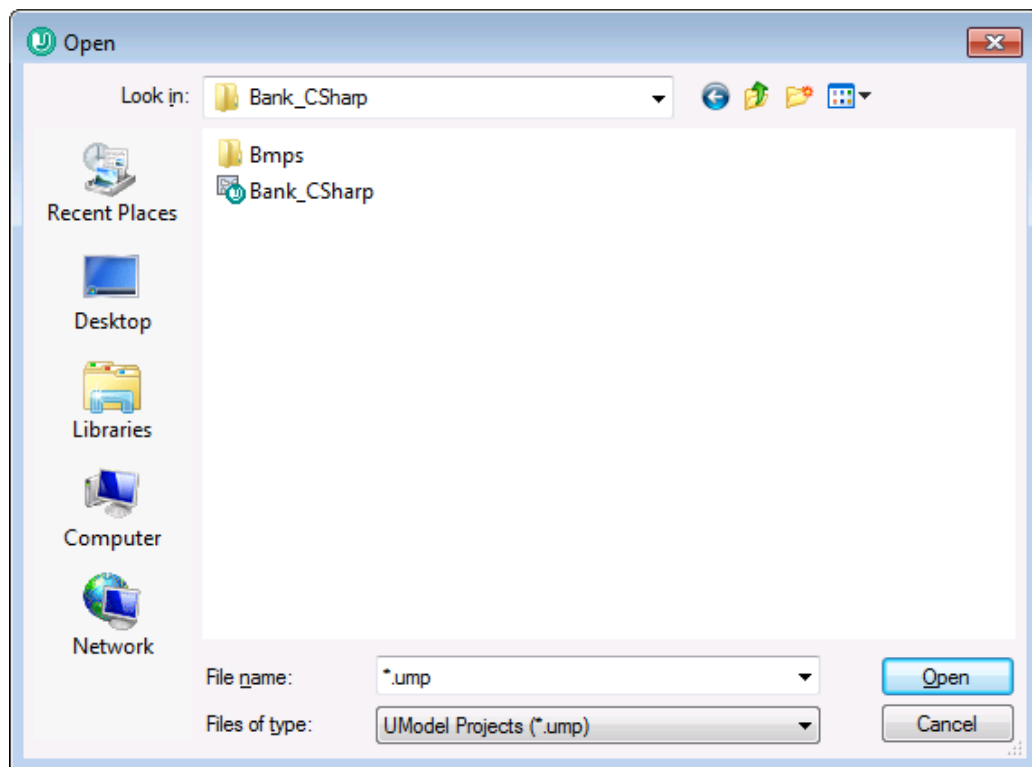
16.3.1 Open from Source Control

The Open from Source Control command creates a local project from an existing source control database, and places it under source control, SourceSafe in this case.

1. Select **Project | Source Control | Open from Source Control**.
The Login dialog box is opened, enter your login details to continue.
The "Create local project from SourceSafe" dialog box appears.
2. Define the directory to contain the new local project e.g. c:\temp\ssc. This becomes the **Working directory**, or the Check Out Folder.

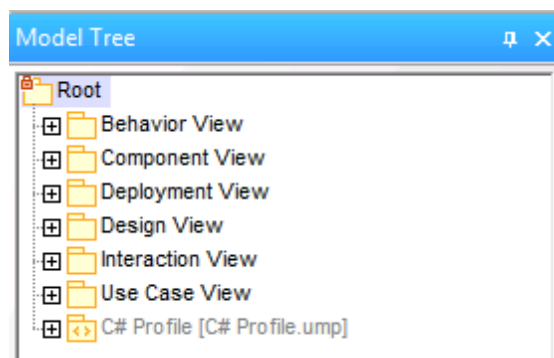


3. Select the SourceSafe project you want to download e.g. Bank_CSharp.
If the folder you define here does not exist at the location, a dialog box opens prompting you to create it.
4. Click **Yes** to create the new directory.
The Open dialog box is now visible.



5. Select the **Bank_CSharp.ump** UModel project file and click Open.

Bank_CSharp.ump now opens in UModel, and the file is placed under source control. This is indicated by the lock symbol visible on the Root folder in the Model Tree window. The Root folder represents both the project file and the working directory for source control operations.

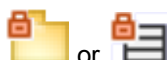


The BankCSharp directory has been created locally, you can now work with these files as you normally would.

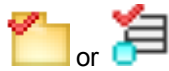
Note:

To place under source control the code files generated when synchronizing code, see: [Add to Source Control](#)

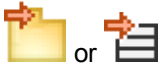
Source control symbols:



The lock symbol denotes that the file, or folder is under source control, but is currently not checked out.



The red check mark denotes checked out, i.e. the UModel project file (or code file) has been checked out for editing. The asterisk in the Application title bar denotes that changes have been made to the file, and you will be prompted to save it when you exit.



The arrow symbol shows that the file(s) have been checked out by someone else in the network, or by you into a different working directory

16.3.2 Enable Source Control

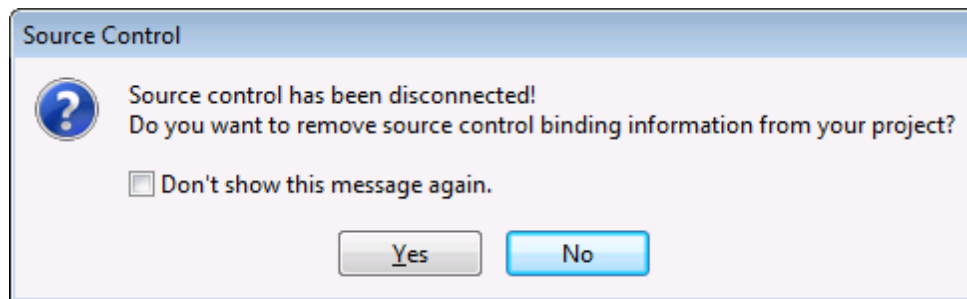
This command allows you to enable or disable source control for a UModel project and is available through the Project menu item, i.e. **Project | Source Control | Enable Source Control**. Selecting this option on any file or folder, enables/disables source control for the whole UModel project.

To enable Source Control for a project:

1. Select the menu option **Project | Source Control** and activate/check the **Enable source control** check box of the fly-out menu. The previous check in/out status of the various files are retrieved and displayed in the Model Tree window.

To disable Source Control for a project:

1. Select the menu option **Project | Source Control** and uncheck the **Enable source control** check box.



You are now prompted if you want to remove the binding information from the project.

To **provisionally** disable source control for the project, select **No**.

To **permanently** disable source control for the project, select **Yes**.

16.3.3 Get Latest Version

Retrieves and places the latest source control version of the selected file(s) in the working directory. The files are retrieved as read-only and are not checked out.

If the affected files are currently checked out, different things occur depending on the specific version control plugin: nothing happens, new data are merged into your local file, or your changes are overwritten.

This command works in a similar fashion to the Get command, but does not display the "Source control - Get" dialog box. It is therefore not possible to specify Advanced get options.

Note that this command automatically performs a recursive get latest version operation when performed on a folder, i.e. it affects all other files below the current one in the package hierarchy.

To get the latest version of a file:

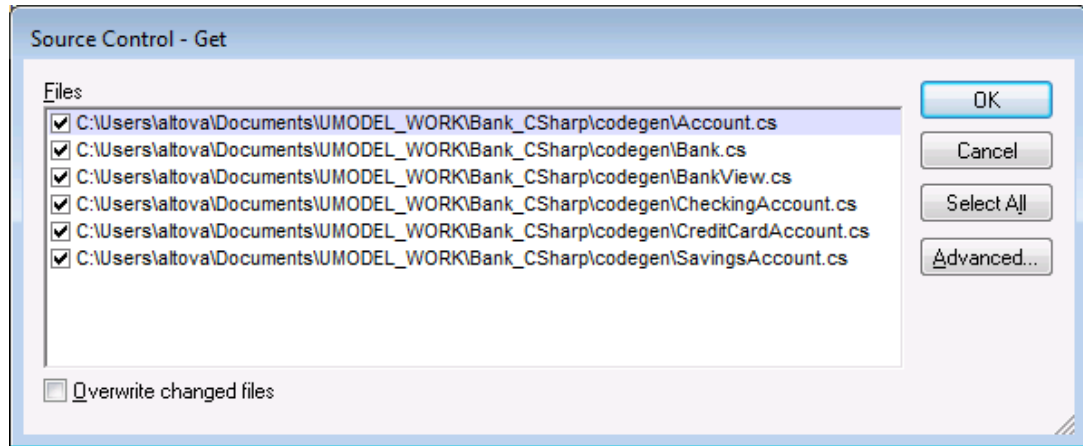
1. Select the file(s) you want to get the latest version of in the Model Tree.
2. Select **Project | Source Control | Get Latest Version**.

16.3.4 Get

Retrieves a read-only copy of the selected files and places them in the working folder. The files are not checked-out for editing per default.

Using Get:

- Select the files you want to get in the Model Tree.
- Select **Project | Source Control | Get**.



Overwrite changed files

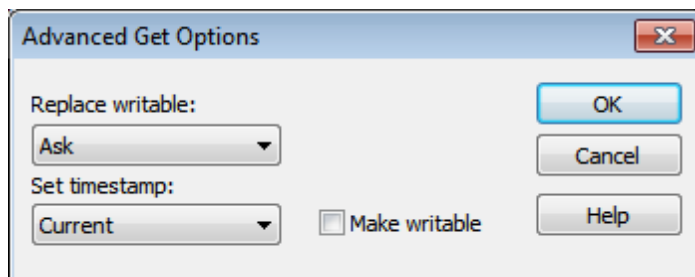
Overwrites those files that have been changed locally with those from the source control database.

Select All

Selects all the files in the list box.

Advanced

Allows you to define the **Replace writable** and **Set timestamp** options in the respective combo boxes.



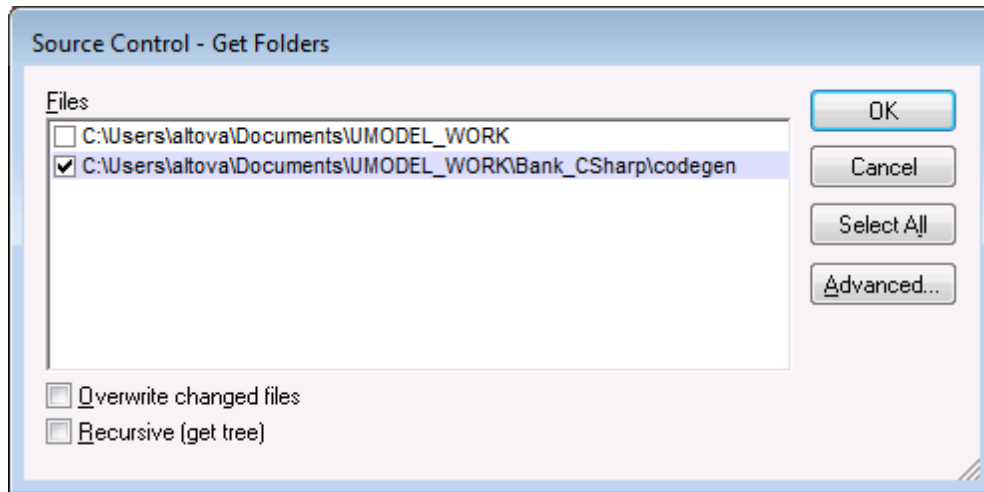
The "Make writable" check box removes the read-only attribute of the retrieved files.

16.3.5 Get Folder(s)

Retrieves read-only copies of files in the selected folders and places them in the working folder. The files are not checked-out for editing per default.

Using Get Folders:

- Select the folder you want to get in the Model Tree.
- Select **Project | Source Control | Get Folders**.



Overwrite changed files

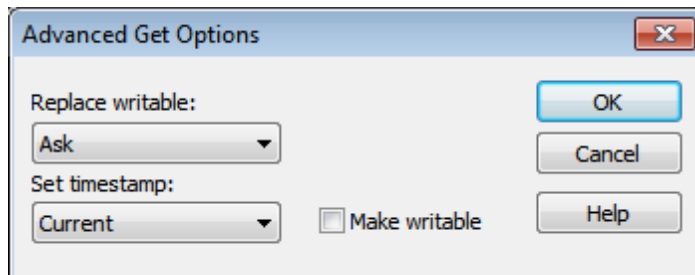
Overwrites those files that have been changed locally with those from the source control database.

Recursive (get tree)

Retrieves all files of the folder tree below the selected folder.

Advanced

Allows you to define the **Replace writable** and **Set timestamp** options in the respective combo boxes.



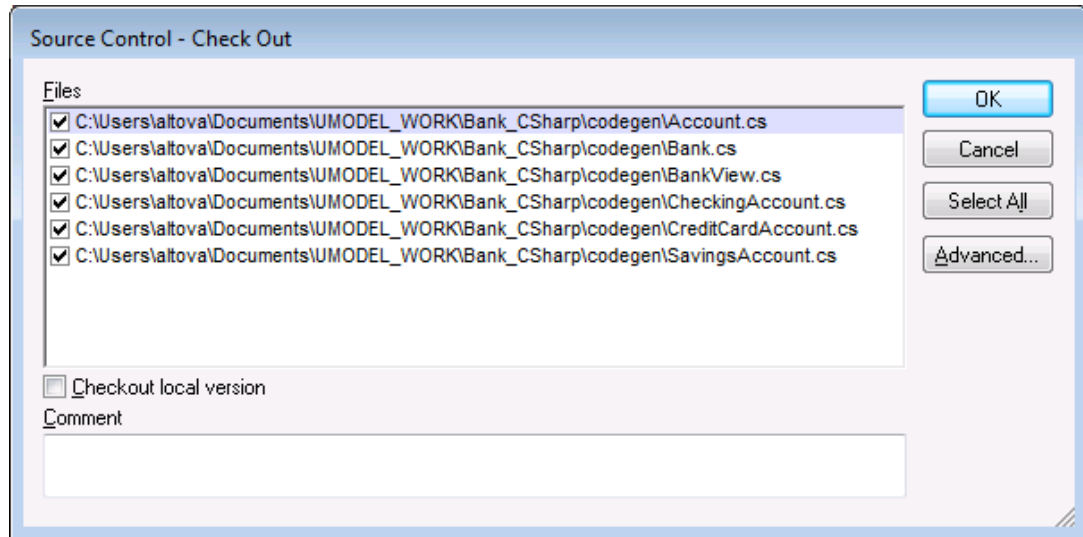
The "Make writable" check box removes the read-only attribute of the retrieved files.

16.3.6 Check Out

This command **checks out** the latest version of the selected files and places writable copies in working directory. The files are flagged as "checked out" for all other users.

To Check Out files:

- Select the file or folder you want to check out in the Model Tree.
- Select **Project | Source Control | Check Out**.



Note:

You can change the number of files to check out, by activating the individual check boxes in the Files list box.

Checkout local version:

Checks out only the local versions of the files, not those from the source control database.

The following items can be checked out:

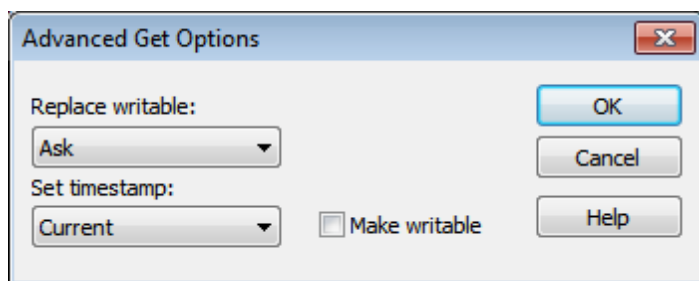
- Single files, click on the respective files (CTRL + click, in the Model Tree)
- Folders, click on the folders (CTRL + click, in the Model Tree)



The red check mark denotes that the file/folder has been checked out.

Advanced

Allows you to define the **Replace writable** and **Set timestamp** options in the respective combo boxes.



The "Make writable" check box removes the read-only attribute of the retrieved files.

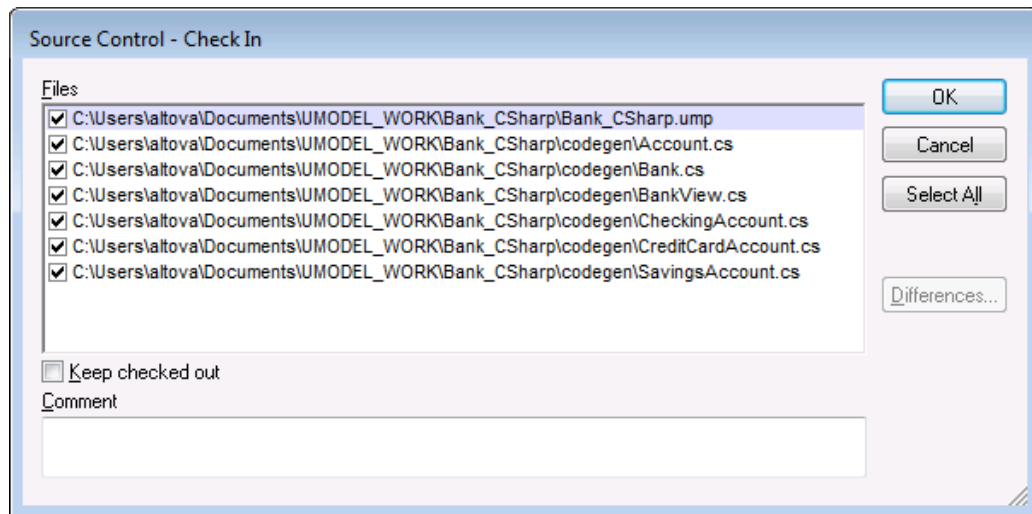
16.3.7 Check In

This command **checks in** the previously checked out files, i.e. your locally updated files, and places them in the source control database.

To Check In files:

- Select the files in the Model Tree
- Select **Project | Source Control | Check In**.

Shortcut: Right-click a checked out item in the project window, and select "Check in" from the Context menu.

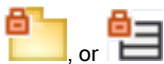


Note:

You can change the number of files to check in, by activating the individual check boxes in the Files list box.

The following items can be checked in:

- Single files, click on the respective files (CTRL + click, in Model Tree)
- Folders, click on the folders (CTRL + click, in Model Tree)



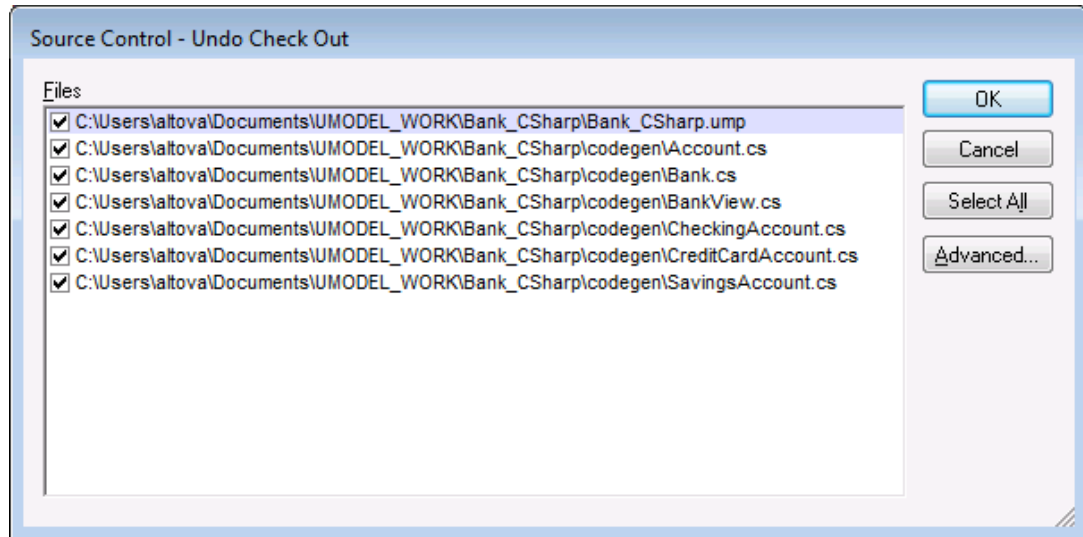
The lock symbol denotes that the file/folder is **under source control**, but is currently not checked out.

16.3.8 Undo Check Out...

This command **discards changes** made to previously checked out files, i.e. your locally updated files, and retains the old files from the source control database.

To Undo Check Out..

- Select the files in the Model Tree
- Select **Project | Source Control | Undo Check Out**.



Note:

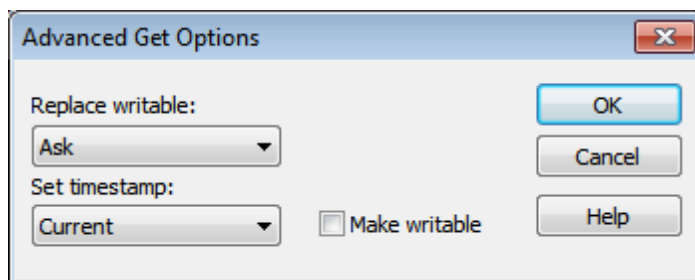
You can change the number of files by activating the individual check boxes in the Files list box.

The Undo check out option can apply to the following items:

- Single files, click on the respective files (CTRL + click, in Model Tree)
- Folders, click on the folders (CTRL + click, in Model Tree)

Advanced

Allows you to define the **Replace writable** and **Set timestamp** options in the respective combo boxes.



The "Make writable" check box removes the read-only attribute of the retrieved files.

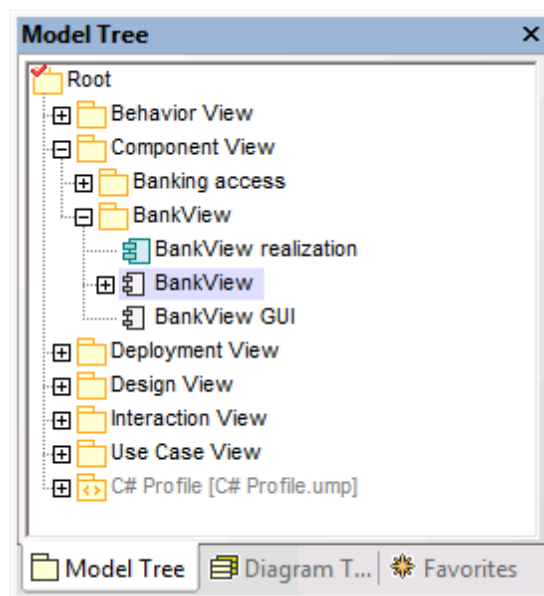
16.3.9 Add to Source Control

Adds the selected files or folders to the source control database and places them under source control. If you are adding a new UModel project you will be prompted for the workspace folder and the location at which your project should be stored.

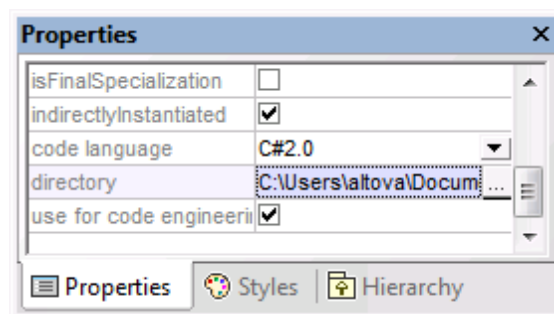
Having placed the UModel project file (*.ump) under source control, you can then add the code files produced by the code-engineering process, to source control as well. For this to work, the generated code files and the UModel project have to be placed **in**, or **under**, the same SourceSafe **working** directory. The working directory used in this section is **C:\Users\Altova\Documents\UMODEL_WORK**.

To add UModel generated code files to source control:

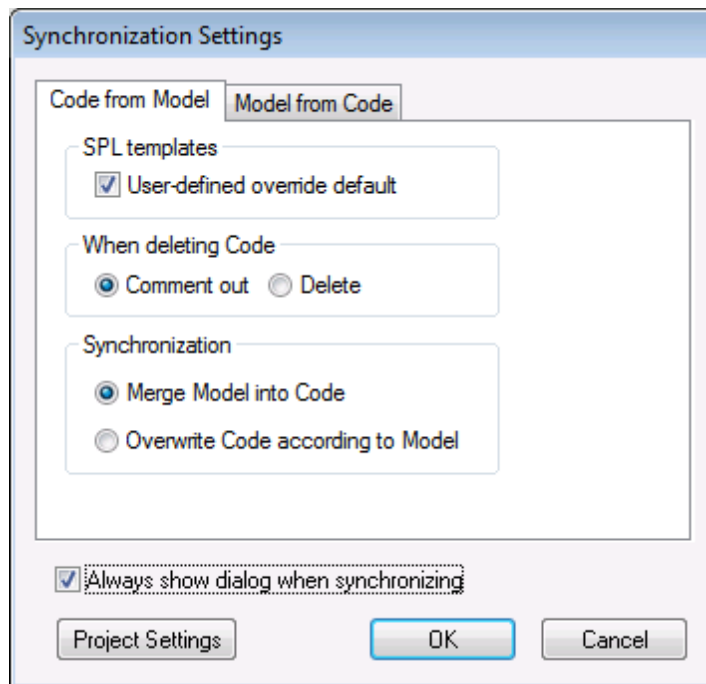
1. Expand the Component View folder in the Model Tree and Navigate to the BankView component.



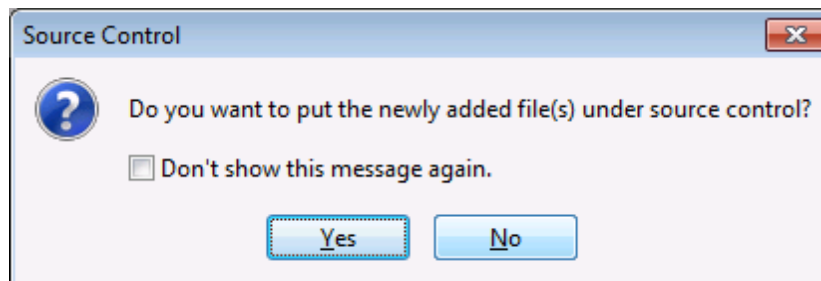
2. Click the BankView component and click the Browse icon next to the "**directory**" field in the Properties window.



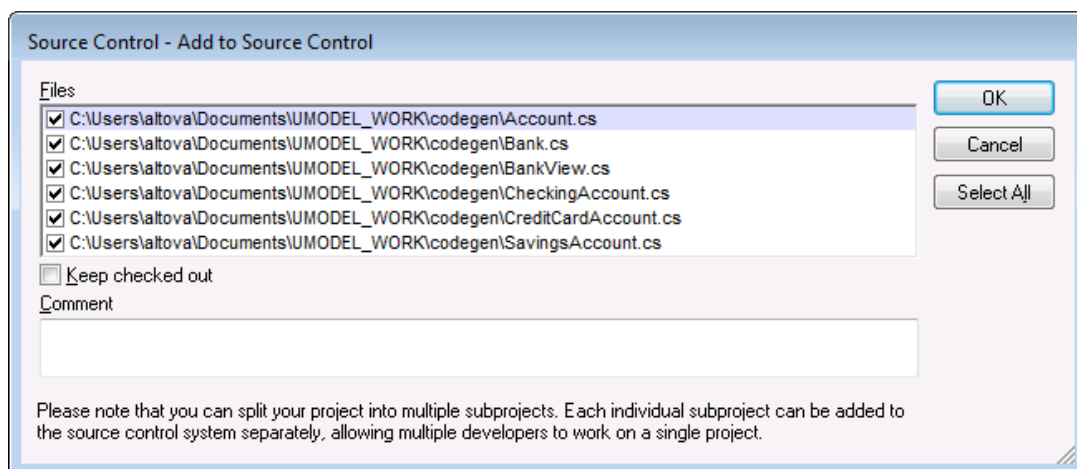
3. Change the code engineering directory to **C:\Users\Altova\Documents\UMODEL_WORK\codegen**.
4. Select the menu item **Project | Merge Program Code from UModel project**.
5. Change the Synchronization settings if necessary, and click OK to confirm.



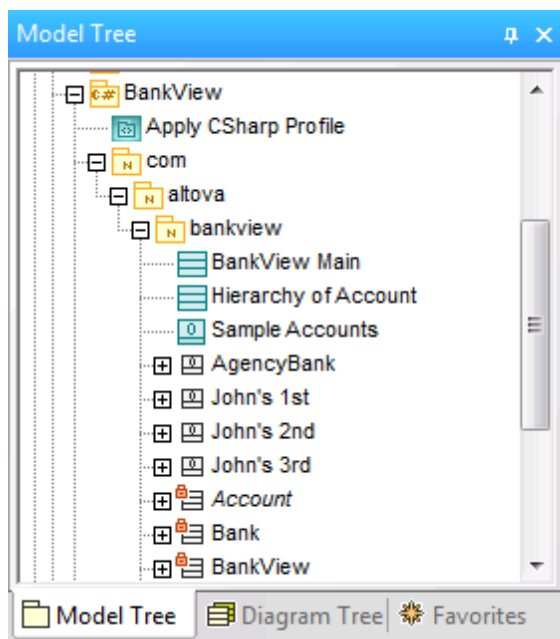
The Messages window displays the code from project process.
A message box opens asking if you want to place the newly created files under source control.



6. Click Yes to do so.
7. The "Add to Source Control" dialog box is opened, allowing you to select the files you want to place under source control.



8. Click OK once you have selected the files you want to place under source control. The lock symbol now appears next to each of the classes/file sources placed under source control.

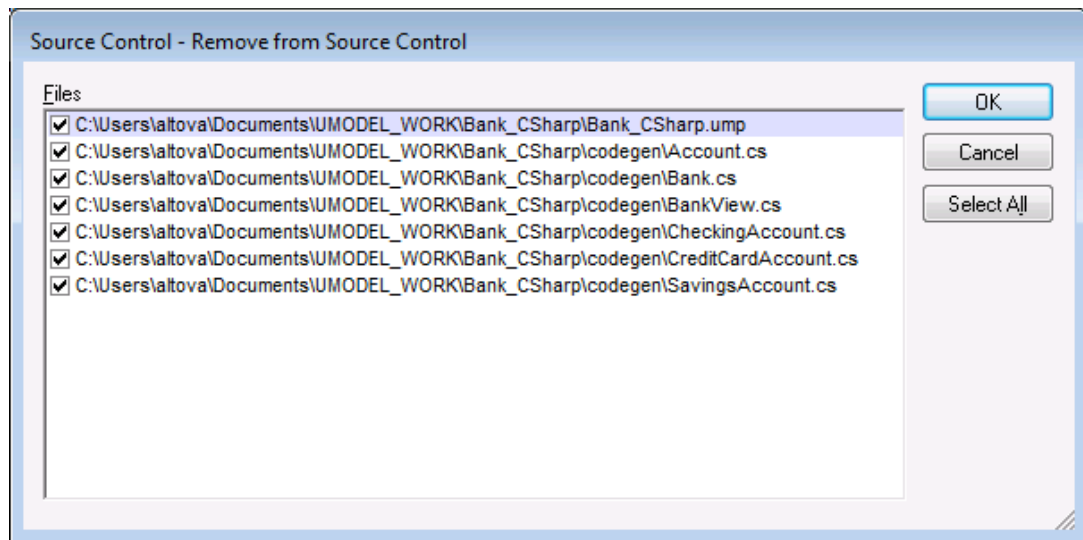


16.3.10 Remove from Source Control

This command **removes** previously added files, from the source control database. These type of files remain visible in the Model Tree but cannot be checked in or out. Use the "Add to Source Control" command to place them back under source control.

To remove files from the source control provider:

- Select the files you want to remove in the Model Tree.
- Select **Project | Source Control | Remove from Source Control**.



Note:

You can change the number of files to remove, by activating the individual check boxes in the Files list box.

The following items can be removed from source control:

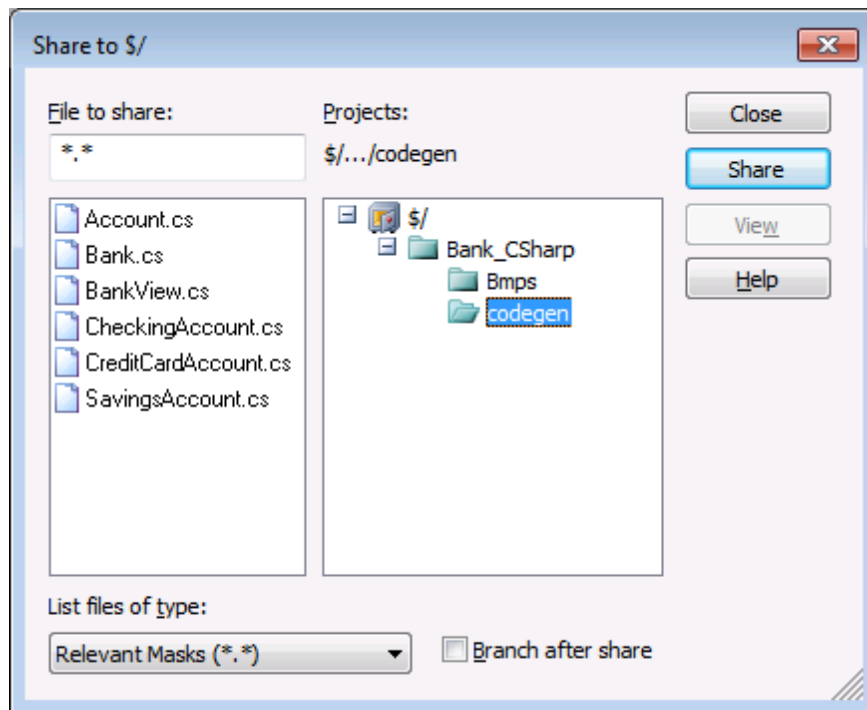
- Single files, click on the respective files (CTRL + click, for several)
- Folders, click on the folder icon.

16.3.11 Share from Source Control

This command shares/branches files from other projects/folders within the source control repository, into the selected folder. To use the Share command you must have the Check in/out rights to the project you are sharing from.

To share a file from source control:

1. Select the folder you want to share files to, in the Model Tree window, and select **Project | Source Control | Share from Source Control**. e.g. BankView Component in the Component View folder.
2. Select the project folder that contains the file you want to share in the "Projects" list box.



3. Select the file you want to share in the "Files to share" list box and click the Share button.
The file is now removed from the "File to share" list.
4. Click the Close button to continue.

Branch after share

Shares the file and creates a new branch to create a separate version.

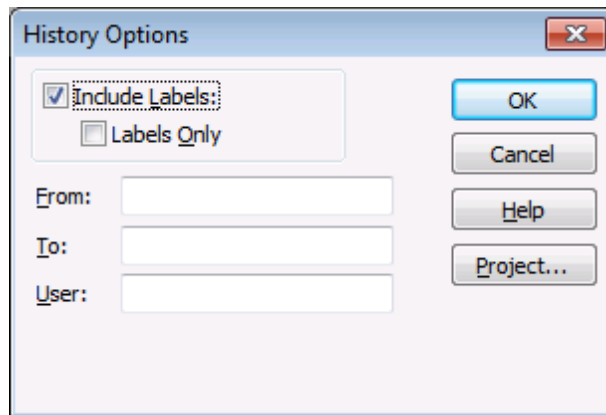
16.3.12 Show History

This command **displays** the history of a file under source control, and allows you to view, see detailed history info, difference, or retrieve previous versions of a file.

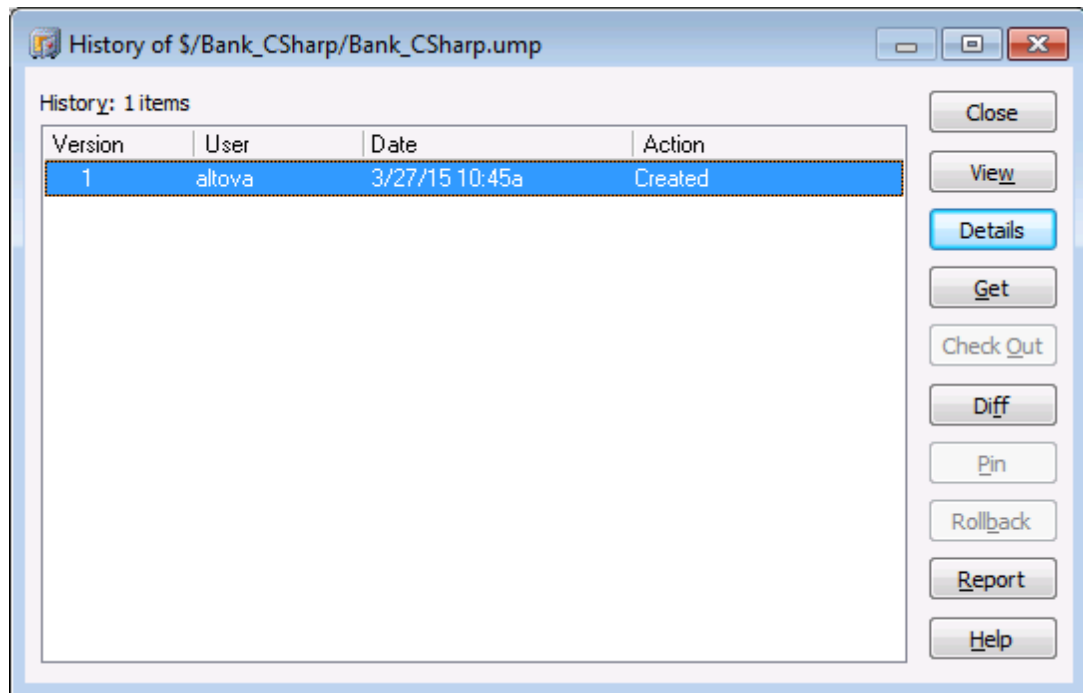
To show the history of a file:

1. Click on the file in the Model Tree window.
2. Select the menu options **Project | Source control | Show history**.

A dialog box prompting for more information opens.



3. Select the appropriate entries and confirm with **OK**.



This dialog box provides various ways of comparing and getting specific versions of the file in question. Double clicking an entry in the list opens the History Details dialog box for that file.

Close

Closes this dialog box.

View

Opens a further dialog box in which you can select the type of viewer you want to see the file with.

Details

Opens a dialog box in which you can see the [properties](#) of the currently active file.

Get

Allows you to retrieve one of the previous versions of the file in the version list, and place it into the working directory.

Check Out

Allows you to check out the **latest** version of the file.

Diff

Opens the [Difference options](#) dialog box, which allows you to define the difference options when viewing the differences between two file versions.

Use CTRL+Click to mark two file versions in this window, then click Diff to view the differences between them.

Pin

Pins or unpins a version of the file, allowing you to define the specific file version to use when differencing two files.

Rollback

Rolls back to the selected version of the file.

Report

Generates a history report which you can send to the printer, file, or clipboard.

Help

Opens the online help of the source control provider plugin.

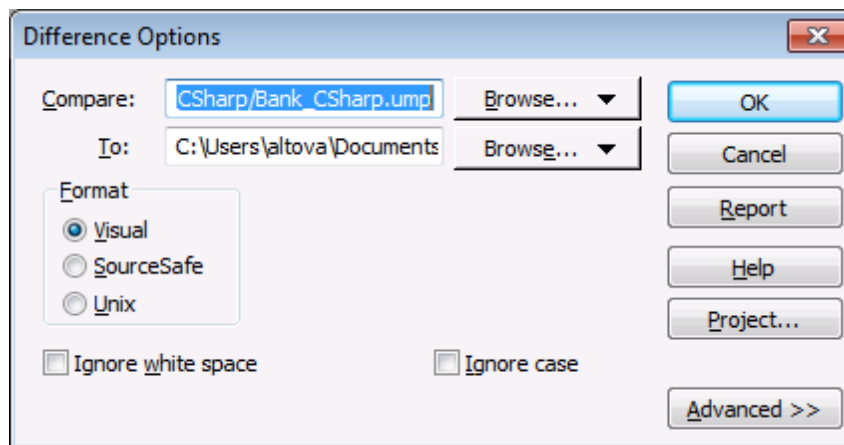
16.3.13 Show Differences

This command **displays** the differences between the file currently in the source control repository, and the **checked in/out** file of the same name in the working directory.

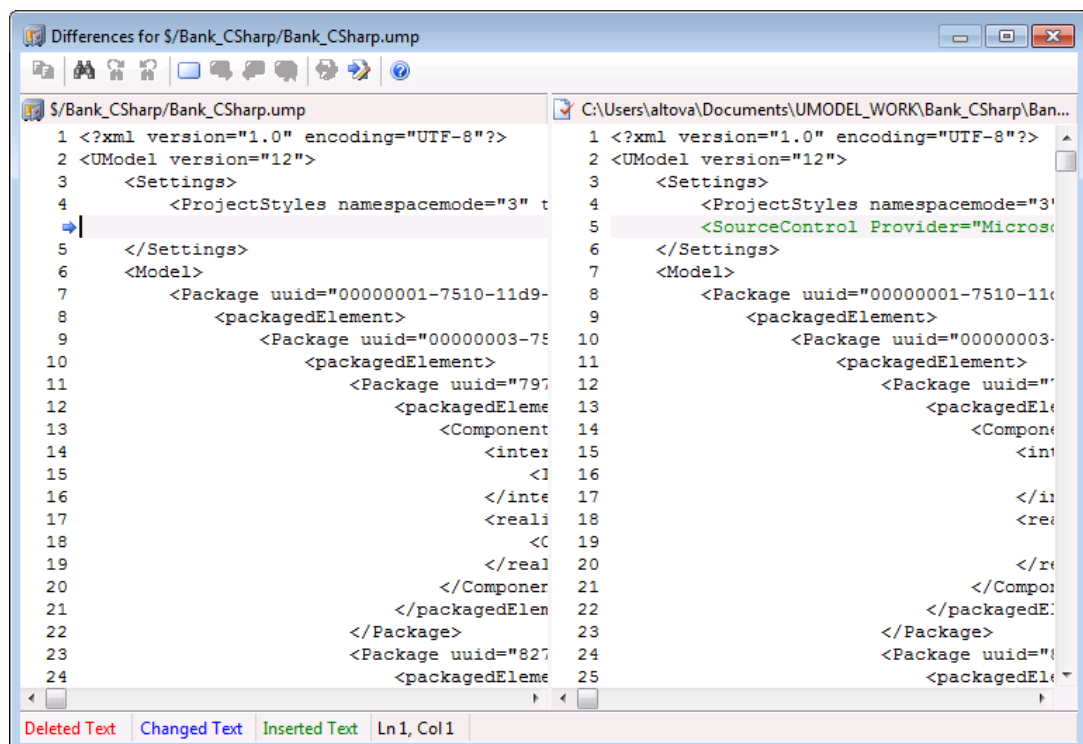
If you have "pinned" one of the files in the **history** dialog box, then the pinned file will be used in the "Compare" text box. Any two files can be selected using the Browse buttons.

To show the differences between two files:

1. Click on a file in the Model Tree window.
2. Select the menu option **Project | Source control | Show Differences**.
A dialog box prompting for more information appears.



3. Select the appropriate entries and confirm with **OK**.



The differences between the two files are highlighted in both windows (this example uses MS SourceSafe).

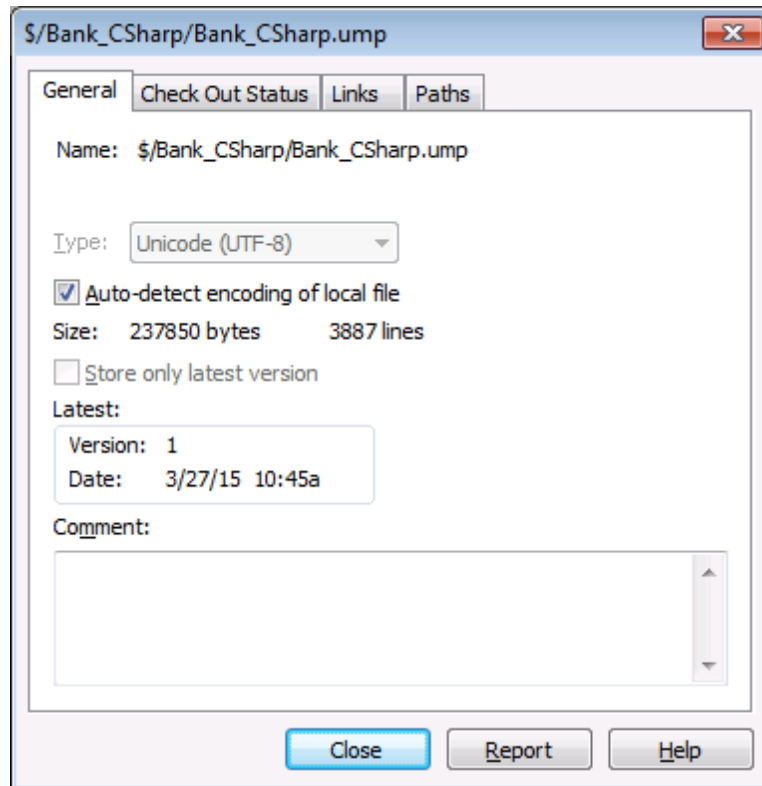
16.3.14 Show Properties

This command **displays** the properties of the currently selected file, and is dependent on the source control provider you use.

To display the properties of the currently selected file:

- Select **Project | Source Control | Properties**.

This command can only be used on single files.



16.3.15 Refresh Status

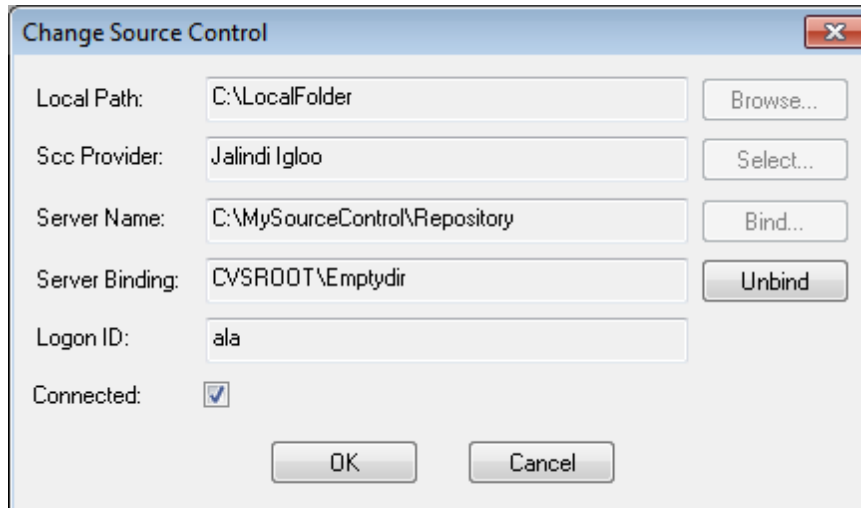
This command **refreshes** the status of all project files, independent of their current status.

16.3.16 Source Control Manager

This command **starts** your source control software with its native user interface.

16.3.17 Change Source Control

This dialog box allows you to change the source control binding that you are using. Click the Unbind button first, then (optionally) click the Select button to select a new source control provider, and finally click the Bind button to bind to a new location in the repository.



The dialog box titled "Change Source Control" contains the following fields and buttons:

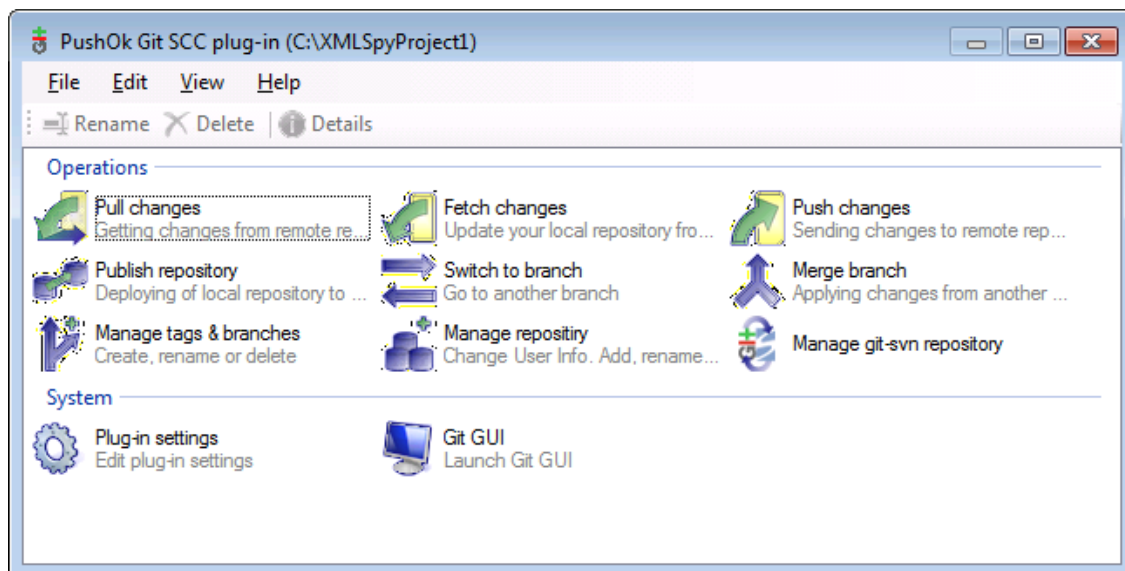
- Local Path:** Text field with "C:\LocalFolder" and a "Browse..." button.
- Scc Provider:** Text field with "Jalindi Igloo" and a "Select..." button.
- Server Name:** Text field with "C:\MySourceControl\Repository" and a "Bind..." button.
- Server Binding:** Text field with "CVSROOT\Emptydir" and an "Unbind" button.
- Logon ID:** Text field with "ala".
- Connected:** A checked checkbox.
- Buttons:** "OK" and "Cancel" buttons at the bottom.

16.4 Source Control with Git

Support for Git as a source control system in UModel is available through a third-party plug-in called **GIT SCC plug-in** (<http://www.pushok.com/software/git.html>).

At the time when this documentation is written, the **GIT SCC plug-in** plug-in is available for experimental use. Registration with the plug-in publisher is required in order to use the plug-in.

The GIT SCC plug-in enables you to work with a Git repository using the commands available in the **Project | Source Control** menu of UModel. Note that the commands in the **Project | Source Control** menu of UModel are provided by the Microsoft Source Control Plug-in API (MSSCCI API), which uses a design philosophy different from Git. As a result, the plug-in essentially intermediates between "Visual Source Safe"-like functionality and Git functionality. On one hand, this means that a command such as **Get latest version** may not be applicable with Git. On the other hand, there are new Git-specific actions, which are available in the "Source Control Manager" dialog box provided by the plug-in (under the **Project | Source Control | Source Control Manager** menu of UModel).



The Source Control Manager dialog box

Other commands that you will likely need to use frequently are available directly under the **Project | Source Control** menu.

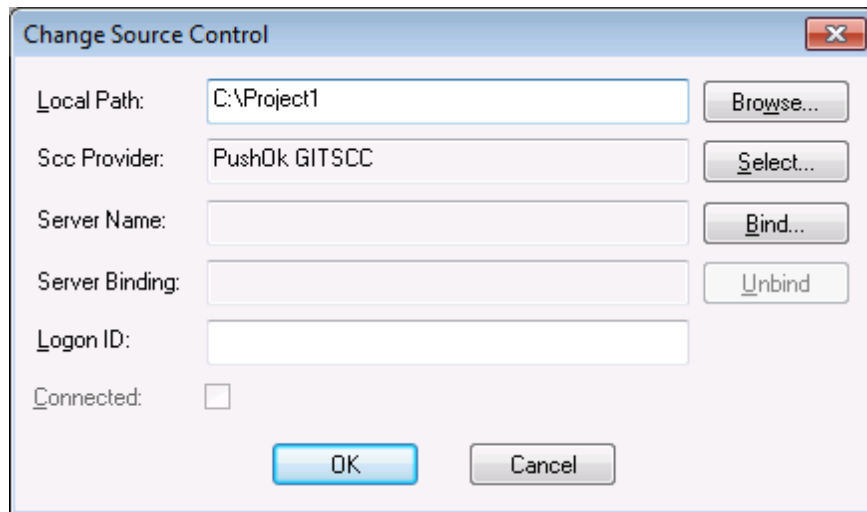
The following sections describe the initial configuration of the plug-in, as well as the basic workflow:

- [Enabling Git Source Control with GIT SCC Plug-in](#)
- [Adding a Project to Git Source Control](#)
- [Cloning a Project from Git Source Control](#)

16.4.1 Enabling Git Source Control with GIT SCC Plug-in

To enable Git source control with UModel, the third-party **PushOK GIT SCC plug-in** must be installed, registered, and selected as source control provider, as follows:

1. Download the plug-in installation file from the publisher's website (<http://www.pushok.com>), run it, and follow the installation steps.
2. On the **Project** menu of UModel, click **Change Source Control**, and make sure **PushOk GITSCC** is selected as source control provider. If you do not see **Push Ok GITSCC** in the list of providers, it is likely that the installation of the plug-in was not successful. In this case, check the publisher's documentation for a solution.



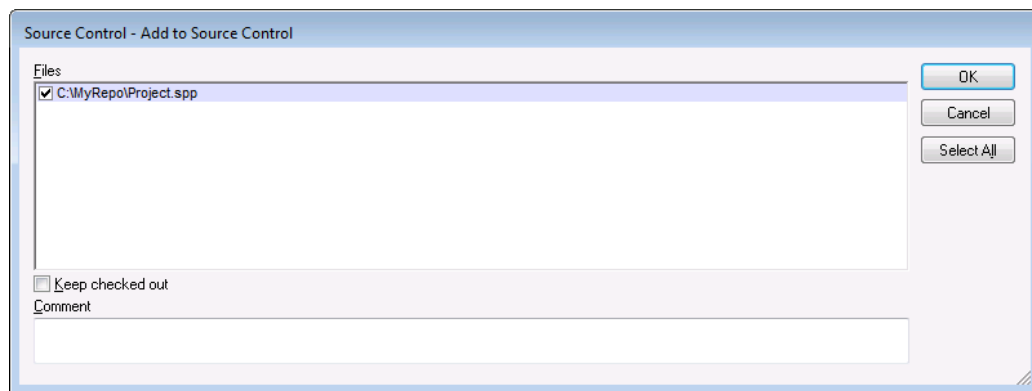
3. When a dialog box prompts you to register the plug-in, click **Registration** and follow the wizard steps to complete the registration process.

16.4.2 Adding a Project to Git Source Control

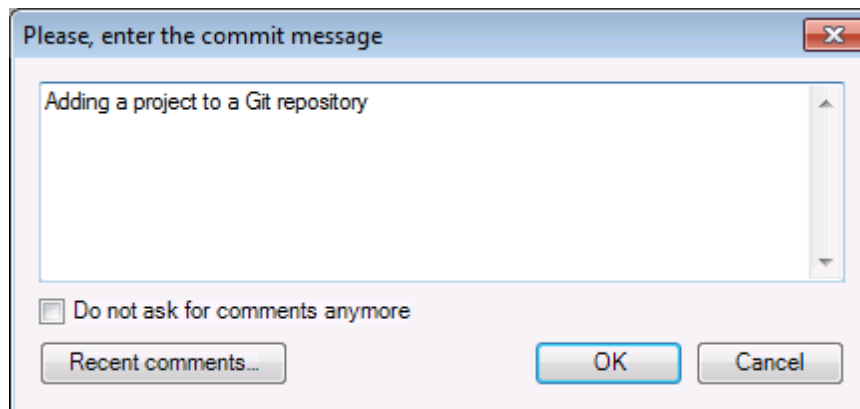
You can save UModel projects as Git repositories. The structure of files or folders that you add to the project would then correspond to the structure of the Git repository.

To add a project to Git source control:

1. Make sure that **PushOK GIT SCC Plug-in** is set as source control provider (see [Enabling Git Source Control with GIT SCC Plug-in](#)).
2. Create a new empty project and make sure that it has no validation errors (that is, the command **Project | Check Project Syntax** does not show any errors or warnings).
3. Save the project to a local folder, for example `C:\MyRepo\Project.ump`.
4. In the **Model Tree** pane, click the **Root** node.
5. On the **Project** menu, under **Source Control**, click **Add to Source Control**.

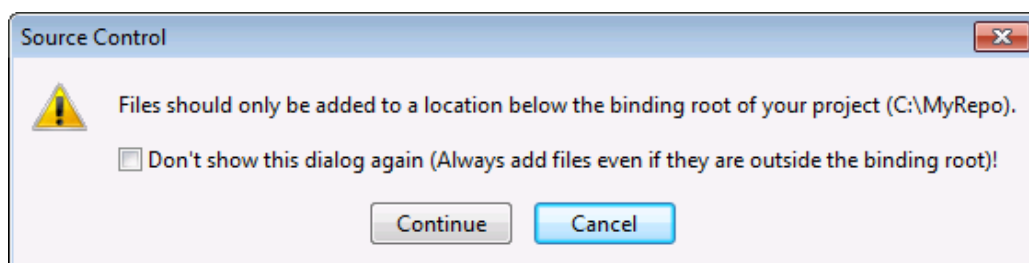


6. Click **OK**.



7. Enter the text of your commit message, and click **OK**.

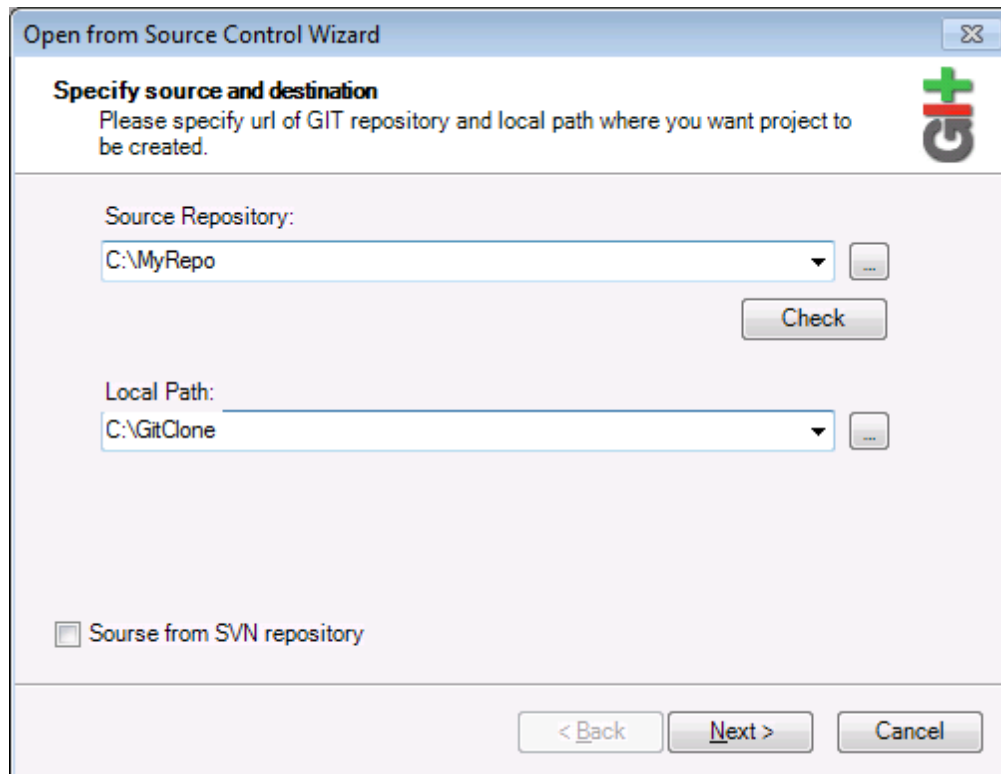
You can now start adding modeling elements (diagrams, classes, packages, and so on) to your project. Note that all project files and folders must be under the root folder of the project. For example, if the project was created in the `C:\MyRepo` folder, then only files under `C:\MyRepo` should be added to the project. Otherwise, if you attempt to add to your project files that are outside the project root folder, a warning message is displayed:



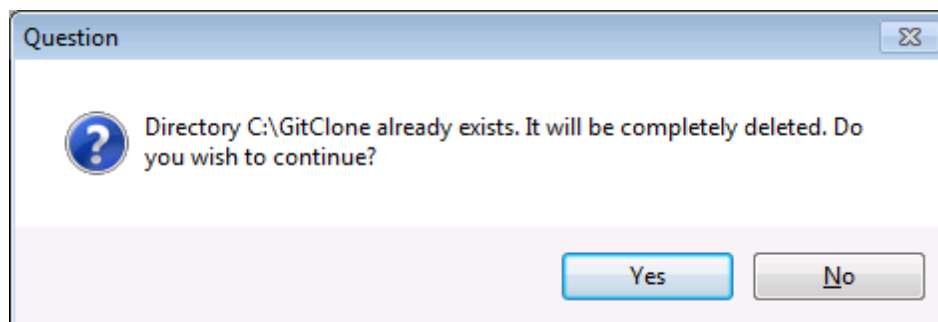
16.4.3 Cloning a Project from Git Source Control

Projects that have been previously added to Git source control (see [Adding a Project to Git Source Control](#)) can be opened from the Git repository as follows:

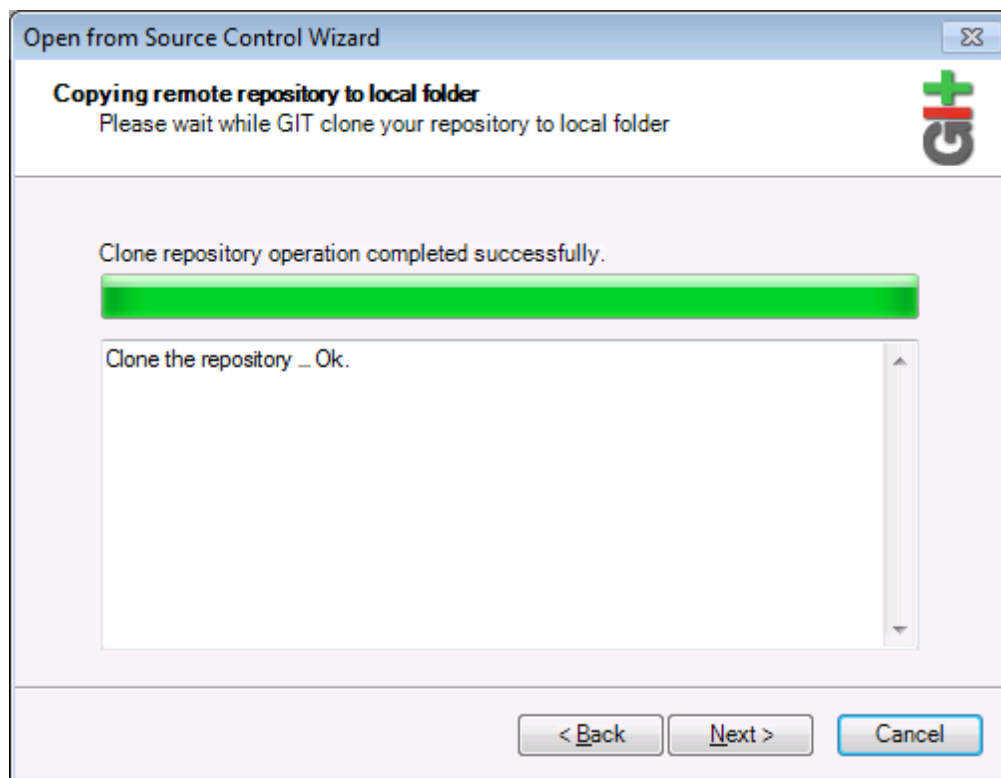
1. Make sure that **PushOK GIT SCC Plug-in** is set as source control provider (see [Enabling Git Source Control with GIT SCC Plug-in](#)).
2. On the **Project** menu, click **Source Control | Open from Source Control**.
3. Enter the path or the URL of the source repository. Click **Check** to verify the validity of the path or URL.



4. Under **Local Path**, enter the path to local folder where you want the project to be created, and click **Next**. If the local folder exists (even if it is empty), the following dialog box opens:



5. Click **Yes** to confirm, and then click **Next**.



6. Follow the remaining wizard steps, as required by your specific case.
7. When the wizard completes, a Browse dialog box appears, asking you to open the UModel Project (*.ump) file. Select the project file to load the project contents into UModel.

Chapter 17

UModel Diagram icons

17 UModel Diagram icons

The following section is a quick guide to the icons that are made available in each of the modeling diagrams.

The icons are split up into two sections:

- **Add** - displays a list of elements that can be added to the diagram.
- **Relationship** - displays a list of relationship types that can be created between elements in the diagram.

17.1 Activity Diagram



Add

Action (CallBehaviorAction)
 Action (CallOperationAction)
 AcceptEventAction
 AcceptEventAction (TimeEvent)
 SendSignalAction

DecisionNode (Branch)
 MergeNode
 InitialNode
 ActivityFinalNode
 FlowFinalNode
 ForkNode (vertical)
 ForkNode (horizontal)
 JoinNode
 JoinNode (horizontal)

InputPin
 OutputPin
 ValuePin

ObjectNode
 CentralBufferNode
 DataStoreNode
 ActivityPartition (horizontal)
 ActivityPartition (vertical)
 ActivityPartition 2-Dimensional

ControlFlow
 ObjectFlow
 ExceptionHandler

Activity
 ActivityParameterNode
 StructuredActivityNode
 ExpansionRegion
 ExpansionNode
 InterruptibleActivityRegion

Note
 Note Link

17.2 Class Diagram

**Relationship:**

Association
Aggregation
Composition
AssociationClass
Dependency
Usage
InterfaceRealization
Generalization

Add:

Package
Class
Interface
Enumeration
Datatype
PrimitiveType
Profile
Stereotype
ProfileApplication
InstanceSpecification

Note
Note Link

17.3 Communication diagram



Add

Lifeline

Message (Call)

Message (Reply)

Message (Creation)

Message (Destruction)

Note

Note Link

17.4 Composite Structure Diagram



Add

Collaboration
CollaborationUse
Part (Property)
Class
Interface
Port

Relationship

Connector
Dependency (Role Binding)
InterfaceRealization
Usage

Note
Note Link

17.5 Component Diagram

**Add:**

Package
Interface
Class
Component
Artifact

Relationship:

Realization
InterfaceRealization
Usage
Dependency

Note
Note Link

17.6 Deployment Diagram

**Add:**

Package

Component

Artifact

Node

Device

ExecutionEnvironment

Relationship:

Manifestation

Deployment

Association

Generalization

Dependency

Note

Note Link

17.8 Object Diagram

**Relationship:**

Association
AssociationClass
Dependency
Usage
InterfaceRealization
Generalization

Add:

Package
Class
Interface
Enumeration
Datatype
PrimitiveType
InstanceSpecification

Note
Note Link

17.9 Package diagram



Add

Package
Profile

Relationship

Dependency
PackageImport
PackageMerge
ProfileApplication

Note
Note Link

17.10 Profile Diagram



Add

Profile

Stereotype

Relationship

Generalization

ProfileApplication

PackageImport

ElementImport

Note

NoteLink

17.11 Protocol State Machine



Add

Simple state
Composite state
Orthogonal state
Submachine state

FinalState

InitialState

EntryPoint

ExitPoint

Choice

Junction

Terminate

Fork

Fork (horizontal)

Join

Join (horizontal)

ConnectionPointReference

Relationship

Protocol Transition

Note

Note link

17.12 Sequence Diagram



Add

Lifeline

CombinedFragment

CombinedFragment (Alternatives)

CombinedFragment (Loop)

InteractionUse

Gate

StateInvariant

DurationConstraint

TimeConstraint

Message (Call)

Message (Reply)

Message (Creation)

Message (Destruction)

Asynchronous Message (Call)

Asynchronous Message (Reply)

Asynchronous Message (Destruction)

Note

Note Link

No message numbering

Simple message numbering

Nested message numbering

Toggle dependent message movement

Toggle automatic creation of replies for messages

Toggle automatic creation of operations in target by typing operation names

17.13 State Machine Diagram



Add

Simple state
Composite state
Orthogonal state
Submachine state

FinalState

InitialState

EntryPoint

ExitPoint

Choice

Junction

Terminate

Fork

Fork (horizontal)

Join

Join (horizontal)

DeepHistory

ShallowHistory

ConnectionPointReference

Relationship

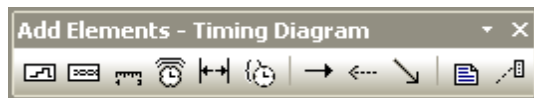
Transition

Note

Note link

Toggle automatic creation of operations in target by typing operation names

17.14 Timing Diagram



Add

Lifeline (State/Condition)

Lifeline (General value)

TickMark

Event/Stimulus

DurationConstraint

TimeConstraint

Message (Call)

Message (Reply)

Asynchronous Message (Call)

Note

Note Link

17.15 Use Case diagram

**Add:**

Package

Actor

UseCase

Relationship:

Association

Generalization

Include

Extend

Note

Note Link

17.16 XML Schema diagram



Add

XSD TargetNamespace
 XSD Schema
 XSD Element (global)
 XSD Group
 XSD ComplexType
 XSD ComplexType (simpleContent)
 XSD SimpleType
 XSD List
 XSD Union
 XSD Enumeration
 XSD Attribute
 XSD AttributeGroup
 XSD Notation
 XSD Import

Relationship

XSD Include
 XSD Redefine
 XSD Restriction
 XSD Extension
 XSD Substitution

Note
 Note link

17.17 Business Process Modeling Notation



Add

Start Event

Intermediate Event

Stop Event

Task

Loop Task

Multi Instance Task

Compensation Task

Collapsed Sub Process

Collapsed Loop Sub Process

Collapsed Multi Instance Sub Process

Collapsed Ad Hoc Process

Collapsed Compensation Sub Process

Expanded Sub Process

Expanded Loop Sub Process

Expanded Multi Instance Sub Process

Expanded Ad Hoc Process

Expanded Compensation Sub Process

Gateway

Inclusive Gateway (OR)

Parallel Gateway (AND)

Data Based Exclusive Gateway (XOR)

Event Based Exclusive Gateway (XOR)

Complex Gateway (Decision/Merge)

Relationship

Sequence Flow

Conditional Flow

Default Flow

Message Flow

Association

Pool

Data Object

Group

Text Annotation

Annotation Association

17.18 Business Process Modeling Notation 2.0



Add

Start Event
Catch Event
Throw Event
End Event

Task
Expanded Sub Process
Collapsed Sub Process
Call Activity
Gateway

Relationship

Sequence Flow
Default Sequence Flow
Conditional Sequence Flow
Message Flow
Association

Pool
Group
Data Object
Data Output
Data Input
Collection Data Object
Data Store
Message

Text Annotation
Annotation Association

17.19 Database Modeling



Add

Table
CheckConstraint
PrimaryKey
ForeignKey
UniqueKey
Index

Relationship

Database Relationship Association
Database Relationship with Attributes

Chapter 18

Menu Reference

18 Menu Reference

The following section lists all the menus and menu options in UModel, and supplies a short description of each.

18.1 File

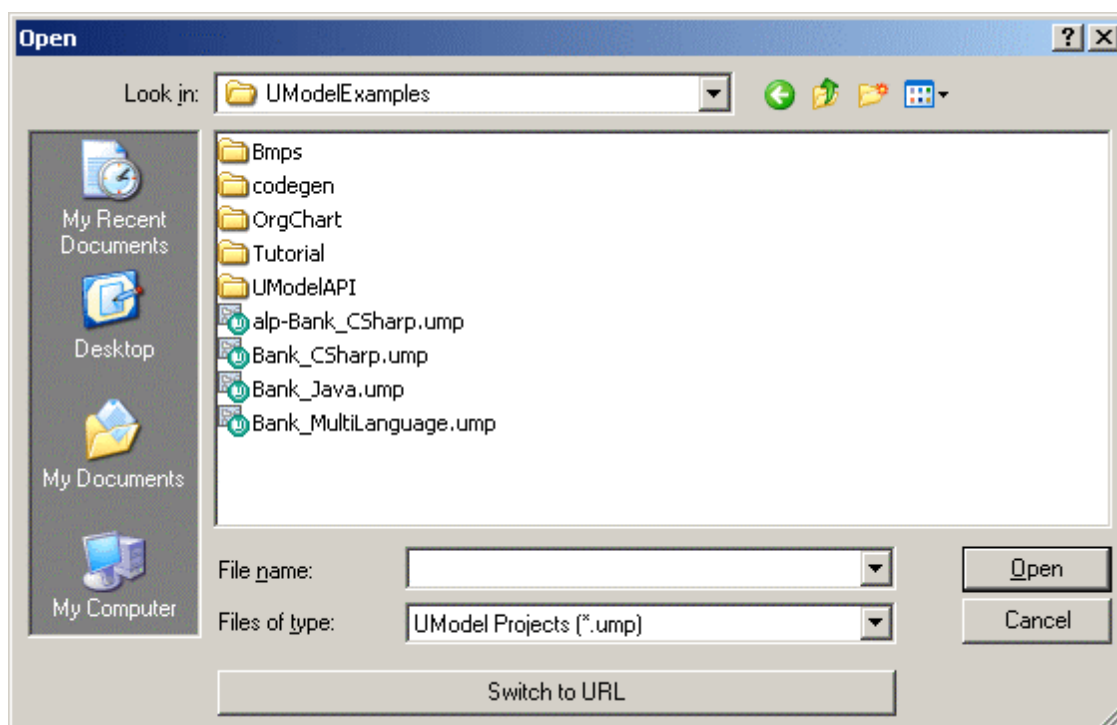
New

Clears the diagram tab, if a previous project exists, and creates a new UModel project.

Open

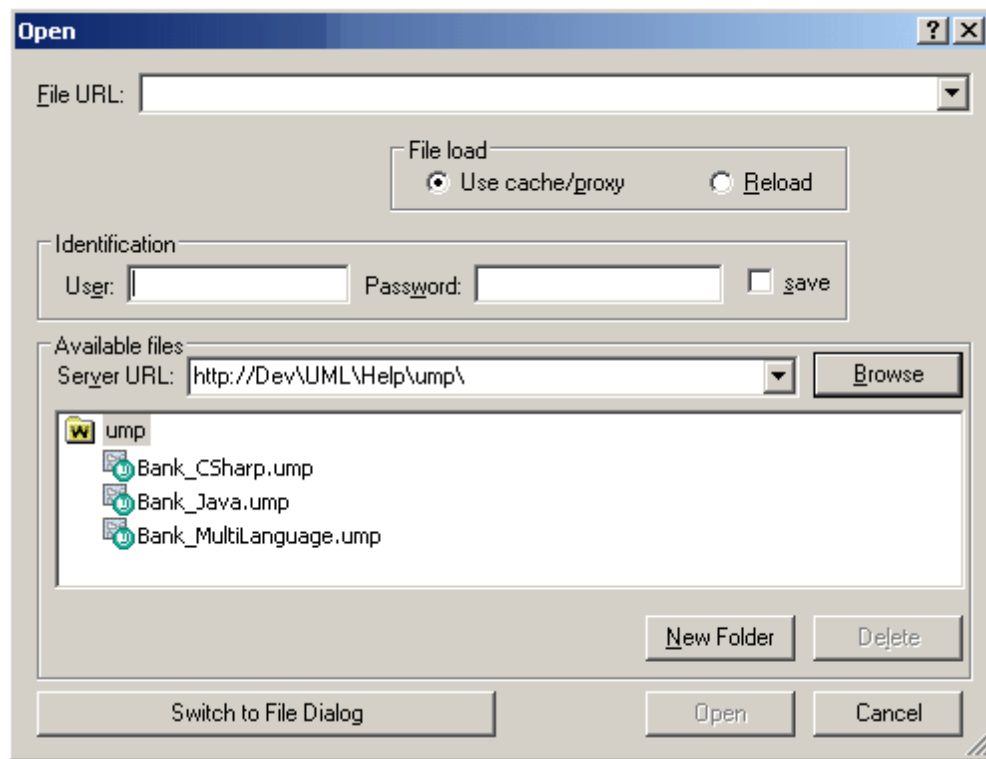
Opens previously defined modeling project. Select a previously saved project file ***.ump** from the Open dialog box.

The standard Open dialog is shown below. Please note the "Switch to URL..." button at the base of the dialog box.



The **Switch to URL** button changes the appearance of the dialog box and allows you to open a UModel project file from an URL.

1. Enter the URL you want to access, in the **Server URL** field.
2. Enter your User-ID in the **User** and **Password** fields, if the server is password protected.
3. Click **Browse** to view and navigate the directory structure of the server.
4. Click the file you want to load into UModel. If the server is a Microsoft® SharePoint® Server, check the *Microsoft® SharePoint® Server* check box. See the Microsoft® SharePoint® Server Notes below for further information about working with files on this type of server.



The file URL now appears in the File URL field at the top of the dialog box.

The **Open** button only becomes active at this point.

5. Click the **Open** button to load the file. The file appears in the main window.

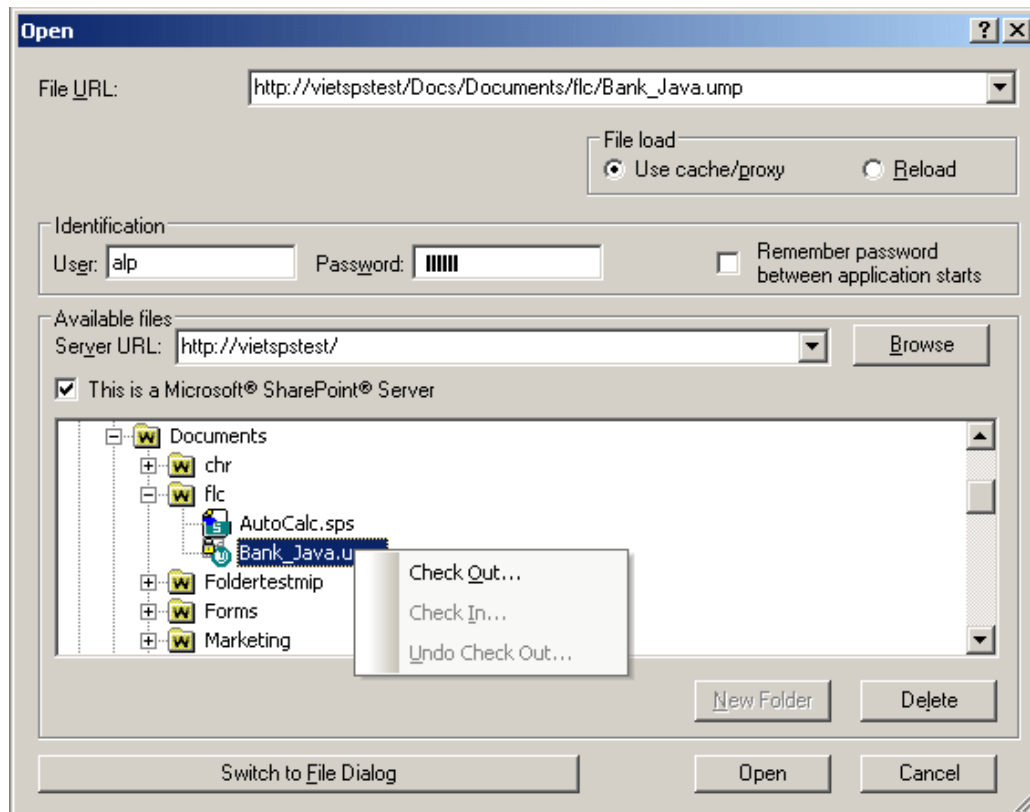
Please note: The Browse function is only available on servers which support WebDAV and on Microsoft SharePoint Servers. The supported protocols are FTP, HTTP, and HTTPS.

To give you more control over the loading process, you can choose to load the file through the local cache or a proxy server (which considerably speeds up the process if the file has been loaded before). Alternatively, you may want to reload the file if you are working, say, with an electronic publishing or database system; select the Reload option in this case.

Microsoft® SharePoint® Server Notes

Note the following points about files on Microsoft® SharePoint® Servers:

- In the directory structure that appears in the Available Files pane, file icons have symbols that indicate the check-in/check-out status of files.



Right-clicking a file pops up a context menu containing commands available for that file.

- The various file icons are shown below:

	Checked in. Available for check-out.
	Checked out by another user. Not available for check-out.
	Checked out locally. Can be edited and checked-in.

- After you check out a file, you can edit it in your Altova application and save it using **File | Save (Ctrl+S)**.
- You can check-in the edited file via the context menu in the Open URL dialog (see *screenshot above*).
- When a file is checked out by another user, it is not available for check out.
- When a file is checked out locally by you, you can undo the check-out with the Undo Check-Out command in the context menu. This has the effect of returning the file unchanged to the server.
- If you check out a file in one Altova application, you cannot check it out in another Altova application. The file is considered to be already checked out to you. The available commands at this point in any Altova application supporting Microsoft® SharePoint® Server will be: **Check In** and **Undo Check Out**.

Reload

Allows you to reload the current project and save, or discard, the changes made since you opened the project file.

Save

Saves the currently active modeling project using the currently active file name.

Save as

Saves the currently active modeling project with a different name, or allows you to give the project a new name if this is the first time you save it.

Save Copy As

Allows you to save a copy of the currently active UModel project with a different file name.

Save Diagram as Image

Opens the "Save as..." dialog box and allows you to save the currently active diagram as a .png, or .emf (enhanced metafile) file. Very large PNG files, in the gigabyte range, can now be saved.

Save all Diagrams as Images

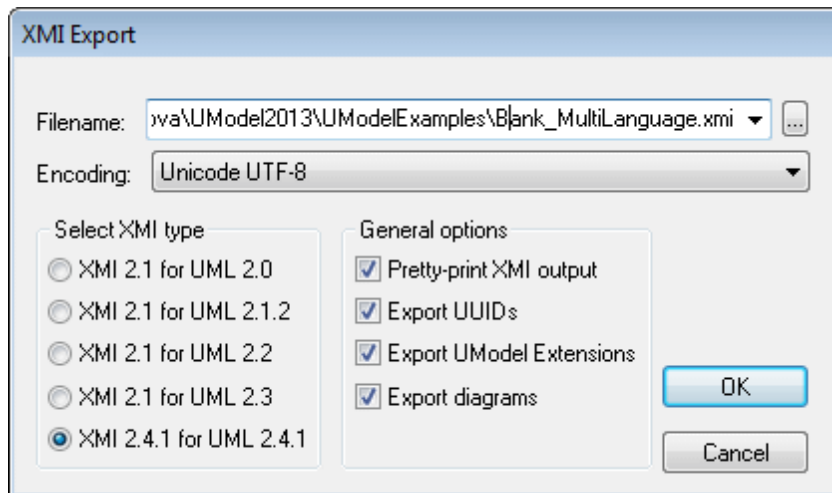
Save all diagrams of the currently active project as a .png, or .emf (enhanced metafile) files.

Import from XMI file

Imports a previously exported XMI file. If the file was produced with UModel, then all extensions etc. will be retained.

Export to XMI file

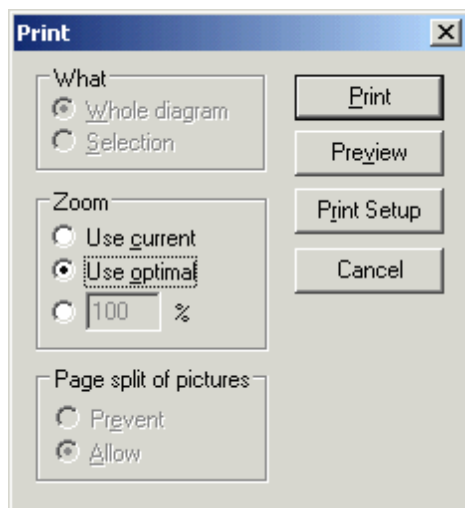
Export the model as an XMI file. You can select the UML version, as well as the specific IDs that you want to export please see [XMI - XML Metadata Interchange](#) for more information.

**Send by Mail**

Opens your default mail application and inserts the current UModel project as an attachment.

Print

Opens the Print dialog box, from where you can print out your modeling project as hard copy.



"Use current", retains the currently defined zoom factor of the modeling project. Selecting this option enables the "Page split of pictures" group.

The Prevent option prevents modeling elements from being split over a page, and keeps them as one unit.

"Use optimal" scales the modeling project to fit the page size. You can also specify the zoom factor numerically.

Print all diagrams

Opens the Print dialog box and prints out all UML diagrams contained in the current project file.

Print Preview

Opens the same Print dialog box with the same settings as described above.

Print Setup

Opens the Print Setup dialog box in which you can define the printer you want to use and the paper settings.

18.2 Edit



Undo

UModel has an unlimited number of "Undo" steps that you can use to retrace you modeling steps.



Redo

The redo command allows you to redo previously undone commands. You can step backward and forward through the undo history using both these commands.

Cut/Copy/Delete

The standard windows Edit commands, allow you to cut, copy, etc., modeling elements, please see "[Cut, copy and paste in UModel Diagrams](#)" for more information.

Paste

using the keyboard shortcut **Ctrl+V**, or "Paste" from the context menu, as well as Paste from the Edit menu, always adds a **new** modeling element to the diagram and to the Model Tree, please see "[Cut, copy and paste in UModel Diagrams](#)".

Paste in Diagram only

using the context menu, i.e. right clicking on the diagram background, only adds a "link/view" of the existing element, to the current diagram and not to the Model Tree, please see "[Cut, copy and paste in UModel Diagrams](#)".

Delete from Diagram only

Deletes the selected modeling elements from the currently active diagram. The deleted elements are not deleted from the modeling project and are available in the Model Tree tab. Note that this option is not available to delete properties or operations from a class, they can be selected and deleted there directly.

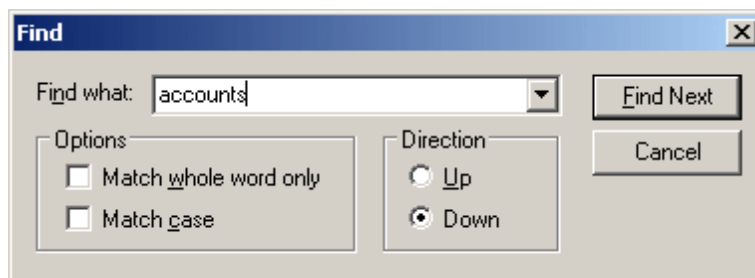
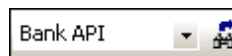
Select all

Select all modeling elements of the currently active diagram. Equivalent to the **Ctrl+A** shortcut.

Find

There are several options you can use to search for modeling elements:

- Use the text box in the Main **title bar**
- Use the menu option Edit | Find
- Press the shortcut **Ctrl+F** to open the find dialog box.



Allows you to search for specific text in:

- Any of the three Model Tree panes: Model Tree, Diagram Tree and Favorites tab.

- The Documentation tab of the Overview pane.
- Any currently active diagram.
- The Messages pane.

Find Next F3

Searches for the next occurrence of the same search string in the currently active tab or diagram.

Find Previous SHIFT+F3

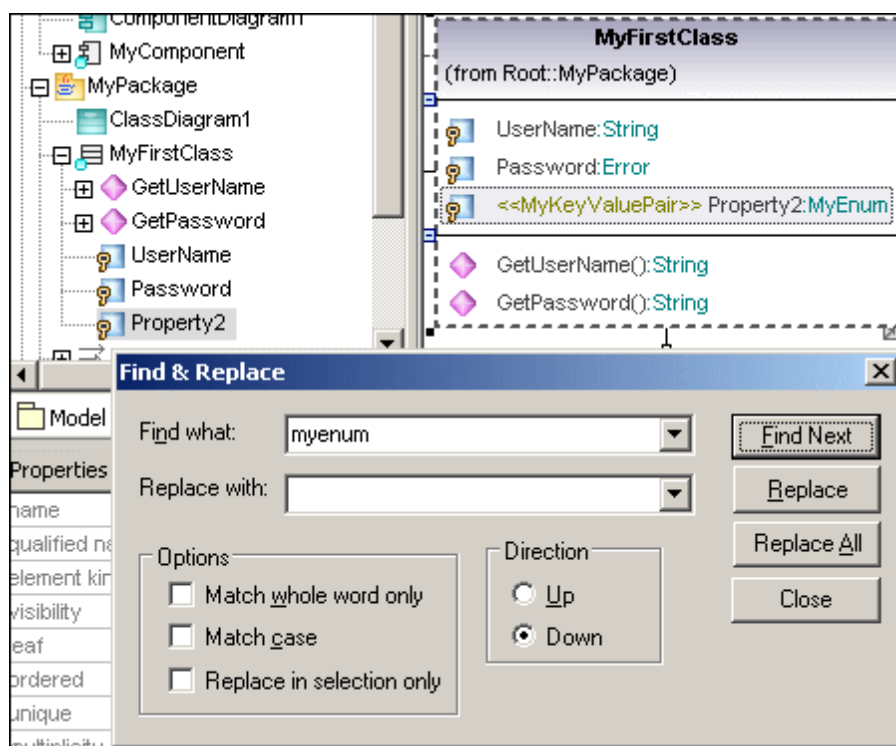
Searches for the previous occurrence of the same search string in the currently active tab or diagram.

Replace

Allows you to search and replace any modelling elements in the project. When the element is found it is highlighted in the diagram as well as in the Model Tree.

Search and replace works in:

- All diagrams
- Any of the three Model Tree panes: Model Tree, Diagram Tree and Favorites tab.
- The Documentation tab of the Overview pane.



Copy as bitmap

Copies the currently active diagram into the clipboard from where you can paste it into the application of your choice.

Please note:

Diagrams are copied into the system clipboard, you have to insert them into another application to see, or get access to them.

Copy selection as bitmap

Copies the currently **selected diagram elements** into the clipboard from where you can paste them into the application of your choice.

18.3 Project

Check Project Syntax...

Checks the UModel project syntax. The project file is checked on multiple levels detailed in the tables below:

Level	Checks if...	Message...
Project level	at least one Java Namespace Root exists	Error
Components	Project file / Directory is set	Error
	If Realization exists	Error
	"Use for code engineering" check box unchecked: no check is performed and syntax check is disabled.	None
Class	Code file name is set.	Error if the local option "Generate missing code file names" is not set.
	If class is nested then no check performed.	Warning if the option is set.
	If contained in a code language namespace	Error
	Type for operation parameter is set	Error
	Type for properties is set	Error
	Operation return type is set	Error
	Duplicate operations (names + parameter types)	Error
	If classes are involved in Realization, only if the class is not nested.	Warning
Interface	Code file name is set.	Error if the local option "Generate missing code file names" is not set.
	Contained in a code language namespace	Warning if the option is set.
	Type for properties are set	Error
	Type for operation parameters are set	Error
	Operation return type is set	Error
	Duplicate operations (names + parameter types)	Error
	If interfaces are involved in a ComponentRealization	Warning
Enumeration	Belongs to Java Namespace Root: gives a warning to say that no code will be generated.	Warning
	Does not belong to Java Namespace Root: no check is performed and syntax check is disabled for the enumeration. No check is performed on	None

contained package

Syntax check for all UML elements involved in code generation

class	Checks name is a valid Java name (no forbidden characters, name is not a keyword)	Error
class property	Checks name is a valid Java name (no forbidden characters, name is not a keyword)	Error
class operation	Checks name is a valid Java name (no forbidden characters, name is not a keyword) Checks for existence of return parameter	Error
class operation parameter	Checks name is a valid Java name (no forbidden characters, name is not a keyword) Checks type has a valid Java type name	Error
interface	Checks name is a valid Java name (no forbidden characters, name is not a keyword)	Error
interface operation	Checks name is a valid Java name (no forbidden characters, name is not a keyword)	Error
interface operation parameter	Checks name is a valid Java name (no forbidden characters, name is not a keyword)	Error
interface properties	Checks name is a valid Java name (no forbidden characters, name is not a keyword)	Error
package with stereotype namespace	Checks name is a valid Java name (no forbidden characters, name is not a keyword)	Error
package without stereotype namespace	no element to check	None
class	multiple inheritance	Error

Please note:

Constraints on model elements are not checked, as they are not part of the Java code generation process. Please see "[constraining model elements](#)" for more information.

Source control

See [Source control systems](#) for detailed information on source control servers and clients and how to use them.

Import Source Directory...

Opens the Import Source Directory wizard shown below. Please see "[Round-trip engineering \(code - model - code\)](#)" for a specific example.

Import Source Project...

Opens the Import Source Project wizard shown below. Clicking the browse  button allows you to select the project file and the specific project type. Please see "[Importing source code into projects](#)" for a specific example.

Java projects:

- JBuilder **.jpx**, Eclipse **.project** project files, as well as NetBeans (project.xml) are currently supported.

C# projects:

- MS Visual Studio projects, **csproj**, **csdprj**..., as well as
- Borland **.bdsproj** project files

VB.NET projects:

- MS Visual Studio projects, **vbproj**, **vbdproj**.

Import Binary Types

Opens the Import Binary Types dialog box allowing you to import Java, C#, and VB binary files. Please see "[Importing Java, C#, and VB binaries](#)" for more information.

Import XML Schema directory

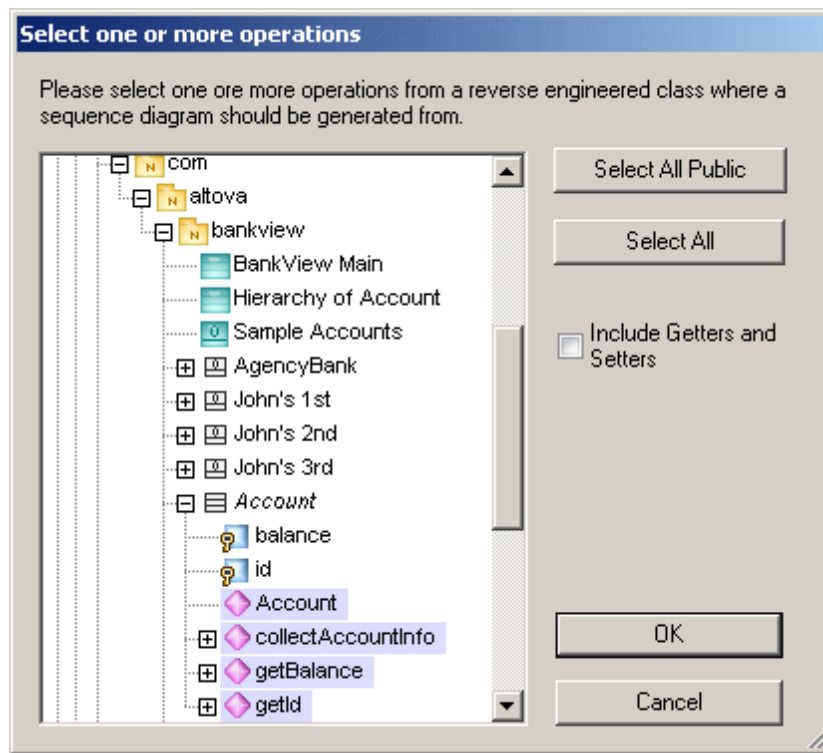
Opens the Import XML Schema Directory allowing you to import all XML Schemas in that directory and optionally all XML Schemas in any of the subfolders.

Import XML Schema File

Opens the Import XML Schema File dialog box allowing you to import schema files. Please see "[XML Schema Diagrams](#)" for more information.

Generate Sequence diagrams...

Opens the Select Operations dialog box, in which you select (multiple) operations that are to be the basis of the generated Sequence diagram.



Having selected the operations and clicked OK, the Sequence Diagram Generation dialog box is opened allowing you to further define the generation settings.

Merge Program Code from UModel Project

Opens the Synchronization Settings dialog box with the "Code from Model" tab active. Clicking the Project Settings button allows you to select the specific programming language settings.

Merging or overwriting code

Assuming that code has been generated once from a model, and changes have since been made to both model and code e.g.:

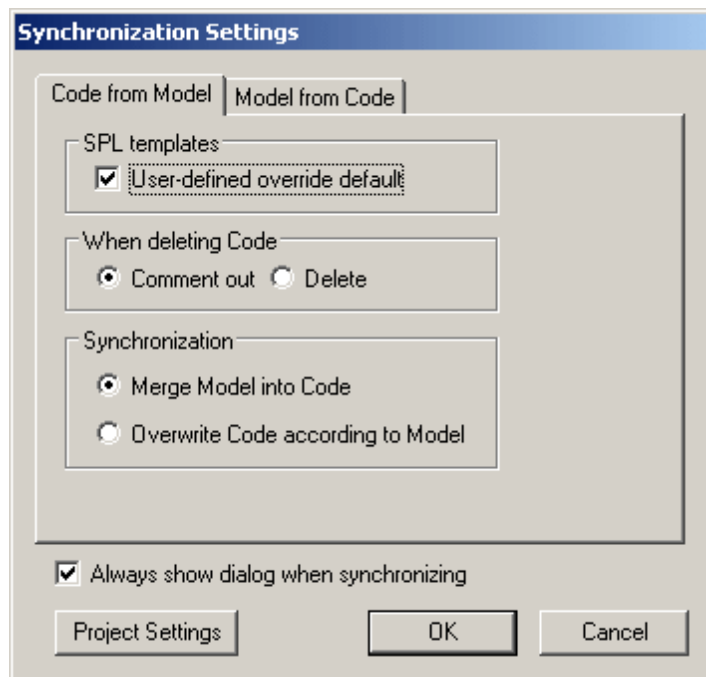
- Model elements have been added in UModel e.g. a new class X
- A new class has been added to the external code e.g. class Y

Merging (model into code) means that:

- the newly added class Y in the external code is **retained**
- the newly added class X, from UModel, is added to the code.

Overwriting (code according to model) means that:

- the newly added class Y in the external code is **deleted**
- the newly added class X, from UModel, is added to the code.



Merge UModel Project from Program Code

Opens the Synchronization Settings dialog box with the "**Model from Code**" tab active. Clicking the Project Settings button allows you to select the specific programming language settings.

Merging or overwriting code

Assuming that code has been generated once from a model, and changes have since been made to both model and code e.g.:

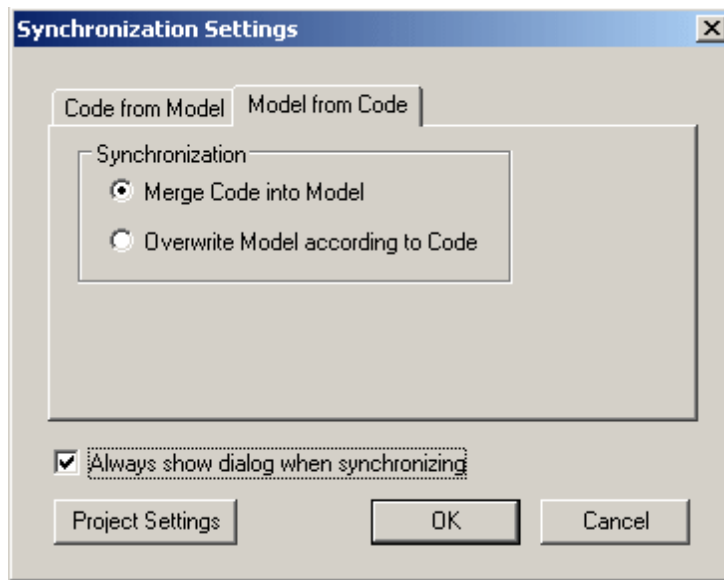
- Model elements have been added in UModel e.g. a new class X
- A new class has been added to the external code e.g. class Y

Merging (code into model) means that:

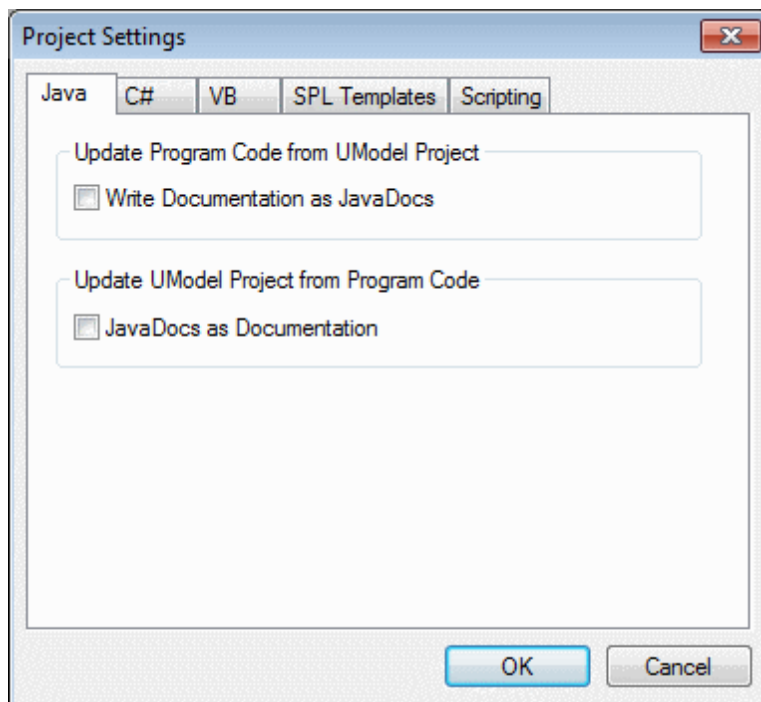
- the newly added class X in UModel, is retained
- the newly added class Y, from the external code, is added to the model

Overwriting (Model according to code) means that:

- the newly added class X in UModel is deleted
- the newly added class Y, from the external code, is added to the model

**Project settings**

Allows you to define the specific languages settings for your project.

**Synchronization Settings...**

Opens the Synchronization Settings dialog box as shown in the screenshots above.

Merge Project...

Merges two UModel project files into one model. The first file you open is the one the second file will be merged into. Please see [Merging UModel projects](#) for more information.

Include Subproject

See [Including other UModel projects](#).

Open Subproject as project

Opens the selected subproject as a new project.

Clear Messages

Clears the syntax check and code merging messages, warnings and errors from the Messages window.

Please note:

Errors are generally problems that must be fixed before code can be generated, or the model code can be updated during the code engineering process. Warnings can generally be deferred until later. Errors and warnings are generated by the syntax checker, the compiler for the specific language, the UModel parser that reads the newly generated source file, as well as during the import of XML files.

Generate documentation

Allows you to generate documentation for the currently open project in HTML, Microsoft Word, and RTF formats (see [Generating UML documentation](#)).

List Elements not used in any Diagram

Creates a list of all elements not used in any diagram in the project.

List shared Packages

Lists all shared packages of the current project.

List included Packages

Lists all include packages in the current project. Java Profile (Java Profile.ump) and Java Lang (Java Lang.ump) are automatically supplied in the Bankview example supplied with UModel.

18.4 Layout

The commands of the Layout menu allow you to line up and align the elements of your modeling diagrams.

When using the marquee (drag on the diagram background) to mark several elements, the element with the dashed outline becomes the "active" element, i.e. the last marked element. All alignment commands use this element as the origin, or basis for the following alignment commands.

Align:

The align command allows you to align modeling elements along their borders, or centers depending on the specific command you select.

Space evenly:

This set of commands allow you to space selected elements evenly both horizontally and vertically.

Make same size:

This set of commands allow you to adjust the width and height of selected elements based on the active element.

Line up:

This set of commands allow you to line up the selected elements vertically or horizontally.

Line Style:

This set of commands allow you to select the type of line used to connect the various modeling elements. The lines can be any type of dependency, association lines used in the various model diagrams.

Autosize:

This command resizes the selected elements to their respective optimal size(s).

Autolayout all:

This command allows you to choose the type of presentation of the modeling elements in the UML diagram tab.

"Force directed"

displays the modeling elements from a centric viewpoint.

"Hierarchic"

displays elements according to their relationships, superclass - derived class etc.

"Block"

displays elements grouped by element size in rectangular fashion.

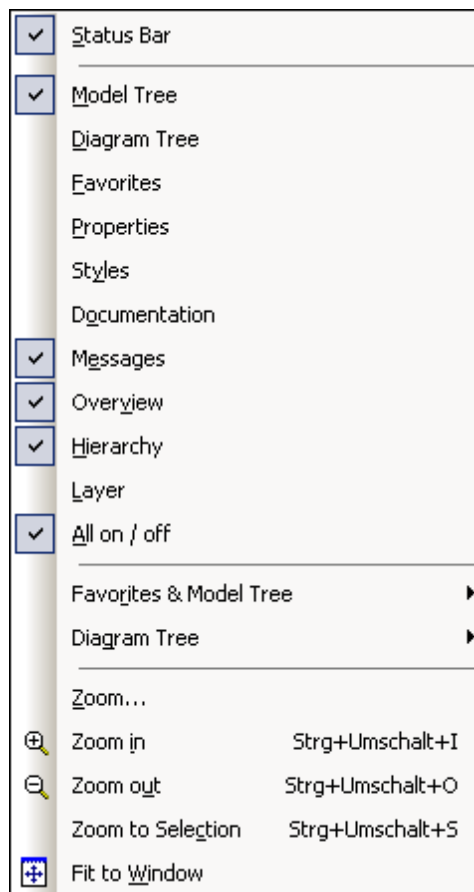
Reposition text labels:

Repositions modeling element names (of the selected elements) to their default positions.

18.5 View

The commands available in this menu allow you to:

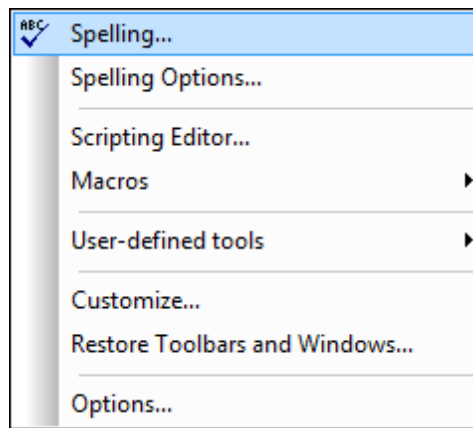
- **Switch**/activate tabs of the various panes
- Define the modeling element **sort criteria** of the Model Tree and Favorites tab
- Define the **grouping criteria** of the diagrams in the Diagram Tree tab
- Show or hide specific UML elements in the Favorites and Model Tree tab
- Define the **zoom** factor of the current diagram.



18.6 Tools

The tools menu allows you to:

- Start spell checking your UModel project and define the spell checker options.
- Access the [Scripting Environment](#) of UModel. You can create, manage and store your own forms, macros and event handlers.
- View and execute the currently defined [macros](#).
- [Customize](#) your version: define your own toolbars, keyboard shortcuts, menus, and macros.
- Restore Toolbars and Windows to their default state.
- Define the global program [settings/options](#).



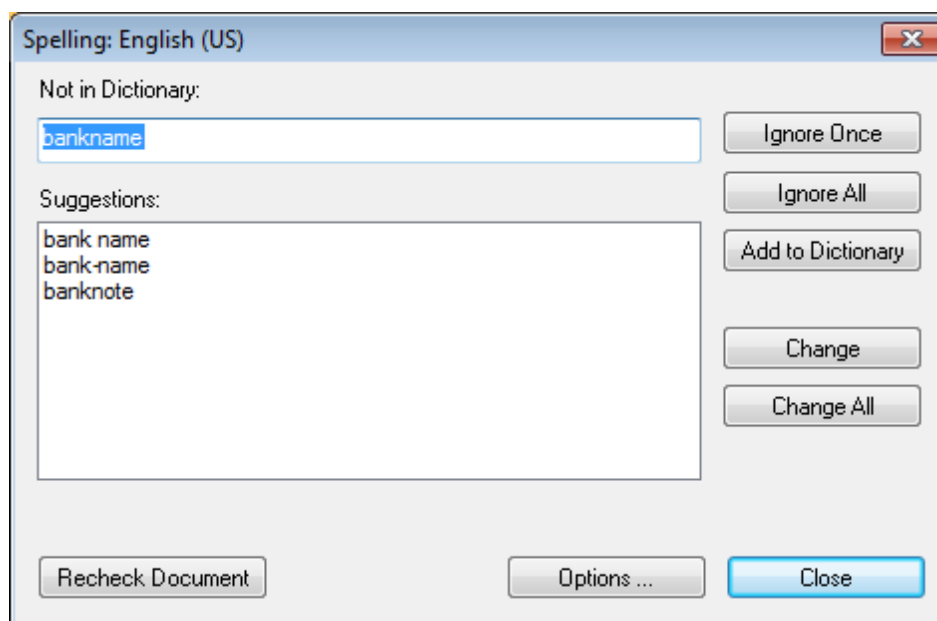
18.6.1 Spelling...

Select Tools | Spelling to start the spell check process. The standard spell checker options are available in this dialog box.

To define the specific spell checker options, click the "Options" button, or select the menu option Tools | Spelling Options.

Please note:

You can spell check entries in the Model Tree as well as in UML diagrams. Right clicking in the Model Tree and selecting "Documentation Spelling" spell checks the comments and notes of the Model Tree.



Not in Dictionary

This text box contains the word that cannot be found in either the selected language dictionary or user dictionary.

Suggestions

This list box displays words resembling the unknown word (supplied from the language and user dictionaries). Double-clicking a word in this list automatically inserts it in the document and continues the spell-checking process.

Ignore once

This command allows you to continue checking the document while ignoring the first occurrence of the unknown word. The same word will be flagged again if it appears in the document.

Ignore all

This command ignores all instances of the unknown word in the whole document.

Add to dictionary

This command adds the unknown word to the **user dictionary**. You can access the user dictionary (in order to edit it) via the Options dialog.

Change

This command replaces the currently highlighted word in the XML document with the (edited) word in the *Not in Dictionary* text box.

Change all

This command replaces all occurrences of the currently highlighted word in the XML document with the (edited) word in the *Not in Dictionary* text box.

Recheck Document

The "Recheck Document" button restarts the check from the beginning of the document.

Adding dictionaries for the spellchecker

For each dictionary language there are two Hunspell dictionary files that work together: a `.aff` file and `.dic` file. All language dictionaries are installed in a `Lexicons` folder at the following location:

On Windows 7 and Windows 8: `C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons`

On Windows XP: `C:\Documents and Settings\All Users\Application Data\Altova\SharedBetweenVersions\SpellChecker\Lexicons`

Within the `Lexicons` folder, different language dictionaries are each stored in different folder:

`<language name>\<dictionary files>`. For example, on a Windows 7 or Windows 8 machine, files for the two English-language dictionaries (English (British) and English (US)) will be stored as below:

```
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English  
(British)\en_GB.aff  
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English  
(British)\en_GB.dic  
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (US)  
\en_US.dic  
C:\ProgramData\Altova\SharedBetweenVersions\SpellChecker\Lexicons\English (US)  
\en_US.dic
```

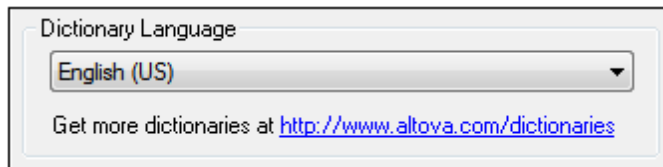
In the Spelling Options dialog, the dropdown list of the *Dictionary Language* combo box displays the language dictionaries. These dictionaries are those available in the `Lexicons` folder and have the same names as the language subfolders in the `Lexicons` folder. For example, in the case of the English-language dictionaries shown above, the dictionaries would appear in the Dictionary Language combo box as: *English (British)* and *English (US)*.

All installed dictionaries are shared by the different users of the machine and the different major versions of Altova products (whether 32-bit or 64-bit).

You can add dictionaries for the spellchecker in two ways, neither of which require that the files

be registered with the system:

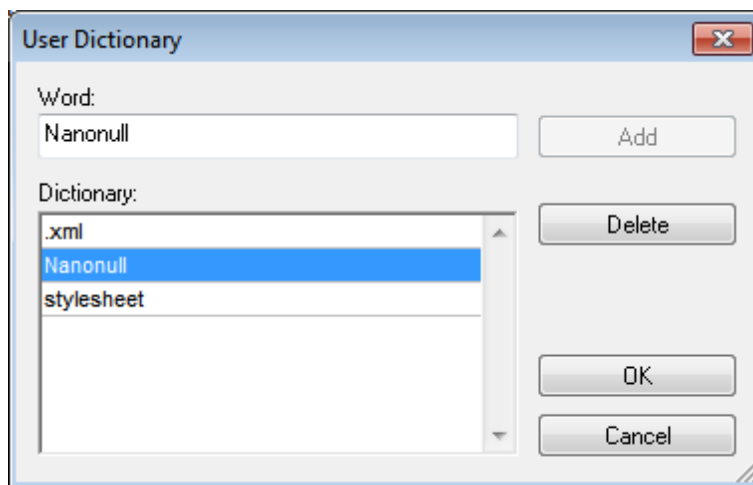
- By adding Hunspell dictionaries into a new subfolder of the `Lexicons` folder. Hunspell dictionaries can be downloaded, for example, from <http://wiki.services.openoffice.org/wiki/Dictionaryes> or <http://extensions.services.openoffice.org/en/dictionaries>. (Note that OpenOffice uses the zipped `oxt` format. So change the extension to `.zip` and unzip the `.aff` and `.dic` file to the language folders in the `Lexicons` folder. Also note that Hunspell dictionaries are based on Myspell dictionaries. So Myspell dictionaries can also be used.)
- By using the [Altova dictionary installer](#), which installs a package of multiple language dictionaries by default to the correct location on your machine. The installer can be downloaded via the link in the Dictionary language pane of the Spelling Options dialog (see *screenshot below*). Installation of the dictionaries must be done with administrator rights, otherwise installation will fail with an error.



Note: It is your choice as to whether you agree to the terms of the license applicable to the dictionary and whether the dictionary is appropriate for your use with the software on your computer.

Working with the user dictionary

Each user has one user dictionary, in which user-allowed words can be stored. During a spellcheck, spellings are checked against a word list comprising the words in the language dictionary and the user dictionary. You can add words to and delete words from the user dictionary via the User Dictionary dialog (*screenshot below*). This dialog is accessed by clicking the User Dictionary button in the Spelling Options dialog (see *second screenshot in this section*).



To add a word to the user dictionary, enter the word in the Word text box and click **Add**. The word will be added to the alphabetical list in the Dictionary pane. To delete a word from the dictionary, select the word in the Dictionary pane and click **Delete**. The word will be deleted from the Dictionary pane. When you have finished editing the User Dictionary dialog, click **OK** for the

changes to be saved to the user dictionary.

Words may also be added to the User Dictionary during a spelling check. If an unknown word is encountered during a spelling check, then the [Spelling dialog](#) pops up prompting you for the action you wish to take. If you click the **Add to Dictionary** button, then the unknown word is added to the user dictionary.

The user dictionary is located at:

On Windows 7 and Windows 8: C:\Users\<user>\Documents\Altova\SpellChecker\Lexicons\user.dic

On Windows XP: C:\Documents and Settings\<user>\My Documents\Altova\SpellChecker\Lexicons\user.dic

18.6.2 Spelling options...

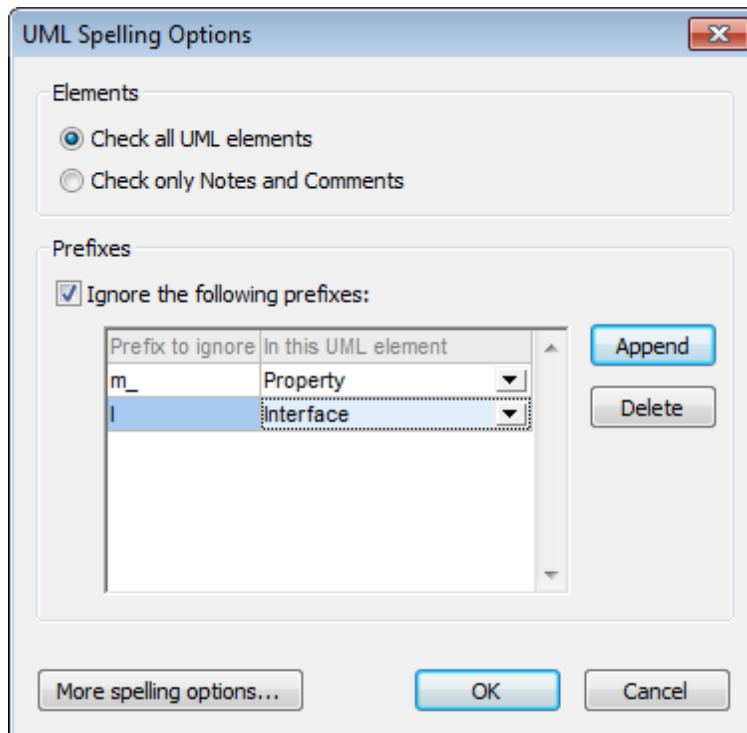
Elements

This group allows you to choose between spell checking all UML elements, or only the Notes and Comments objects.

Prefixes

Double clicking in the "Prefix to ignore" column lets you enter the prefixes, of specific UML elements, you want to ignore during spell checking, e.g. m_ for properties, and I for Interfaces.

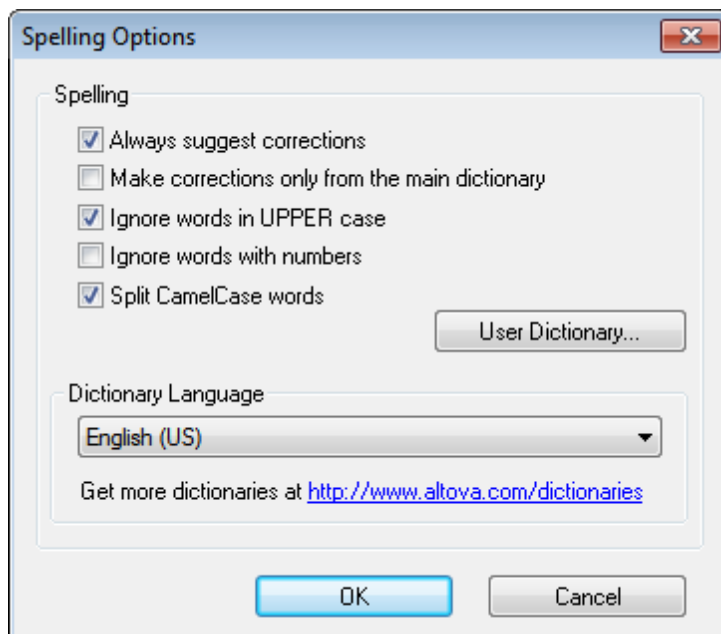
The "Append" button adds a new row to the Prefixes table. "Delete" deletes the currently active row.



Clicking the "More spelling options..." button opens the Spelling Options dialog box shown below.

More Spelling Options

The Spelling Options dialog is used to define global spellchecker options.



Always suggest corrections:

Activating this option causes suggestions (from both the language dictionary and the user dictionary) to be displayed in the Suggestions list box. Disabling this option causes no suggestions to be shown.

Make corrections only from main dictionary:

Activating this option causes only the language dictionary (main dictionary) to be used. The user dictionary is not scanned for suggestions. It also disables the **User Dictionary** button, preventing any editing of the user dictionary.

Ignore words in UPPER case:

Activating this option causes all upper case words to be ignored.

Ignore words with numbers:

Activating this option causes all words containing numbers to be ignored.

Split CamelCase words

CamelCase words are words that have capitalization within the word. For example the word "CamelCase" has the "C" of "Case" capitalized, and is therefore said to be CamelCased. Since CamelCased words are rarely found in dictionaries, the spellchecker would flag them as errors. To avoid this, the *Split CamelCase words* option splits CamelCased words into their capitalized components and checks each component individually. This option is checked by default.

Dictionary Language

Use this combo box to select the dictionary language for the spellchecker. The default selection is US English. Other language dictionaries are available for download free of charge from the [Altova website](http://www.altova.com/dictionaries).

18.6.3 Scripting Editor

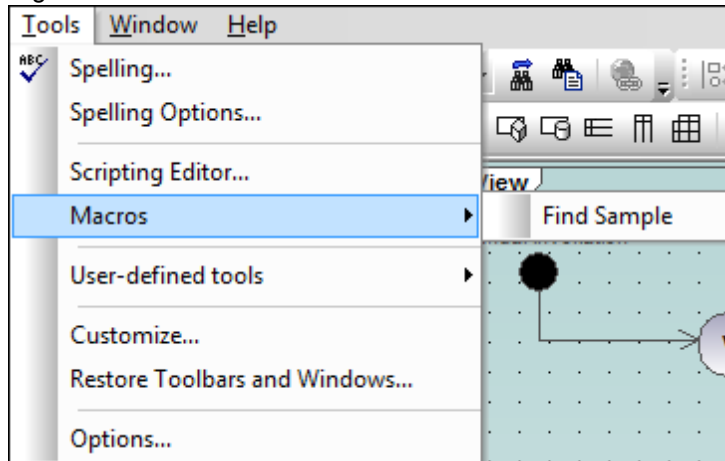
The Scripting Editor command opens the Scripting Editor window. How to work with the Scripting Editor is described in the [Scripting section](#) of this documentation.

Note: The .NET Framework version 2.0 or higher will have to be installed on your machine in order for the Scripting Editor to run.

18.6.4 Macros

Placing the mouse cursor over the Macros entry, opens a flyout menu containing a list of the currently defined macros in the Scripting Project. The active Scripting Project is defined in the [Scripting tab](#) of the Options dialog box.

E.g.



18.6.5 User-defined Tools

Placing the cursor over the **User-defined Tools** command rolls out a sub-menu containing custom-made commands that use external applications. You can create these commands in the [Tools tab](#) of the Customize dialog. Clicking one of these custom commands executes the action associated with this command.

The **User-Defined Tools | Customize** command opens the [Tools tab](#) of the Customize dialog (in which you can create the custom commands that appear in the menu of the **User-Defined Tools** command.)

18.6.6 Customize...

The customize command lets you customize UModel to suit your personal needs.

[Commands](#)

[Toolbars](#)

[Tools](#)

[Keyboard](#)

[Menu](#)

[Macros](#)

[Plug-ins](#)

[Options](#)

Commands

The Commands tab allows you customize your menus or toolbars.

To add a command to a toolbar or menu:

1. Open this dialog box using **Tools | Customize**.
 2. Select the command category in the Categories list box. The commands available appear in the Commands list box.
 3. Click on a command in the commands list box and drag "it" to an to an existing menu or toolbar.
 4. An I-beam appears when you place the cursor over a valid position to drop the command.
 5. Release the mouse button at the position you want to insert the command.
- A small button appears at the tip of mouse pointer when you drag a command. The check mark below the pointer means that the command cannot be dropped at the current cursor position.
 - The check mark disappears whenever you can drop the command (over a tool bar or menu).
 - Placing the cursor over a menu when dragging, opens it, allowing you to insert the command anywhere in the menu.
 - Commands can be placed in menus or tool bars. If you created you own toolbar you can populate it with your own commands/icons.

Please note:

You can also edit the commands in the [context menus](#) (right click anywhere opens the context menu), using the same method. Click the Menu tab and then select the specific context menu available in the Context Menus combo box.

To delete a command or menu:

1. Open this dialog box using **Tools | Customize**.
 2. Click on the menu entry or icon you want to delete, and drag with the mouse.
 3. Release the mouse button whenever the check mark icon appears below the mouse pointer.
- The command, or menu item is deleted from the menu or tool bar.

Toolbars

The Toolbars tab allows you to activate or deactivate specific toolbars, as well as create your own specialized ones.

Toolbars contain symbols for the most frequently used menu commands. For each symbol you

get a brief "tool tip" explanation when the mouse cursor is directly over the item and the status bar shows a more detailed description of the command.

You can drag the toolbars from their standard position to any location on the screen, where they appear as a floating window. Alternatively you can also dock them to the left or right edge of the main window.

To activate or deactivate a toolbar:

1. Click the check box to activate (or deactivate) the specific toolbar.

To create a new toolbar:

1. Click the **New...** button, and give the toolbar a name in the Toolbar name dialog box.
2. Add commands to the toolbar using the [Commands](#) tab of the Customize dialog box.

To reset the Menu Bar

- Click the Menu Bar entry and
- Click the **Reset** button, to reset the menu commands to the state they were when installed.

To reset all toolbar and menu commands

- Click the **Reset All** button, to reset all the toolbar commands to the state they were when the program was installed. A prompt appears stating that all toolbars and menus will be reset.
- Click Yes to confirm the reset.

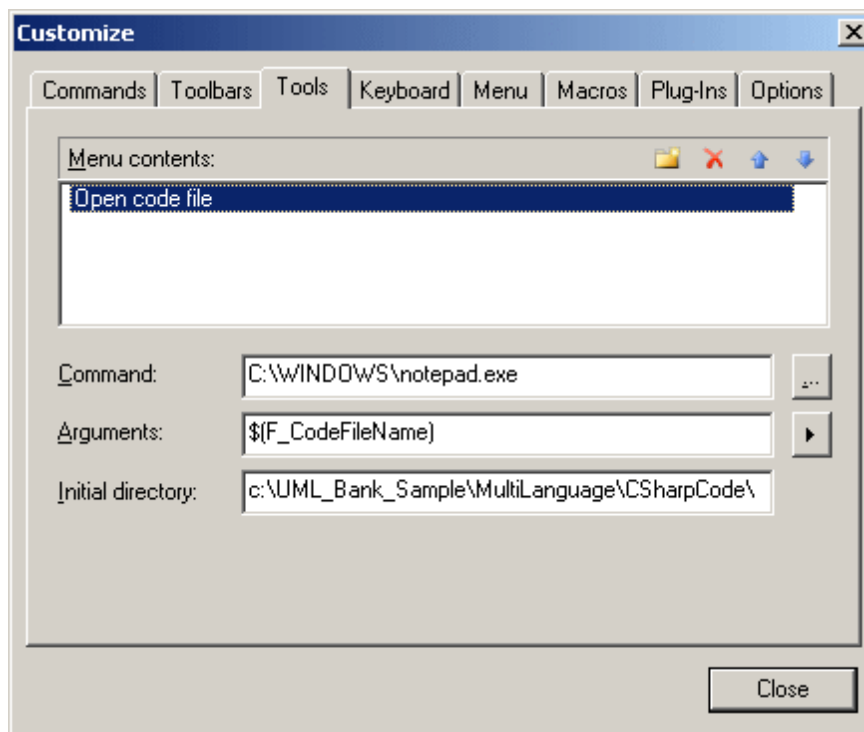
Show text labels:

This option places explanatory text below toolbar icons when activated.

Tools

The Tools tab allows you to create your own menu **entries**, and associated arguments, to start external tools from within UModel via the Tools menu.

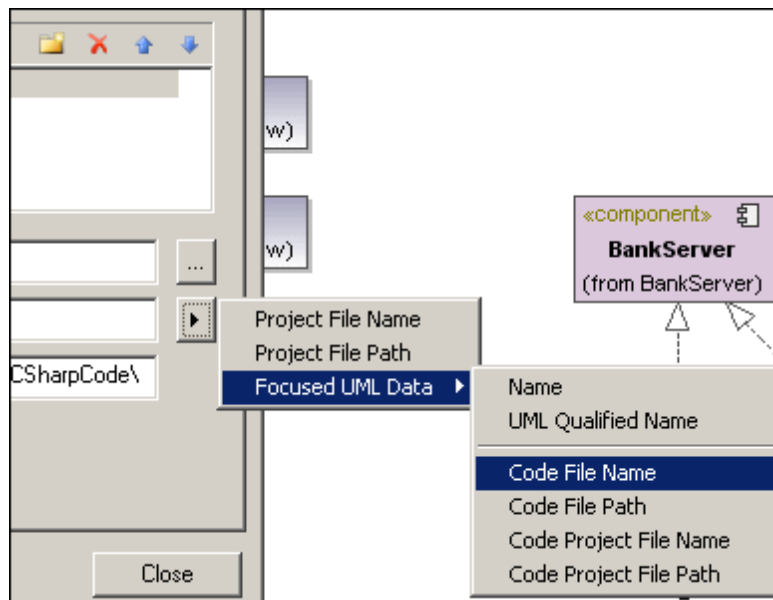
Click the folder icon (in the Menu contents: bar) to add a new menu entry, and use the Command field to associate it to an application.



The Tools tab now also supports the definition of **Arguments**. Arguments are variables that are assigned specific values when the external tool is started using the menu item.

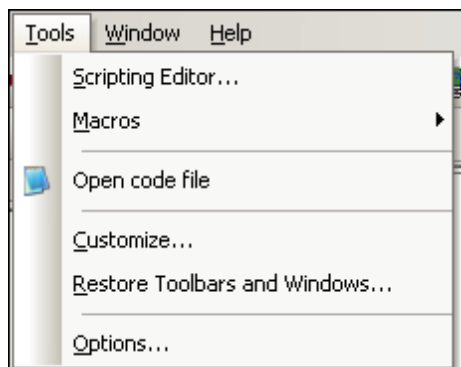
E.g. Open the source code file of the currently selected UML class in Notepad.

1. Select the menu item **Tools | Customize** and click the **Tools** tab.
2. Enter the name and path of the external application e.g. C:\...\notepad.exe.
3. Click the Flyout button of the Arguments field and select the argument you want to use, e.g. Code File Name.

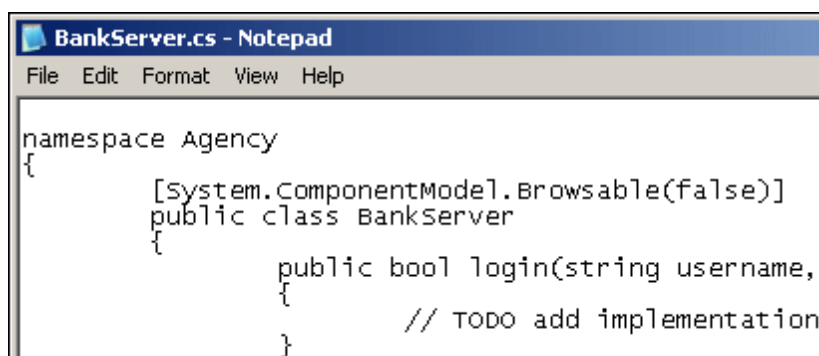


4. Click the Close button to finish.

5. Click the Tools menu and select "Open code file".



The BankServer.cs file is opened in Notepad and is ready to be edited.



UModel Arguments

Project File Name

The file name of the active UModel project file, e.g. "Test.ump".

Project File Path

The absolute file path of the active UModel project file, e.g. "c:\MyDirectory\Test.ump".

Focused UML Data – Name

The name of the currently focused UML element, e.g. "Class1".

Focused UML Data – UML Qualified Name

The qualified name of the currently focused UML element, e.g. "Package1::Package2::Class1".

Focused UML Data – Code File Name

The code file name of the currently focused UML class, interface or enumeration as shown in the Property window (relative to the realizing component), e.g. "Class1.cs" or "MyNamespace \Class1.Java".

Focused UML Data – Code File Path

The code file path of the currently focused UML class, interface or enumeration as shown in the Property window, e.g. "C:\Temp\MySource\Class1.cs".

Focused UML Data – Code Project File Name

The file name of the code project the currently focused UML class, interface or enumeration

belongs to.

The code project file name can be relative to the UModel project file and is the same as shown in the Properties of the component, e.g. "C:\Temp\MySource\MyProject.vcproj" or "MySource\MyProject.vcproj".

Focused UML Data – Code Project File Path

The file path of the code project the currently focused UML class, interface or enumeration belongs to, e.g. "C:\Temp\MySource\MyProject.vcproj".

Keyboard

The Keyboard tab allows you to define (or change) keyboard shortcuts for any command.

To assign a new Shortcut to a command:

1. Select the commands category using the **Category** combo box.
2. Select the **command** you want to assign a new shortcut to, in the Commands list box
3. Click in the "**Press New Shortcut Key:**" text box, and press the shortcut keys that are to activate the command.
The shortcuts appear immediately in the text box. If the shortcut was assigned previously, then that function is displayed below the text box.
4. Click the **Assign** button to permanently assign the shortcut.
The shortcut now appears in the Current Keys list box.
(To **clear** this text box, press any of the control keys, **Ctrl**, **Alt** or **Shift**).

To de-assign (or delete a shortcut):

1. Click the shortcut you want to delete in the Current Keys list box, and
2. Click the **Remove** button (which has now become active).
3. Click the Close button to confirm all the changes made in the Customize dialog box.

Menu

The Menu tab allows you to customize the main menu bars as well as the context menus.

You can customize both the Default and **UModel Project** menu bars.

The **Default** menu is the one visible when no XML documents of any type are open.

The **UModel Project** menu is the menu bar visible when a *.ump file has been opened.

To customize a menu:

1. Select the menu bar you want to customize from the "Show Menus for:" combo box
2. Click the Commands tab, and drag the commands to the menu bar of your choice.

To delete commands from a menu:

1. Click right on the command, or icon representing the command, and
 2. Select the **Delete** option from the pop-up menu,
- or,
1. Select **Tools | Customize** to open the Customize dialog box, and
 2. Drag the command away from the menu, and drop it as soon as the check mark icon appears below the mouse pointer.

To reset either of the menu bars:

1. Select either the Default or UModel Project entry in the combo box, and
2. Click the **Reset** button just below the menu name.
A prompt appears asking if you are sure you want to reset the menu bar.

To customize any of the Context menus (right click menus):

1. Select the context menu from the "Select context menus" combo box.
2. Click the [Commands](#) tab, and drag the specific commands to context menu that is now open.

To delete commands from a context menu:

1. Click right on the command, or icon representing the command, and
 2. Select the **Delete** option from the pop-up menu
- or,
1. Select **Tools | Customize** to open the Customize dialog box, and
 2. Drag the command away from the context menu, and drop it as soon as the check mark icon appears below the mouse pointer.

To reset any of the context menus:

1. Select the context menu from the combo box, and
 2. Click the **Reset** button just below the context menu name.
- A prompt appears asking if you are sure you want to reset the context menu.

To close an context menu window:

1. Click on the **Close icon** at the top right of the title bar, or
2. Click the Close button of the Customize dialog box.

Menu shadows

- Click the Menu shadows check box, if you want all your menus to have shadows.

Macros

The Macros tab allows you to select from the macros defined in the Scripting Project that is currently active in UModel.

The active Scripting Projects are specified in the Scripting tab of the Options dialog, or in the [Scripting tab](#) of the project settings.

Plug-Ins

The Plug-Ins tab allows you to select a UModel Plug-in file (.dll) to the list of currently active Plug-Ins.

See also [Testing the UModel plug-in](#).

Options

The **Options** tab allows you to set general environment settings.

When active, the **Show ScreenTips on toolbars** check box displays a tooltip label when the mouse pointer is placed over a toolbar button. The label contains a short description of the button function. If the **Show shortcut keys in ScreenTips** check box is selected, the tooltip label displays the associated keyboard shortcut, if one has been assigned.

When active, the **Large Icons** check box switches between the standard size icons, and larger versions of the icons.

18.6.7 Restore Toolbars and Windows

The **Restore Toolbars and Windows** command closes down UModel and re-starts it with the default settings. Before it closes down a dialog pops up asking for confirmation about whether UModel should be restarted.

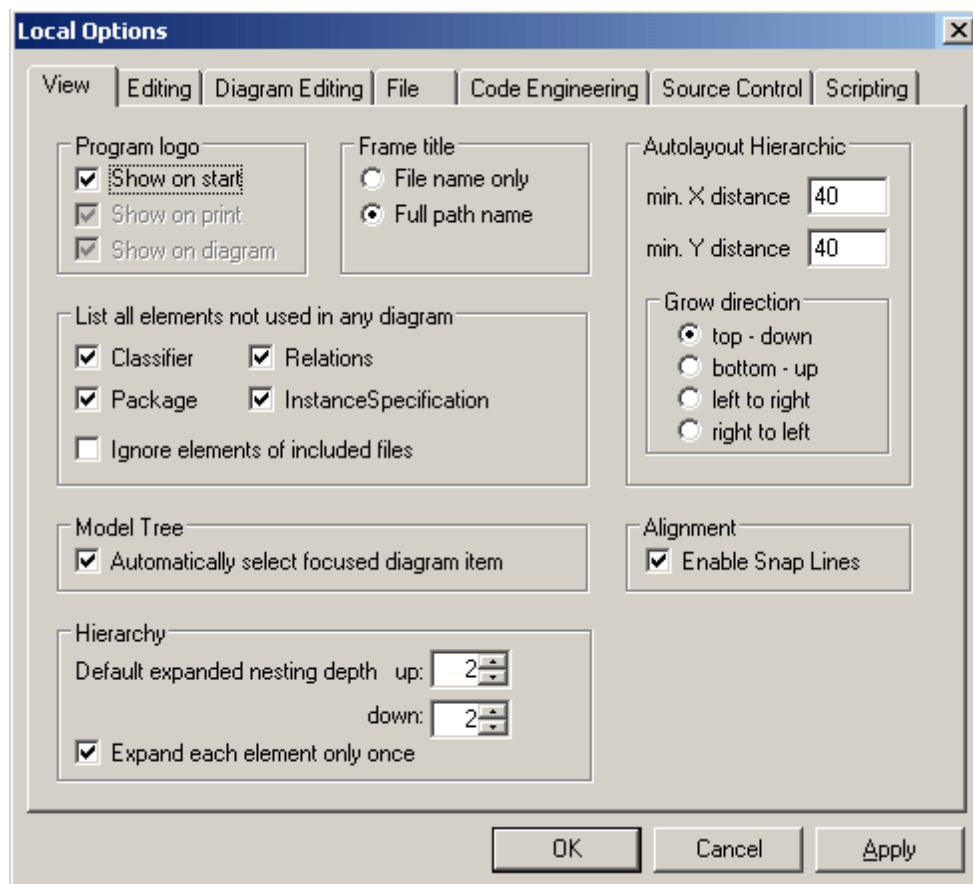
This command is useful if you have been resizing, moving, or hiding toolbars or windows, and would now like to have all the toolbars and windows as they originally were.

18.6.8 Options

Select the menu item **Tools | Options** to define your project options.

The **View** tab allows you to define:

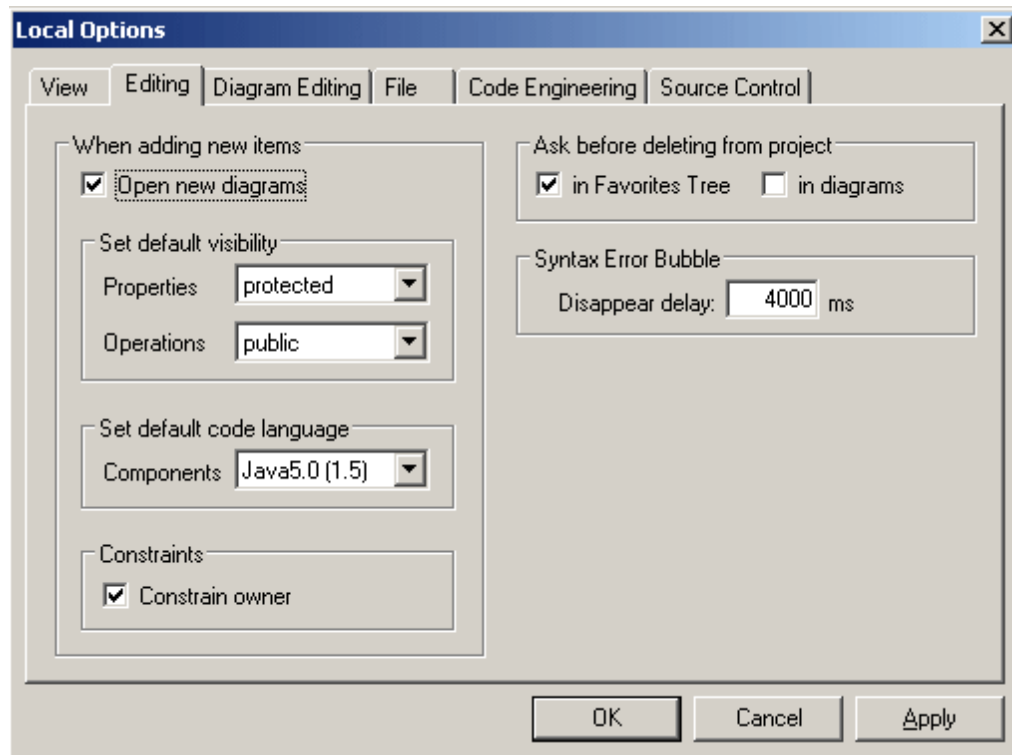
- Where the program logo should appear.
- The application title bar contents.
- The types of elements you want listed when using the "List elements not used in any diagram" context menu option in the Model Tree, or Favorites tab. You also have the option of ignoring elements contained in **included** files.
- If a selected element in a diagram is automatically selected/synchronized in the Model Tree.
- The default depth of the hierarchy view when using the **Show graph view** in the Hierarchy tab.
- The Autolayout Hierarchic settings, which allow you to define the nesting depth up and down in the hierarchy window.
- "Expand each element only once", only allows one of the same classifiers to be expanded in the same image/diagram.
- If you want snap lines to help you align elements when dragging in a diagram.



The **Editing** tab allows you to define:

- If a new Diagram created in the Model Tree tab, is also automatically opened in the main

- area.
- Default visibility settings when adding new elements - Properties or Operations.
- The default code language when a new component is added.
- If a newly added constraint, is to automatically constrain its owner as well.
- If a prompt should appear when **deleting** elements from a project, from the Favorites tab or in any of the diagrams. This prompt can be deactivated when deleting items there; this option allows you to reset the "prompt on delete" dialog box.
- The delay with which the syntax error pop-up message should be closed.

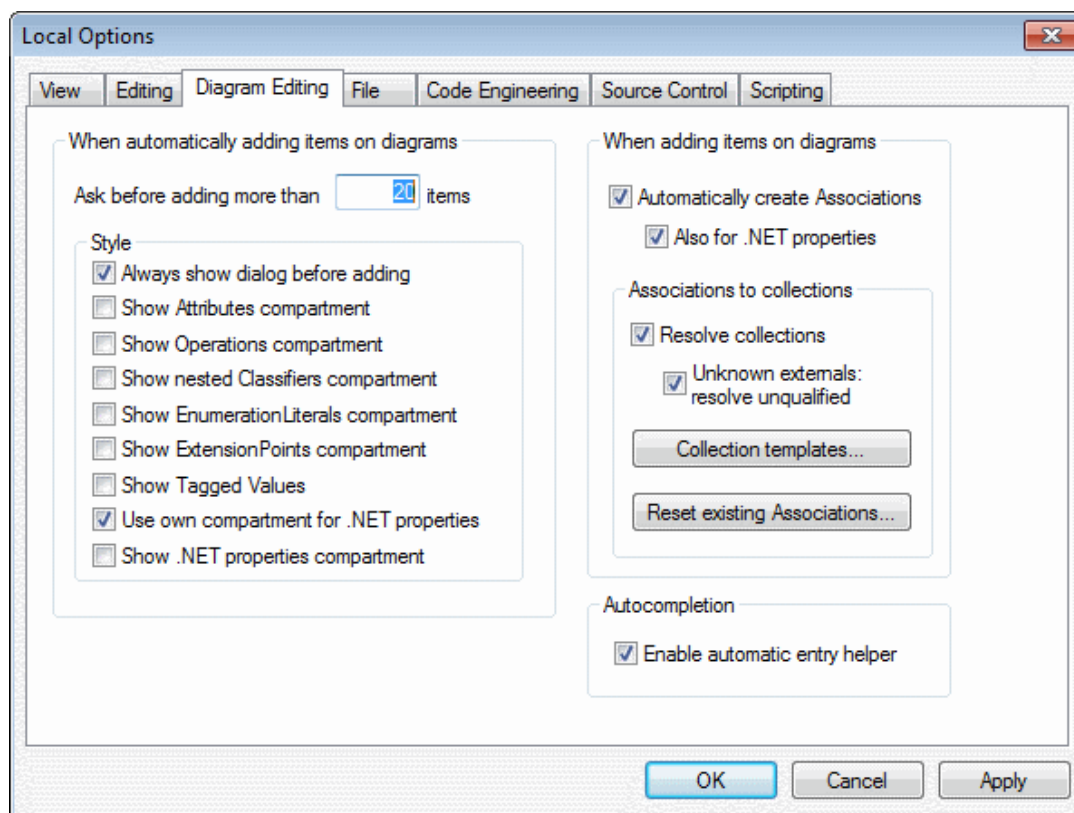


The **Diagram Editing** tab allows you to define:

- The number of items that can be automatically added to a diagram, before a prompt appears.
- The display of Styles when they are automatically added to a diagram.
- If Associations between modeling elements, are to be created automatically when items are added to a diagram.
- If the associations to collections are to be resolved.
- If templates from unknown externals are to be resolved as not fully qualified.
- or use preexisting Collection Templates, or define new ones.

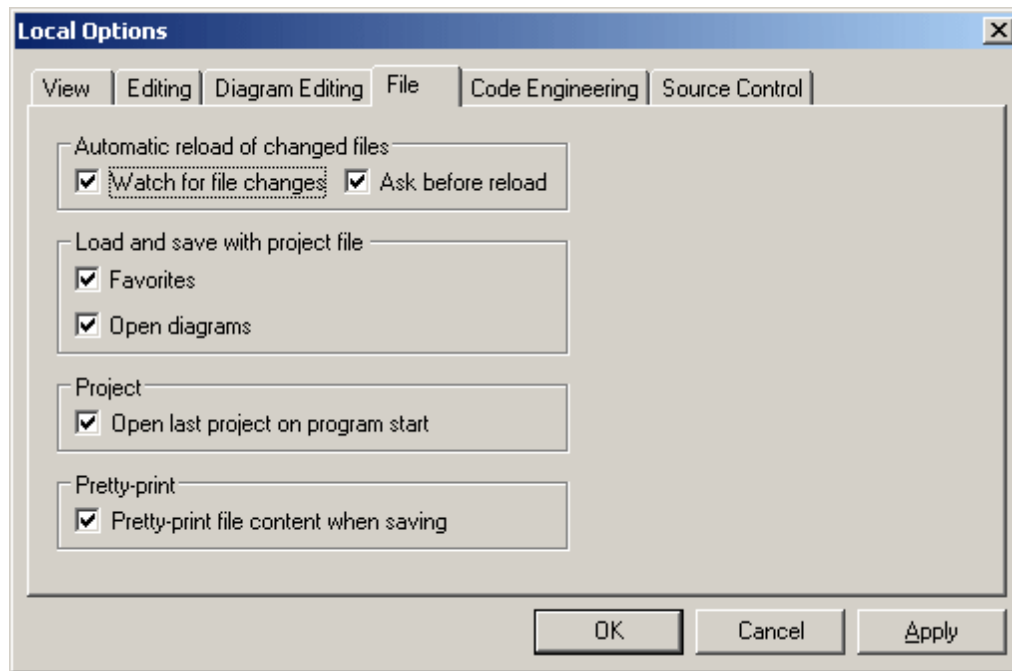
Collection Templates should be defined as fully qualified i.e. a.b.c.List. If the template has this namespace then UModel automatically creates a Collection Association. Exception: If the template belongs to the Unknown Externals package, and the option "Unknown externals: resolve unqualified", is enabled, then only the template name is considered (i.e. List instead of a.b.c.List).

- If the autocompletion window is to be available when editing attributes or operations in the class diagram.



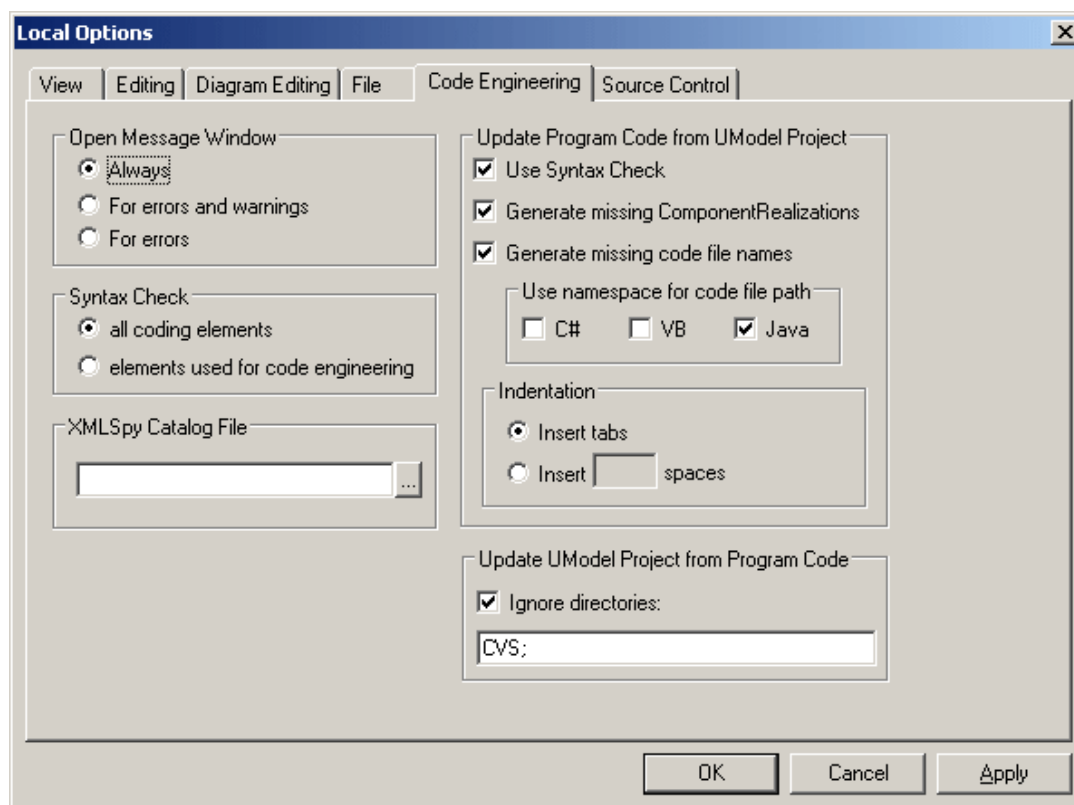
The **File** tab allows you to define:

- The actions performed when files are changed.
- If the contents of the Favorites tab are to be loaded and saved with the current project, as well as the any currently open diagrams.
- If the previously opened project is to automatically be opened when starting the application.
- If you want to structure the project file with CR/LF and tab indents in a **pretty-print** format.



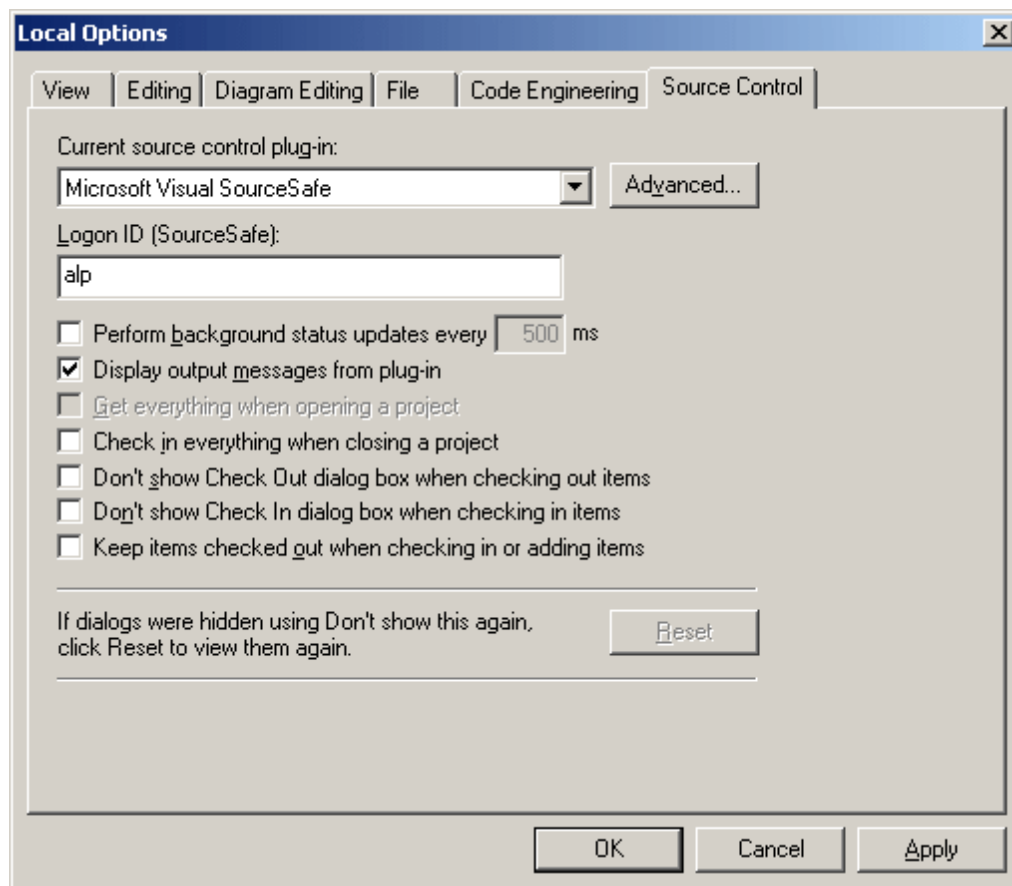
The **Code Engineering** tab allows you to define:

- The circumstances under which the Message window will open.
- If **all coding elements** i.e. those contained in a Java / C# / VB namespace root, as well as those assigned to a Java / C# / VB component, are to be checked, or only **elements used for code engineering**, i.e. where "use for code engineering" check box is active, are to be checked.
- When updating program code if:
 - If a syntax check is to be performed.
 - If missing ComponentRealizations are to be automatically generated.
 - If missing code file names in the merged code are to be generated.
 - If namespaces are to be used in the code file path.
- The Indentation method used in the code, i.e. tabs or any number of spaces.
- The directories to be ignored when updating a UModel project from code, or directory. Separate the respective directories with a semicolon ";". Child directories of the same name are also ignored.
- The location of the XMLSpy Catalog File, **RootCatalog.xml**, which enables UModel as well as XMLSpy to retrieve commonly used schemas (as well as stylesheets and other files) from local user folders. This increases the overall processing speed, and enables users to work offline.



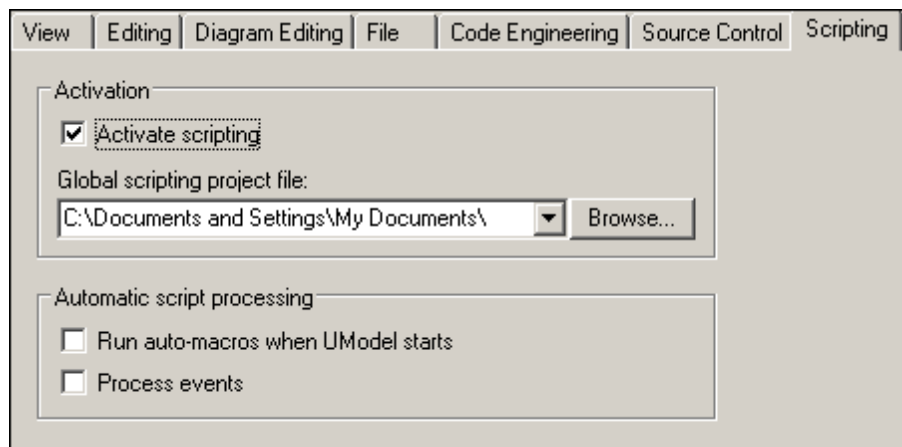
The **Source Control** tab allows you to define:

- The current source control plug-in using the combo box. The Advanced button allows you to define the specific settings of the source control plug-in that you selected. These settings change depending on the source control plug-in that you use.
- The login ID for the source control provider.
- Specific settings check in/out settings.
- The Reset button is made available if you have checked/activated the "Don't show this again" option in one of the dialog boxes. The Don't show this again prompt is then reenabled.



The **Scripting** tab allows you to define:

- If the [Scripting environment](#) should be active for the current UModel project.
- Which Global scripting file you want to use
- If auto-macros are to be executed when UModel starts
- If Scripting events are to be processed.



18.7 Window

Cascade:

This command rearranges all open document windows so that they are all cascaded (i.e. staggered) on top of each other.

Tile horizontally:

This command rearranges all open document windows as **horizontal tiles**, making them all visible at the same time.

Tile vertically:

This command rearranges all open document windows as **vertical tiles**, making them all visible at the same time.

Arrange icons:

Arranges haphazardly positioned, iconized diagrams, along the base of the diagram viewing area.

Close:

Closes the currently active diagram tab.

Close All:

Closes all currently open diagram tabs.

Close All but Active:

Closes all diagram tabs except for the currently active one.

Next:

Switches to the next modeling diagram in the tab sequence, or the next hyperlinked element.

Previous:

Switches to the previous modeling diagram in the tab sequence, or the previous hyperlinked element.

Window list:

This list shows all currently open windows, and lets you quickly switch between them.

You can also use the **Ctrl+Tab** or **Ctrl F6** keyboard shortcuts to cycle through the open windows.

18.8 Help

Allows access to the Table of Contents and Index of the UModel documentation, as well as Altova web site links. The Registration option opens the Altova Licensing Manager, which contains the licensing information for all of Altova products.

Chapter 19

Code Generator

19 Code Generator

UModel includes a built-in code generator which can automatically generate Java, C#, Visual Basic, or XML Schema files from UML models.

Note on **alias** usage:

When reverse engineering code which contains namespace or class aliases, you can now choose if the alias is to be resolved. (Check box "Resolve aliases" in Import Source Project/Directory dialog box).

When updating code, i.e. model to code, alias definitions are retained in the code as is. The scope of the alias definitions are the files in which they appear.

e.g.

```
using Q = System.Collections.Generic.Queue<String>;  
Q myQueue;
```

Any potentially conflicting aliases are added to the "Unknown Externals" if their use is unclear.

Note:

The Resolve alias settings can be changed at any time in the Project Settings dialog box.

19.1 The way to SPL (Spy Programming Language)

This section gives an overview of Spy Programming Language, the code generator's template language.

It is assumed that you have prior programming experience, and are familiar with operators, functions, variables and classes, as well as the basics of object-oriented programming - which is used heavily in SPL.

The templates used by UModel are supplied in the ...\\UModelspl folder. You can use these files as an aid to help you in developing your own templates.

How code generator works

Inputs to the code generator are the template files (.spl) and the object model provided by UModel. The template files contain SPL instructions for creating files, reading information from the object model and performing calculations, interspersed with literal code fragments in the target programming language.

The template file is interpreted by the code generator and outputs **.java**, **.cs** source code files, , or any other type of file depending on the template.

19.1.1 Basic SPL structure

An SPL file contains literal text to output, interspersed with code generator instructions.

Code generator instructions are enclosed in square brackets '[' and ']'.
Multiple statements can be included in a bracket pair. Additional statements have to be separated by a new line or a colon ':'.
Valid examples are:

```
[$x = 42  
$x = $x + 1]
```

or

```
[$x = 42: $x = $x + 1]
```

Adding text to files

Text not enclosed by '[' and ']', is written directly to the current output file.

To output literal square brackets, escape them with a backslash: '\[' and '\]'; to output a backslash use '\\.

Comments

Comments inside an instruction block always begin with a ' character, and terminate on the next line, or at a block close character ']'.

19.1.2 Variables

Any non-trivial SPL file will require variables. Some variables are predefined by the code generator, and new variables may be created simply by assigning values to them.

The **\$** character is used when **declaring** or **using** a variable, a variable name is always prefixed by **\$**.

Variable names are **case sensitive**.

Variables types:

- integer - also used as boolean, where 0 is false and everything else is true
- string
- object - provided by UModel
- iterator - see [foreach](#) statement

Variable types are declared by first assignment:

```
[$x = 0]
```

x is now an integer.

```
[$x = "teststring"]
```

x is now treated as a string.

Strings

String constants are always enclosed in double quotes, like in the example above. `\n` and `\t` inside double quotes are interpreted as newline and tab, `\` is a literal double quote, and `\\` is a backslash. String constants can also span multiple lines.

String concatenation uses the **&** character:

```
[$BasePath = $outputpath & "/" & $JavaPackageDir]
```

Objects

Objects represent the information contained in the UModelproject. Objects have **properties**, which can be accessed using the **.** operator. It is not possible to create new objects in SPL (they are predefined by the code generator, derived from the input), but it is possible to assign objects to variables.

Example:

```
class [= $class.Name]
```

This example outputs the word "class", followed by a space and the value of the **Name** property of the **\$class** object.

The following table show the relationship between UML elements their SPL equivalents along with a short description.

Predefined variables

UML element	SPL property	Multiplicity	UML	UModel	Description

			Attribute / Association	Attribute / Association	
BehavioralFeature	isAbstract		isAbstract:Boolean		
BehavioralFeature	raisedException	*	raisedException:Type		
BehavioralFeature	ownedParameter	*	ownedParameter:Parameter		
BehavioredClassifier	interfaceRealization	*	interfaceRealization:InterfaceRealization		
Class	ownedOperation	*	ownedOperation:Operation		
Class	nestedClassifier	*	nestedClassifier:Classifier		
Classifier	namespace	*		namespace:Package	packages with code language <<namespace>> set
Classifier	rootNamespace	*		project root namespace:String	VB only - root namespace
Classifier	generalization	*	generalization:Generalization		
Classifier	isAbstract		isAbstract:Boolean		
ClassifierTemplateParameter	constrainingClassifier	*	constrainingClassifier		
Comment	body		body:String		
DataType	ownedAttribute	*	ownedAttribute:Property		
DataType	ownedOperation	*	ownedOperation:Operation		
Element	kind			kind:String	
Element	owner	0..1	owner:Element		
Element	appliedStereotype	*		appliedStereotype:StereotypeApplication	applied stereotypes
Element	ownedComment	*	ownedComment:Comment		
ElementImport	importedElement	1	importedElement:PackageableElement		
Enumeration	ownedLiteral	*	ownedLiteral:EnumerationLiteral		

Enumeration	nestedClassifier	*		nestedClassifier::Classifier	
Enumeration	interfaceRealization	*		interfaceRealization:Interface	
EnumerationLiteral	ownedAttribute	*		ownedAttribute:Property	
EnumerationLiteral	ownedOperation	*		ownedOperation:Operation	
EnumerationLiteral	nestedClassifier	*		nestedClassifier:Classifier	
Feature	isStatic		isStatic:Boolean		
Generalization	general	1	general:Classifier		
Interface	ownedAttribute	*	ownedAttribute:Property		
Interface	ownedOperation	*	ownedOperation:Operation		
Interface	nestedClassifier	*	nestedClassifier:Classifier		
InterfaceRealization	contract	1	contract:Interface		
MultiplicityElement	lowerValue	0..1	lowerValue:ValueSpecification		
MultiplicityElement	upperValue	0..1	upperValue:ValueSpecification		
NamedElement	name		name:String		
NamedElement	visibility		visibility:VisibilityKind		
NamedElement	isPublic			isPublic:Boolean	visibility <public>
NamedElement	isProtected			isProtected:Boolean	visibility <protected>
NamedElement	isPrivate			isPrivate:Boolean	visibility <private>
NamedElement	isPackage			isPackage:Boolean	visibility <package>
NamedElement	namespacePrefix			namespacePrefix:String	XSD only - namespace prefix when exists
NamedElement	parseableName			parseableName:String	CSharp, VB only - name with escaped keywords (@)

Namespace	elementImport	*	elementImport:ElementImport		
Operation	ownedReturnParameter	0..1		ownedReturnParameter:Parameter	parameter with direction return set
Operation	type	0..1		type	type of parameter with direction return set
Operation	ownedOperationParameter	*		ownedOperationParameter:Parameter	all parameters excluding parameter with direction return set
Operation	implementedInterface	1		implementedInterface:Interface	CSharp only - the implemented interface
Operation	ownedOperationImplementations	*		implementedOperation:OperationImplementation	VB only - the implemented interfaces/operations
OperationImplementation	implementedOperationOwner	1		implementedOperationOwner:Interface	interface implemented by the operation
OperationImplementation	implementedOperationName			name:String	name of the implemented operation
OperationImplementation	implementedOperationParseableName			parseableName:String	name of the implemented operation with escaped keywords
Package	namespace	*		namespace:Package	packages with code language <<namespace>> set
PackageableElement	owningPackage	0..1		owningPackage	set if owner is a package
PackageableElement	owningNamespacePackage	0..1		owningNamespacePackage:Package	owning package with code language <<namespace>> set
Parameter	direction		direction:ParameterDirectionKind		
Parameter	isIn			isIn:Boolean	direction <in>
Parameter	isInOut			isInOut:Boolean	direction <inout>
Parameter	isOut			isOut:Boolean	direction <out>
Parameter	isReturn			isReturn:Boolean	direction <return>
Parameter	isVarArgList			isVarArgList:Boolean	true if parameter is a variable argument list

Parameter	defaultValue	0..1	defaultValue:ValueSpecification		
Property	defaultValue	0..1	defaultValue:ValueSpecification		
RedefinableElement	isLeaf		isLeaf:Boolean		
Slot	name			name:String	name of the defining feature
Slot	values	*	value:ValueSpecification		
Slot	value			value:String	value of the first value specification
StereotypeApplication	name			name:String	name of applied stereotype
StereotypeApplication	taggedValue	*		taggedValue:Slot	first slot of the instance specification
StructuralFeature	isReadOnly		isReadOnly		
StructuredClassifier	ownedAttribute	*	ownedAttribute:Property		
TemplateBinding	signature	1	signature:TemplateSignature		
TemplateBinding	parameterSubstitution	*	parameterSubstitution:TemplateParameterSubstitution		
TemplateParameter	paramDefault			paramDefault:String	template parameter default value
TemplateParameter	ownedParameteredElement	1	ownedParameteredElement:ParameterableElement		
TemplateParameterSubstitution	parameterSubstitution			parameterSubstitution:String	Java only - code wildcard handling
TemplateParameterSubstitution	parameterDimensionCount			parameterDimensionCount:Integer	code dimension count of the actual parameter
TemplateParameterSubstitution	actual	1	Ow nedActual:ParameterableElement		
TemplateParameterSubstitution	formal	1	formal:TemplateParameter		
TemplateSignature	template	1	template:TemplateableElement		
TemplateSignature	ownedParameter	*	ow nedParameter:TemplateParameter		

TemplateableElement	isTemplate			isTemplate:Boolean	true if template signature set
TemplateableElement	ownedTemplateSignature	0..1	ownedTemplateSignature:TemplateSignature		
TemplateableElement	templateBinding	*	templateBinding:TemplateBinding		
Type	typeName	*		typeName:PackageableElement	qualified code type names
TypedElement	type	0..1	type:Type		
TypedElement	postTypeModifier			postTypeModifier:String	postfix code modifiers
ValueSpecification	value			value:String	string value of the value specification

Adding a prefix to attributes of a class during code generation

You might need to prefix all new attributes with the "m_" characters in your project.

All new coding elements are written using the SPL templates:

If you look into UModelSPL\C#[Java]\Default\Attribute.spl, you can change the way how the name is written, e.g. replace

```
write $Property.name
```

by

```
write "m_" & $Property.name
```

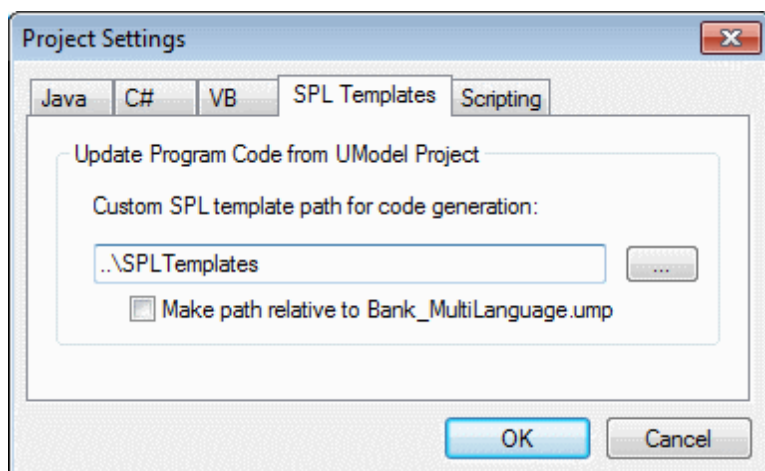
It is highly recommended that you immediately update your model from code, after code generation to ensure that code and model are synchronized!

Please note:

As previously mentioned copy the SPL templates one directory higher (i.e. above the **default** directory to UModelSPL\C#) before modifying them. This ensures that they are not overwritten when you install a new version of UModel. Please make sure that the "user-defined override default" check box is activated in the **Code from Model** tab of the Synchronization Setting dialog box.

SPL Templates

SPL templates can be specified per UModel project using the menu option **Project | Project Settings** (as shown in the screenshot below). Relative paths are also supported. Templates which are not found in the specified directory, are searched for in the local default directory.



Global objects

\$Options	an object holding global options:
	generateComments:bool generate doc comments (true/false)
\$Indent	a string used to indent generated code and represent the current nesting level
\$IndentStep	a string, used to indent generated code and represent one nesting level
\$NamespacePrefix	XSD only – the target namespace prefix if present

String manipulation routines

integer **Compare**(s)

The return value indicates the lexicographic relation of the string to s (case sensitive):

<0:	the string is less than s
0:	the string is identical to s
>0:	the string is greater than s

integer **CompareNoCase**(s)

The return value indicates the lexicographic relation of the string to s (case insensitive):

<0:	the string is less than s
0:	the string is identical to s
>0:	the string is greater than s

integer **Find**(s)

Searches the string for the first match of a substring s.

Returns the zero-based index of the first character of s or -1 if s is not found.

string **Left**(n)

Returns the first n characters of the string.

integer **Length**()

Returns the length of the string.

string **MakeUpper**()

Returns a string converted to upper case.

string **MakeUpper**(n)

Returns a string, with the first n characters converted to upper case.

string **MakeLower**()

Returns a string converted to lower case.

string **MakeLower**(n)

Returns a string, with the first n characters converted to lower case.

string **Mid**(n)

Returns a string starting with the zero-based index position n

string **Mid**(n, m)

Returns a string starting with the zero-based index position n and the length m

string **RemoveLeft**(s)

Returns a string excluding the substring s if Left(s.Length()) is equal to substring s.

string **RemoveLeftNoCase**(s)

Returns a string excluding the substring s if Left(s.Length()) is equal to substring s (case insensitive).

string **RemoveRight**(s)

Returns a string excluding the substring s if Right(s.Length()) is equal to substring s.

string **RemoveRightNoCase**(s)

Returns a string excluding the substring s if Right(s.Length()) is equal to substring s (case insensitive).

string **Repeat**(s, n)

Returns a string containing substring s repeated n times.

string **Right**(n)

Returns the last n characters of the string.

19.1.3 Operators

Operators in SPL work like in most other programming languages.

List of SPL operators in descending precedence order:

.	Access object property
()	Expression grouping
true	boolean constant "true"
false	boolean constant "false"
&	String concatenation
-	Sign for negative number
not	Logical negation
*	Multiply
/	Divide
%	Modulo
+	Add
-	Subtract
<=	Less than or equal
<	Less than
>=	Greater than or equal
>	Greater than
=	Equal
<>	Not equal
and	Logical conjunction (with short circuit evaluation)
or	Logical disjunction (with short circuit evaluation)
=	Assignment

19.1.4 Conditions

SPL allows you to use standard "if" statements. The syntax is as follows:

```
if condition
    statements
else
    statements
endif
```

or, without else:

```
if condition
    statements
endif
```

Please note that there are no round brackets enclosing the condition!

As in any other programming language, conditions are constructed with logical and comparison [operators](#).

Example:

```
[if $namespace.ContainsPublicClasses and $namespace.Prefix <> ""]
    whatever you want ['inserts whatever you want, in the resulting file]
[endif]
```

Switch

SPL also contains a multiple choice statement.

Syntax:

```
switch $variable
    case X:
        statements
    case Y:
    case Z:
        statements
    default:
        statements
endswitch
```

The case labels must be constants or variables.

The switch statement in SPL does not fall through the cases (as in C), so there is no need for a "break" statement.

19.1.5 Collections and foreach

Collections and iterators

A collection contains multiple objects - like a ordinary array. Iterators solve the problem of storing and incrementing array indexes when accessing objects.

Syntax:

```
foreach iterator in collection
    statements
next
```

Example:

```
[foreach $class in $classes
    if not $class.IsInternal
        ]      class [=$class.Name];
    endif
next]
```

Example 2:

```
[foreach $i in 1 To 3
    Write "// Step " & $i & "\n"
    ` Do some work
next]
```

Foreach steps through all the items in \$classes, and executes the code following the instruction, up to the **next** statement, for each of them.

In each iteration, **\$class** is assigned to the next class object. You simply work with the class object instead of using, classes[i]->Name(), as you would in C++.

All collection iterators have the following additional properties:

Index	The current index, starting with 0
IsFirst	true if the current object is the first of the collection (index is 0)
IsLast	true if the current object is the last of the collection

Example:

```
[foreach $enum in $facet.Enumeration
    if not $enum.IsFirst
        ], [
    endif
    ]" [=$enum.Value]" [
next]
```

Collection manipulation routines:

collection **SortByName**(bAscending)

returns a collection whose elements are sorted by name (case sensitive) in ascending or descending order.

collection **SortByNameNoCase**(bAscending)
returns a collection whose elements are sorted by name (case insensitive) in ascending or descending order

Example:

```
$SortedNestedClassifier = $Class.nestedClassifier.SortByNameNoCase( true )
```

collection **SortByKind**(bAscending)
returns a collection whose elements are sorted by kind names (e.g. "Class", "Interface",...) in ascending or descending order.

collection **SortByKindAndName**(bAscendingKind, bAscendingName)
returns a collection whose elements are sorted by kind (e.g. "Class", "Interface",...) in ascending or descending order and if the kinds are equal by name (case sensitive in ascending or descending order)

collection **SortByKindAndNameNoCase**(bAscending)
returns a collection whose elements are sorted by kind (e.g. "Class", "Interface",...) in ascending or descending order and if the kinds are equal by name (case insensitive in ascending or descending order)

19.1.6 Subroutines

Code generator supports subroutines in the form of procedures or functions.

Features:

- By-value and by-reference passing of values
- Local/global parameters (local within subroutines)
- Local variables
- Recursive invocation (subroutines may call themselves)

Subroutine declaration

Subroutines

Syntax example:

```
Sub SimpleSub()  
... lines of code  
EndSub
```

- **Sub** is the keyword that denotes the procedure.
- **SimpleSub** is the name assigned to the subroutine.
- Round **parenthesis** can contain a parameter list.
- The code block of a subroutine starts immediately after the closing parameter parenthesis.
- **EndSub** denotes the end of the code block.

Please note:

Recursive or cascaded subroutine **declaration** is not permitted, i.e. a subroutine may not contain another subroutine.

Parameters

Parameters can also be passed by procedures using the following syntax:

- All parameters must be variables
- Variables must be prefixed by the **\$** character
- Local variables are defined in a subroutine
- Global variables are declared explicitly, outside of subroutines
- Multiple parameters are separated by the comma character **,** within round parentheses
- Parameters can pass values

Parameters - passing values

Parameters can be passed in two ways, by value and by reference, using the keywords **ByVal** and **ByRef** respectively.

Syntax:

```
' define sub CompleteSub()  
[Sub CompleteSub( $param, ByVal $paramByValue, ByRef $paramByRef )  
] ...
```

- **ByVal** specifies that the parameter is passed by value. Note that most objects can only be passed by reference.

- **ByRef** specifies that the parameter is passed by reference. This is the default if neither **ByVal** nor **ByRef** is specified.

Function return values

To return a value from a subroutine, use the **return** statement. Such a function can be called from within an expression.

Example:

```
' define a function
[Sub MakeQualifiedName( ByVal $namespacePrefix, ByVal $localName )
if $namespacePrefix = ""
    return $localName
else
    return $namespacePrefix & ":" & $localName
endif
EndSub
]
```

Subroutine invocation

Use **call** to invoke a subroutine, followed by the procedure name and parameters, if any.

```
Call SimpleSub()
```

or,

```
Call CompleteSub( "FirstParameter", $ParamByValue, $ParamByRef )
```

Function invocation

To invoke a function (any subroutine that contains a **return** statement), simply use its name inside an expression. Do not use the **call** statement to call functions.

Example:

```
$QName = MakeQualifiedName($namespace, "entry")
```

19.2 Error Codes

Operating System Error Codes

- 201 File not found: '%s'
- 202 Cannot create file '%s'
- 203 Cannot open file '%s'
- 204 Cannot copy file '%s' to '%s'

Syntax Error Codes

- 401 Keyword expected
- 402 '%s' expected
- 403 No output file specified
- 404 Unexpected end of file
- 405 Keyword not allowed

Runtime Error Codes

- 501 Unknown variable '%s'
- 502 Redefinition of variable '%s'
- 503 Variable '%s' is not a container
- 504 Unknown property '%s'
- 505 Cannot convert from %s to %s
- 507 Unknown function
- 508 Function already defined
- 509 Invalid parameter
- 510 Division by zero
- 511 Unknown method
- 512 Incorrect number of parameters
- 513 Stack overflow

Chapter 20

UModel Programmer's Reference

20 UModel Programmer's Reference

UModel is an Automation Server. That is, it is an application that exposes programmable objects to other applications (called Automation Clients). As a result, an Automation Client can directly access the objects and functionality that the Automation Server makes available. This is beneficial to an Automation Client because it can make use of the functionality of UModel. For example, an Automation Client can use the reverse engineering functionality of UModel. Developers can therefore improve their applications by using the ready-made functionality of UModel.

The programmable objects of UModel are made available to Automation Clients via the **UModel API**, which is a COM API. The object model of the API and a complete description of all available objects are provided in this documentation (see [UModel API Reference](#)).

The Application API can be accessed from within the following environments:

- [Scripting Editor](#)
- [IDE Plug-ins](#)
- [External programs](#)

Each of these environments is described briefly below.

Scripting Editor: Customizing and modifying UModel functionality

You can customize your installation of UModel by modifying and adding functionality to it. You can also create Forms for user input and modify the user interface so that it contains new menu commands and toolbar shortcuts. All these features are achieved by writing scripts that interact with objects of the Application API. To aid you in carrying out these tasks efficiently, UModel offers you an in-built Scripting Editor. A complete description of the functionality available in the Scripting Editor and how it is to be used is given in the [Scripting Editor](#) section of this documentation. The supported programming languages are **JScript** and **VBScript**.

IDE Plug-ins: Creating plug-ins for UModel

UModel enables you to create your own plug-ins and integrate them into UModel. You can do this using UModel's special interface for plug-ins. A description of how to create plug-ins is given in the section [IDE Plug-ins](#) .

An application object gets passed to most methods that must be implemented by an IDE plug-in and gets called by the application. Typical languages used to implement an IDE plug-in are **C#** and **C++**. For more information, see the section [IDE Plug-ins](#) .

External programs

Additionally, you can manipulate UModel with external scripts. For example, you could write a script to open UModel at a given time, then open a UModel project generate UML documentation , and print it out. External scripts would again make use of the Application API to carry out these tasks. For a description of the Application API, see [The UModel API](#).

Using the Application API from outside UModel requires an instance of UModel to be started first. How this is done depends on the programming language used. See [Programming Languages](#) for information about individual languages.

Essentially, UModel will be started via its COM registration. Then the `Application` object associated with the UModel instance is returned. Depending on the COM settings, an object associated with an already running UModel can be returned. Any programming language that supports creation and invocation of COM objects can be used. The most common of these are listed below.

- [JScript](#) and VBScript script files have a simple syntax and are designed to access COM objects. They can be run directly from a DOS command line or with a double click on Windows Explorer. They are best used for simple automation tasks.
- [C#](#) is a full-fledged programming language that has a wide range of existing functionality. Access to COM objects can be automatically wrapped using C#.
- [Java](#): Altova products come with native Java classes that wrap the Application API and provide a full Java look-and-feel.
- Other programming languages that make useful alternatives are: Visual Basic for Applications, Perl, and Python.

20.1 Release Notes

For each release of the UModel API, important changes since the previous release are listed below. A change in the major version of the type library (e.g. 1.0 => 2.0 => 3.0) means that non-scripting clients (e.g. IDE Plug-ins written in C#, VB.NET, C++,...) should be recompiled.

Automation Interface for UModel 2016 - type library version 5.2

UModel API - UMLData	The following methods have been added: • <code>IUMLInstanceSpecification::SetSlotInstanceValueAt</code> • <code>IUMLSlot::InsertSlotInstanceValueAt</code> • <code>IUMLDataAll::InsertSlotInstanceValueAt</code> • <code>IUMLDataAll::SetSlotInstanceValueAt</code>

Automation Interface for UModel 2015r4 - type library version 5.1

UModel API - UMLData	The following methods have been added: • <code>IUMLGuiNodeLink::AddOwnedGuiNodeLink</code> • <code>IUMLDataAll::AddOwnedGuiNodeLink</code>

Automation Interface for UModel 2013 - type library version 5.0

Changes since the previous release (UModel 2012 - type library version 4.1) are as follows:

UModel API	• IDocument has a new method <code>GenerateSequenceDiagramsForAllOperations</code>
UModel API - UMLData	• ENUMExportXMIType has a new entry for UML2.4 (eXMI24ForUML24). • ENUMUMLGuiTextLabelKind has a new literal <code>eTextLabel_DotNetPropertyName</code> • ENUMUMLPredefinedElement has new literals for SysML 1.2. • IUMLGuiSequenceDiagram has new properties <code>UseForForwardEngineering</code> and <code>CodeOperation</code> regarding code generation. • <code>IUMLExecutionEvent</code>, <code>IUMLCreationEvent</code>, <code>IUMLDestructionEvent</code>, <code>IUMLSendOperationEvent</code>, <code>IUMLSendSignalEvent</code>, <code>IUMLReceiveOperationEvent</code> and <code>IUMLReceiveSignalEvent</code> have been removed since the corresponding classes are not part of UML2.4 anymore.

Automation Interface for UModel 2012 - type library version 4.1

Changes since the previous release (UModel 2011r3 - type library version 4.0) are as follows:

- For model transformations, [IModelTransformationDlg](#), [IModelTransformationTypeMappings](#) and [IModelTransformationTypeMapping](#) have been introduced. [IDocument](#) has a new method **ModelTransformation**.
- [ILocalOptionsView](#) has a new property **EnableSnapLines**.

UModel API - UMLData

- [ENUMCodeLang](#) has a new literal `eCodeLang_UML`.
- [ENUMUMLPredefinedElement](#) has several new literals for model transformations.

Automation Interface for UModel 2011r3 - type library version 4.0

Changes since the previous release (UModel 2011r2 - type library version 3.2) are as follows:

UModel API

- [ILocalOptionsDiagramEditing](#) has new properties **UseDotNetPropertyCompartment** and **ShowDotNetPropertyCompartment** for the handling of .NET properties.
- [IDialog](#) has new properties **Application** and **Parent**
- [IImportSourceDlg](#) has got new properties **Content_UseDotNetPropertyCompartment** and **Content_ShowDotNetPropertyCompartment** for the handling of .NET properties.

UModel API - UMLData

- Property **BehaviorSpecification** can be set for [IUMLBehavior](#)
- [ENUMUMLGuiStyleKind](#) has a new literal `eUMLGuiStyle_ShowDotNetPropertyCompartment` for the handling of .NET properties

Automation Interface for UModel 2011r2 - type library version 3.2

Changes since the previous release (UModel 2011 - type library version 3.1) are as follows:

UModel API

- For state machine code generation, [IGenerateStateMachineCodeDlg](#) has been introduced and [IDocument](#) has got method **GenerateStateMachineCode**.
- [IGenerateDocumentationDlg](#) has new properties **UseFixedDesign** and **SPSFile** for SPS documentation generation; a new property **Include_IncludedPredefinedSubprojects** and a new method **Fonts_SetDefaults**.
- [ENUMDocumentationOutputFormat](#) has got a new literal for documentation generation in PDF format
- [ENUMUMLPredefinedElement](#) has several new literals for BPMN2 support

UModel API - UMLData

- [IUMLGuiBPMN2Diagram](#), [IUMLGuiBPMN2ChoreographyDiagram](#) and [IUMLGuiBPMN2CollaborationDiagram](#) have been introduced for BPMN2

Automation Interface for UModel 2011 - type library version 3.1

Changes since the previous release (UModel 2010r3 - type library version 3.0) are as follows:

UModel API

- [IImportDatabaseDlg](#) has been introduced for importing databases.
- [IDocument](#) has new methods **ImportDatabase** for importing databases and **MergeProject3Way** for 3-way project merges

UModel API - UMLData

- [ENUMCodeLangVersion](#) and [ENUMUMLPredefinedElement](#) have new literals for database support
- [ENUMUMLDBDataSourceMethod](#) has been introduced for database support

Automation Interface for UModel 2010r3 - type library version 3.0

Changes since the previous release (UModel 2010r2 - type library version 2.1) are as follows:

UModel API

- [IGenerateSequenceDiagramDlg](#) has property **OperationIgnoreList** to ignore distinct operations when generating a sequence diagram from source code.
- The [IGenerateDocumentationDlg](#) interface has the new property **Details_Constraints**.
- [ENUMDiagramLayoutKind](#) has a new entry for layout kind "Block"
- [ENUMCodeLangVersion](#) has a new entry for C# 4.0

UModel API - UMLData

- [IUMLElement](#) has method **GetOwnedElementsOfKind** to retrieve all owned elements of a specific kind.
- [IUMLCClass](#), [IUMLEnumeration](#) and [IUMLIInterface](#) have method **GetCodeFilePath** to get the full code file path (also see [GetCodeFileName](#) which returns the file name only).
- [IUMLIInterface](#) has property **Protocol** and method **SetNewProtocol** to reference a [IUMLProtocolStateMachine](#).
- [IUMLConstraint](#) has new properties **OwningTransition** and **OwningState**.
- [IUMLState](#) has property **StateInvariant** and method **SetNewStateInvariant**.
- [IUMLPort](#) has a new property **Protocol**
- [IUMLStructuredClassifier](#) has method **InsertOwnedPortAt**.
- [ENUMUMLPredefinedElement](#) has several new literals for C# 4.0
- Additionally following new interfaces have been introduced:
[IUMLValueSpecificationAction](#), [IUMLProtocolStateMachine](#), [IUMLProtocolTransition](#),
[IUMLGuiProtocolStateMachineDiagram](#)

Automation Interface for UModel 2010r2 - type library version 2.1

Changes since the previous release (UModel 2010 - type library version 2.0) are as follows:

UModel API

- [IGenerateSequenceDiagramDlg](#) has property **SplitIntoSmallerDiagrams** for the new sequence diagram split.

UModel API - UMLData

- [ENUMExportXMIMType](#) has a new entry for UML2.3.
- [IUMLAction](#) has property **IsLocallyReentrant**.
- [IUMLPort](#) has property **IsConjugated**.
- Property **ConnectorKind** of [IUMLConnector](#) is now read-only.
- [IUMLClassifier](#) has property **IsFinalSpecialization**.
- [IUMLActivityGroup](#) now derives from [IUMLNamedElement](#).

Automation Interface for UModel 2010 - type library version 2.0

Changes since the previous release (UModel 2009 - type library version 1.0) are as follows:

UModel Plug-ins

- Support for ActiveX controls. Any IDE Plug-in which is also an ActiveX control will be displayed in a Dialog Control Bar inside UModel (also see [ActiveX Controls](#) or the [StatisticsActiveX](#) sample)

UModel API

- [Application](#) has property **ServicePackVersion** and method **RunMacro** to start a macro of an (already loaded) scripting project.
- [IDocument](#) has the new methods: **SaveCopyAs**, **CanFocusUMLDataInModelTree**, **FocusUMLDataInModelTree**, **Reload**.
- Property **FocusedUMLDataNotifier** of [IDocument](#) can be used to get the new [IFocusedUMLDataEvents](#) interface for focus change events.
- Method **OnModifiedFlagChanged** of [IDocumentEvents](#) has the [IDocument](#) interface as second parameter.
- The [IDiagramWindow](#) interface has following methods for auto-layout: **Autolayout**, **AutolayoutSelection**.
- The [IProjectSettingsDlg](#) interface has the new properties **CSharp_ResolveAliases** and **VBasic_ResolveAliases**.
- The [IGenerateDocumentationDlg](#) interface has the new properties **EmbedCSSinHTML** and **CreateFolderForDiagrams**.
- [ILocalOptionsCodeEngineering](#) has following new properties:
CodeFromModel_Indentation_InsertTabs,
CodeFromModel_Indentation_InsertNSpaces.

UModel API - UMLData

- The new interface [IUMLHyperlink2Model](#) allows hyperlinks to model elements (in the Model Tree). [IUMLNamedElement](#) has got **InsertOwnedHyperlink2ModelAt**, [IUMLGuiTextHyperlink](#) has **SetHyperlinkModelElementAddress** to set links to model elements.
- [IUMLCommentTextHyperlink](#) is new and enables hyperlinks for [IUMLComments](#) (also see [How to create and use Hyperlinks](#)). [IUMLComment](#) is extended by **InsertOwnedCommentTextHyperlinkAt** and **OwnedHyperlinks** to insert and access these hyperlinks.
- In the same way [IUMLGuiDiagram](#) has been extended by **InsertOwnedGuiTextHyperlinkAt** and **OwnedHyperlinks**.
- The [IUMLElement](#) interface has the properties **OwnedDocComment** and **OwnedDocCommentBody** to directly access the body of the comment, which is shown in the documentation window when the element is focused.
- [IUMLGuiDiagram](#) has a method **AddUMLGuiContainmentLink** to insert containment lines on diagrams.
- [ENUMUMLGuiTextLabelKind](#) has the new literal **eTextLabel_InformationFlow**.
- [ENUMUMLPredefinedElement](#) has new literals for XSD datatypes and the SysML profile.
- Additionally, following new interfaces have been added: [IUMLInformationFlow](#), [IUMLGuiContainmentLink](#), [IUMLGuiSysMLActivityDiagram](#), [IUMLGuiSysMLBlockDefinitionDiagram](#), [IUMLGuiSysMLInternalBlockDiagram](#), [IUMLGuiSysMLPackageDiagram](#), [IUMLGuiSysMLParametricDiagram](#), [IUMLGuiSysMLRequirementDiagram](#), [IUMLGuiSysMLSequenceDiagram](#), [IUMLGuiSysMLStateMachineDiagram](#), [IUMLGuiSysMLUseCaseDiagram](#)

20.2 Scripting Editor

The Scripting Editor of UModel uses the Form Editor components of the Microsoft .NET Framework, and thus provides access to the Microsoft .NET Framework. This means that JScripts and VBScripts not only work with the UModel API—which is a COM API and the API of UModel—but can also access and use classes of the Microsoft .NET framework.

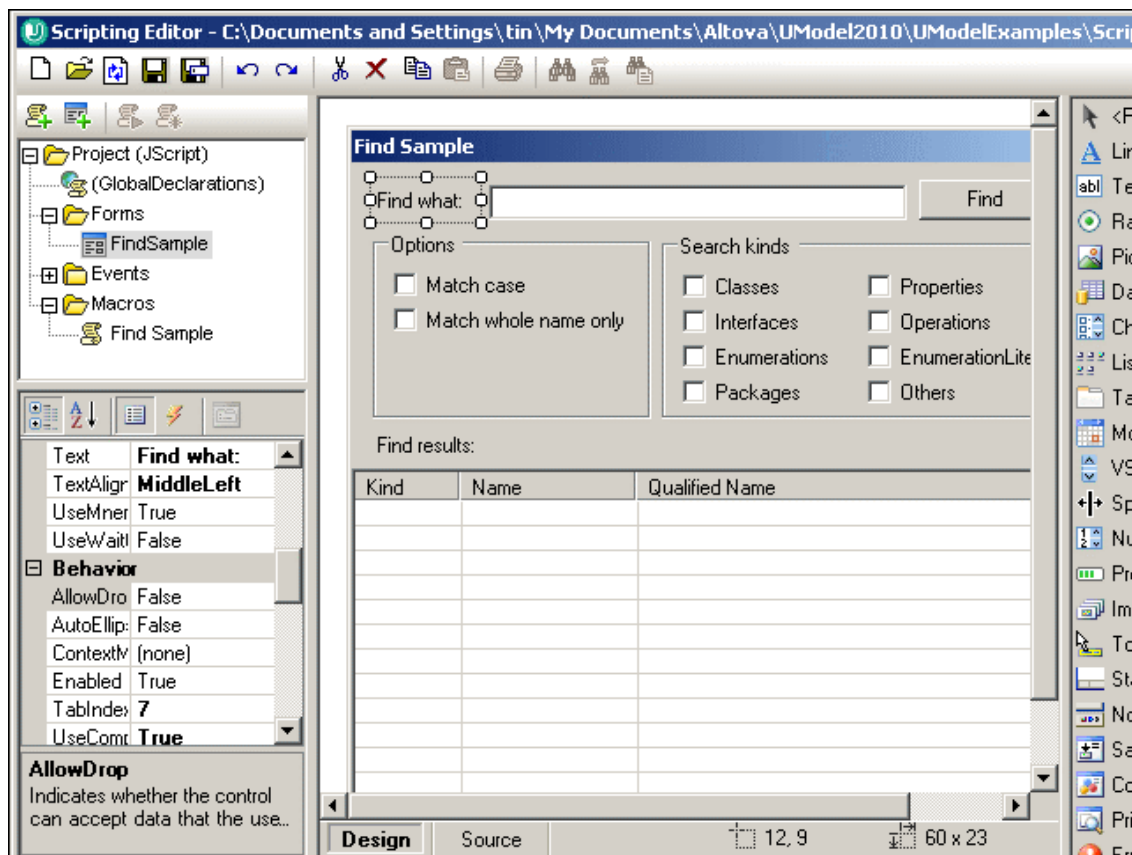
You can therefore create and use your own macros and forms within UModel, and thus add to and modify the functionality of your installation of UModel.

Note: Visual Basic is **not supported** as a language in the scripting environment. Only VBScript and JavaScript are. Ensure that you use VBScript syntax and not Visual Basic syntax in the scripting environment.

Note: Microsoft's **.NET Framework 2.0 or higher** is a system prerequisite for Scripting Editor, and it must be installed before UModel is installed.

The Scripting Editor

The Scripting Editor (*screenshot below*) opens in a separate window and is accessed via the **Tools | Scripting Editor** menu command in the UModel GUI. The programming languages that can be used in the Scripting Environment are **JScript** and **VBScript**. The scripting language can be changed by right-clicking the Project item in the Project window, selecting **Scripting Language**, and selecting the language you want.



What you can do with the Scripting Editor

In the Scripting Editor, you can create Forms, Event Handlers, and Macros to build up a Scripting Project. A Scripting Project can then be set as the Global Scripting Project for UModel, thus enabling scripts in the Scripting Project to be used in the application. Additionally, different Scripting Projects can be assigned to different UModel projects, thus allowing different scripts to be used for different UModel projects.

Every script project can define the .NET runtime version it wants to use. An application can handle multiple scripting projects with different .NET runtime versions simultaneously, but the appropriate .NET version must be installed. For example, script projects with .NET 4.0 will only run on computers having .NET 4.0 installed.

Documentation about the Scripting Editor

The documentation describing the Scripting Environment (this section) is organized into the following parts:

- [An overview](#), which provides a high level description of the Scripting Editor and Scripting Projects.
- [A list of steps required to create a Scripting Project](#).
- [An explanation of Global Declarations](#), together with an example.
- [A description of how to create Forms](#).
- [A discussion of UModel-specific event handlers](#).
- [An explanation of how to use macros](#) in the Scripting Editor and in UModel.

20.2.1 Overview

The Scripting Editor provides an interface in which you can: (i) graphically design Forms while assigning scripts for components in the Form; (ii) create Event Handlers, and (iii) create Macros.

These Forms, Event Handlers, and Macros are organized into scripting projects, which are then assigned to UModel application projects and can be used in the application.

Variables and functions can be defined in a Global Declarations script, which is always executed before Macro or Event Handler scripts.

This section gives an overview of the Scripting Editor and Scripting Projects. It is organized into the following sections:

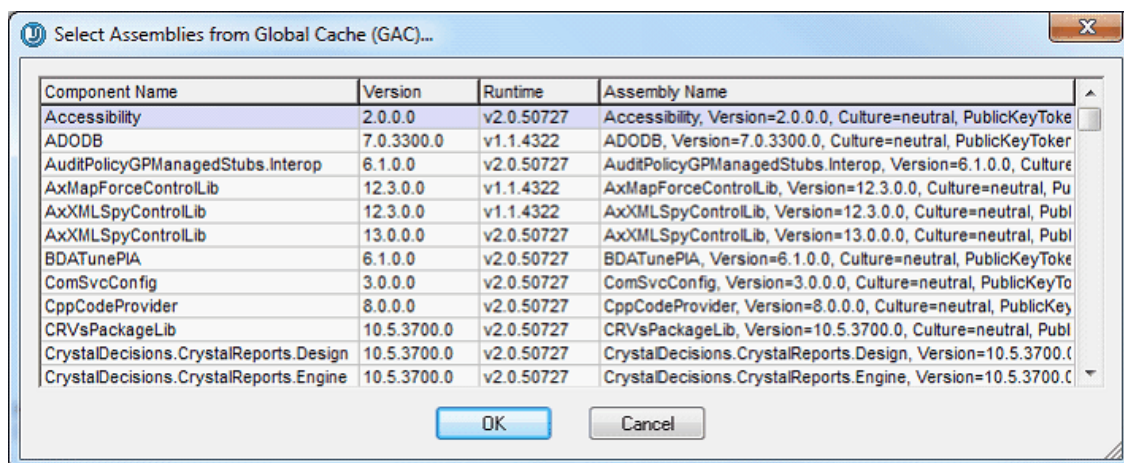
- [Scripting Projects in UModel](#), which describes how the scripting projects you create with the Scripting Editor will be used in UModel.
- [The Scripting Editor GUI](#), which provides a detailed look at the different parts of the Scripting Editor GUI and how they are to be used.
- [Components of a Scripting Project](#), which explains the different components that go to make up a scripting project.

The details about the creation of the various components ([Global Declarations](#), [Forms](#), [Event Handlers](#), and [Macros](#)) are described in their respective sections.

.NET assemblies

Every scripting project can have references to .NET assemblies—in addition to the default references. .NET assemblies can be added for the whole scripting project or for individual macros (by using the new `CLR.LoadAssembly` command in the source code; see [Built-in Commands](#)). Assemblies can be added, for example, from the Global Assembly Cache.

To add an assembly, right-click the project or macro, and, from the context menu that pops up, select **Add .NET Assembly | Assembly from Global Cache (GAC)**.



This works in the same way as with Visual Studio and allows access not only to the complete Microsoft .NET Framework but also to any user-defined assembly.

Scripting Projects in UModel

All scripts and scripting information created in the Scripting Editor are stored in **Altova Scripting Projects** (**.asprj** files).

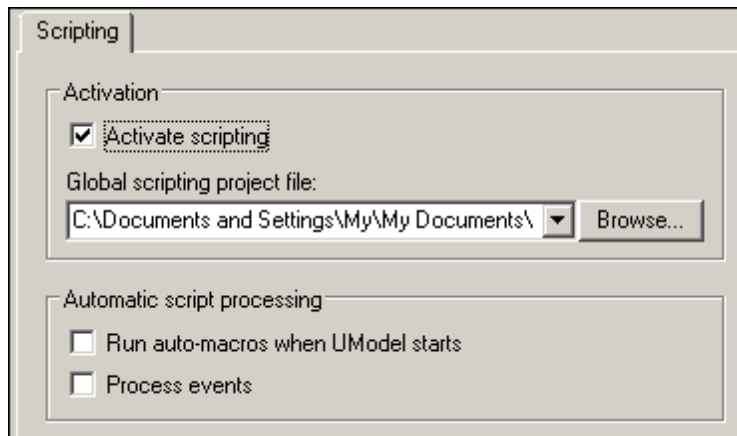
You can create any number of Altova Scripting Projects. After a scripting project has been created, it can be used in the following ways:

- It can be set as the global scripting project for UModel. Scripts in the global scripting project can then be called from within the application, and macros of the Global Scripting Project can be used for all UModel projects.
- It can be assigned to an UModel project (as an application project). When an UModel project is open in UModel, scripts in the associated scripting project can be called.

Your UModel package contains a sample scripting project called `ScriptSampleFind.asprj`. This file is located in the folder: `C:\Documents and Settings\<username>\My Documents\Altova\UModel2016\Examples\` and contains global declarations for a few standard tasks.

Setting the global scripting project of an application

The global scripting project of an application is set in the Scripting tab of the Options dialog of UModel (*screenshot below, Tools | Options*).



To set a global scripting project for UModel, check the *Activate Scripting* check box and then browse for the Altova Scripting Project (**.asprj**) file you want. You can also specify: (i) whether Auto-Macros in the scripting project should be automatically executed when UModel starts, and (ii) whether application event handler scripts in the project should be automatically executed or not; check or uncheck the respective check boxes accordingly.

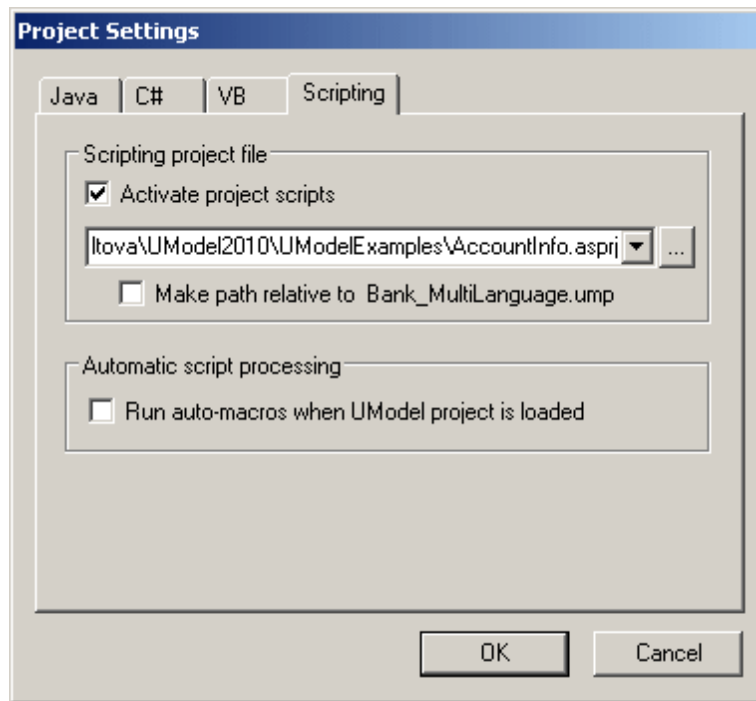
Note: Nested script execution is possible, i.e. Macros can call other macros, and events are received during macro, or event, execution.

Assigning a scripting project to an UModel project

A scripting project is assigned to an UModel project as follows:

1. In the UModel GUI, open the required application project.
2. Select the menu command **Project | Project Settings | Scripting** then click the

Scripting tab.



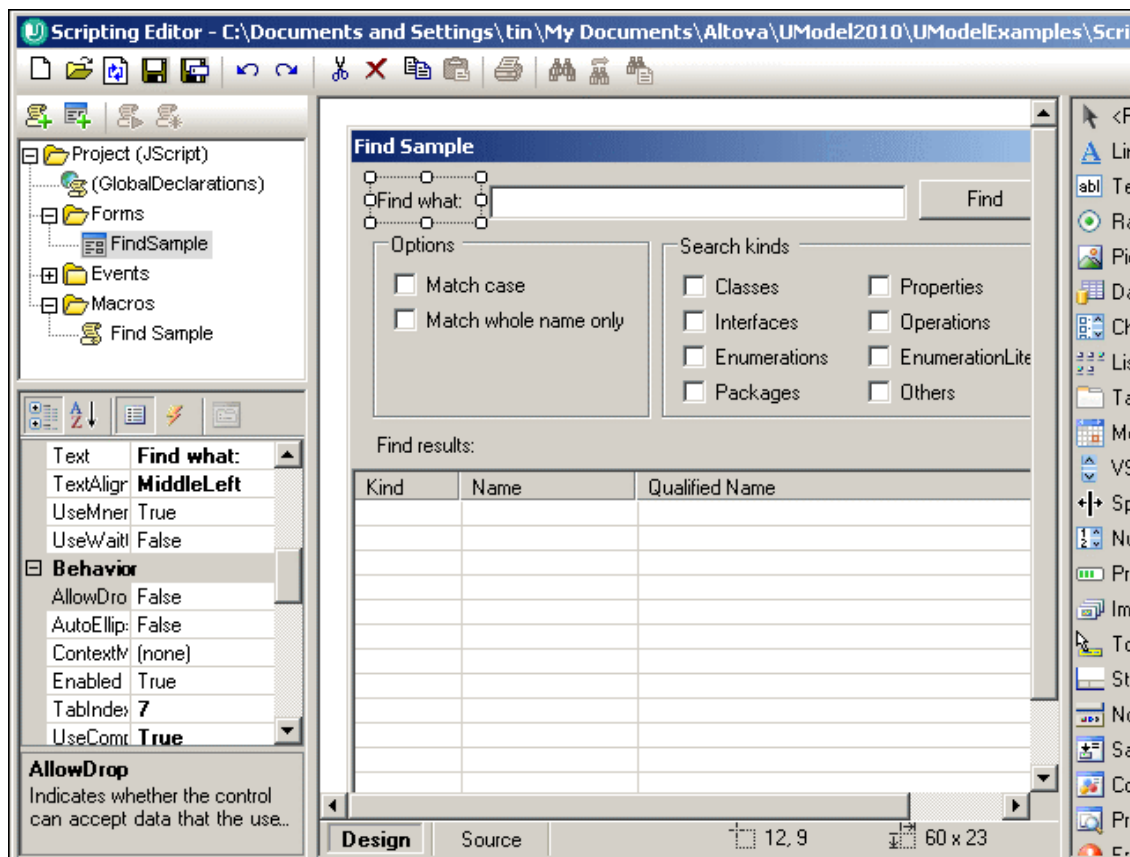
3. Check the *Activate Project Scripts* check box and select the required scripting project (.asprj file). If you wish to run Auto-Macros when the UModel project is loaded, check the *Run Auto-Macros* check box.
4. Click **OK** to finish.

Note: To deactivate (that is, unassign) the scripting project of an UModel project, uncheck the *Activate Project Scripts* check box.

The Scripting Editor GUI

The Scripting Editor GUI is shown below. It has the following parts:

- A [toolbar](#)
- A [Scripting Project Tree pane](#) (top left-hand side)
- A [Properties and Events pane](#) (bottom left)
- A [Main Window](#) with Design and Source tabs
- A [Form Object Palette](#) (right-hand side)



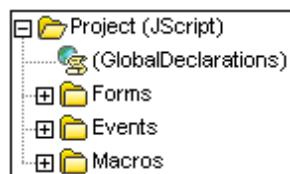
Scripting Editor toolbar

The Scripting Editor toolbar contains icons for:

- Standard file commands such as **New**, **Open**, **Save**, and **Print**. These commands are used to create new scripting projects, open existing scripting projects, and save and print scripting projects.
- Standard editing commands such as **Copy**, **Paste**, **Undo**, **Redo**, **Find**, and **Replace**. Note that the **Find** and **Replace** commands are applied to code in the Source tab of the Scripting Editor.

Scripting Project Tree

The Scripting Project Tree (*screenshot below*) shows the various components of the scripting project, structured along four main branches: (i) Global Declarations, (ii) Forms, (iii) Events, and (iv) Macros.



The Scripting Project Tree provides access to each component of the scripting project. For

example, in order to display and edit a particular Form, expand the Forms folder in the tree (see *screenshot above*), right-click the Form you wish to display or edit, and click **Open** from the context menu that pops up.

A quicker way to open a Form, Event, macro, or the Global Declarations script, is to double-click the respective icon, or text. To delete a Form or Macro from the scripting project, right-click the component and select the **Delete** command from the context menu.

The Scripting Project Tree pane contains a toolbar with icons (*screenshot below*).

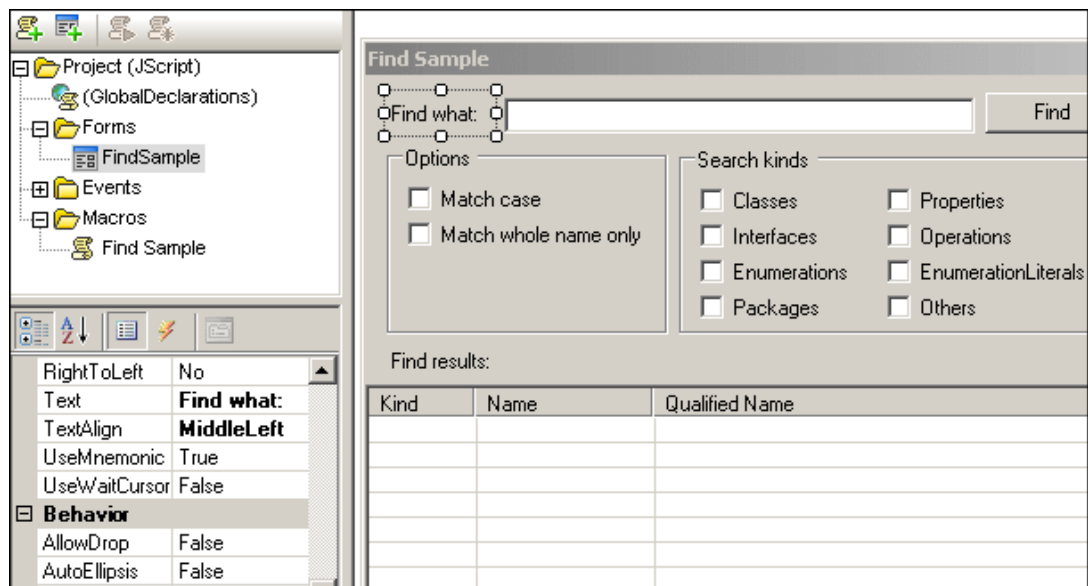


The icons, from left to right, are for: (i) [creating a new macro](#), (ii) [creating a new form](#), (iii) [running a macro](#), and (iv) [debugging a macro](#). These commands are also available in the context menu that appears when you right-click any component in the Scripting Project Tree.

Properties and Events

The Properties and Events pane (*screenshot below*) displays the following:

- Form properties, when the Form is selected
- Object properties, when an object in a Form is selected. (The screenshot below shows, at left, the properties of the object selected in the Form at right.)
- Form events, when a Form is selected
- Object events, when an object in a Form is selected



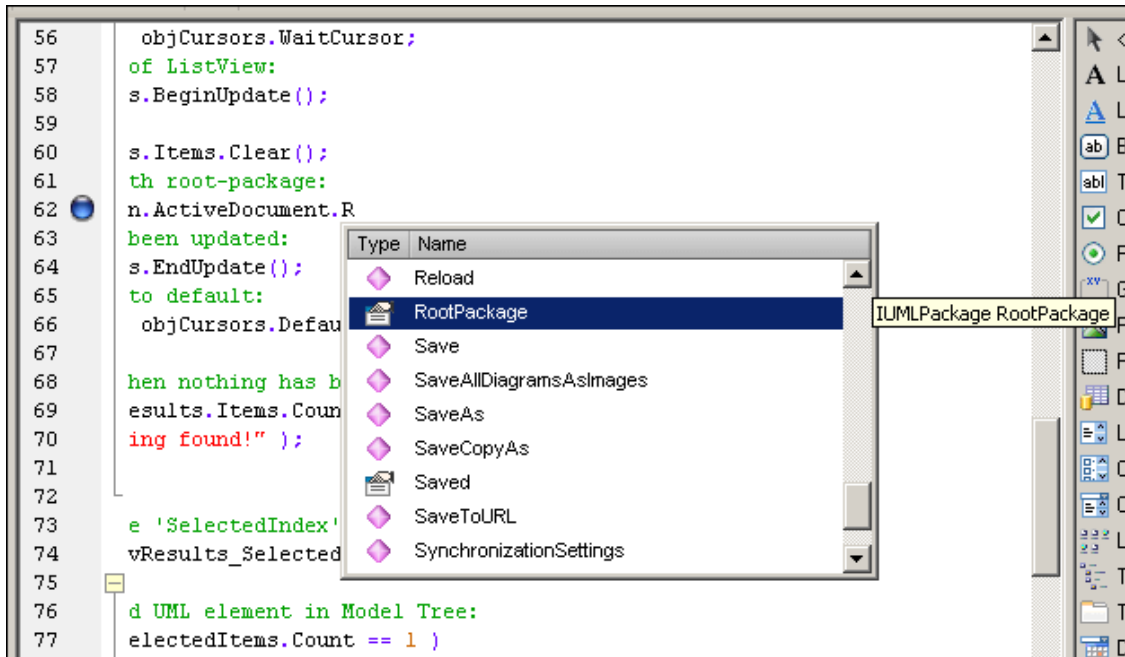
To switch between the properties and events of the selected component, click, respectively, the **Properties** icon (third from left in the Properties and Events toolbar, see *screenshot above*) and the **Events** icon (fourth from left).

The first and second icons from left in the toolbar are, respectively, the **Categorized** and **Alphabetical** icons. These display the properties or events either organized by category or organized in ascending alphabetical order.

When a property or event is selected, a short description of it is displayed at the bottom of the Properties and Events pane.

Main Window

The Main Window displays one component at a time and has one or two tabs depending on what is being displayed. If a Global Declarations script, an Event, or a Macro is being displayed, then a single tab, the Source tab, displays the source code of the selected component.



The Source tab supports:

- syntax coloring
- source code folding
- setting/deleting bookmarks using **CTRL+F2**
- autocompletion entry helper with parameter info (shown below)
- Goto Brace, Goto Brace Extend
- Zoom In / Zoom Out
- full method/property signature shown next to the autocompletion entry helper (shown above)

```

QualifiedName(
    IUMLNamedElement FindOwnedMemberWithQualifiedName( string strName )
  
```

- brace highlighting during code entry
- mouse over popups; placing the mouse over a known method or property, displays its signature (and documentation if available):

```

var sDir = objApplication.InstallationDirectory;
string IApplication.InstallationDirectory
installation directory of the application, e.g. C:\Program Files\Altova\UModel2009
  
```

If a **Form** is being displayed, then the Main Window has two tabs: a Design tab showing and enabling the layout of the Form, and a Source tab containing the source code for the Form. Content in both the Design tab and Source tab can be edited.

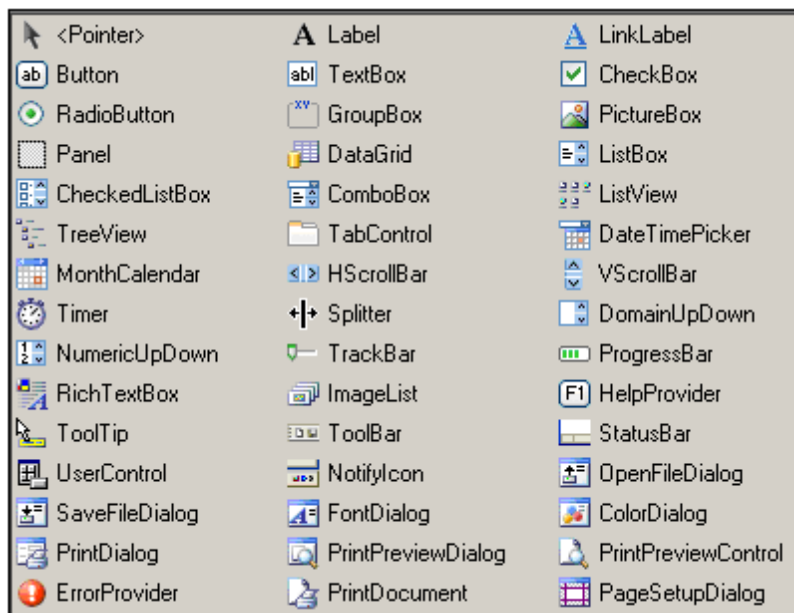
Note: Since JScript and VB Script are untyped languages, entry helpers and auto-completion is supported only in cases of "fully qualified constructs" such as `Application.ActiveDocument.RootPackage.xxx` and "predefined" names such as `objDocument123.RootPackage.xxx`.

If names start with `objUMLxxx`, members of the corresponding `IUMLxxx` interface will be shown. For example, the UModel COM API has an interface, `IUMLClass`. If while coding, you use `objUMLClass`, `objUMLClass123`, or `objUMLClassParent`, the members of the corresponding `IUMLClass` will be displayed.

If names start with `objApplication`, `objDocument`, or `objDiagramWindow`, then members of the corresponding interface will be shown. This also applies to all other interfaces defined in the UModel API. Auto-completion entry helper and parameter info are shown during editing, but can also be obtained on demand by pressing **Ctrl+Space**.

Form Object Palette

The Form Object Palette contains all the objects that are available for designing Forms and looks something like the screenshot below. Registered ActiveX controls can be added to the Form Object Palette by right-clicking the pane and selecting the **Add ActiveX Control** command



To insert an object from the Form Object Palette click the object you want in the palette, then click at the location in the Form where you wish to insert the object. The object will be placed at this location. In many cases you will need to supply some properties of the object via the Properties and Events pane. You can drag the object to other locations as well as resize it. Further, a number of editing commands, such as centering and stacking objects, can be

accessed via the context menu of the selected Form object.

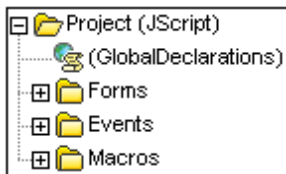
Some Form objects, such as `Timer`, are not added to the Form but are created as Tray Components in a tray at the bottom of the Main Window. You can select the object in the tray and set properties and event handlers for the object via the Properties and Events pane. For an example of how Tray Components are handled, see [Form usage and commands](#).

Components of a Scripting Project

An Altova Scripting Project consists of the following four major components:

- *Global Declarations*, a component which contains definitions of variables and functions that are available to, and can be used by, all Forms, Macros, and Event Handler scripts in the scripting project.
- *Forms*, a component which contains all the Forms defined in the scripting project.
- *Events*, a component which contains Event Handler scripts for all application-based—as opposed to Form-based—events.
- *Macros*, a component which contains all the Macros defined in the scripting project.

These components are displayed in and accessed via the Scripting Project Tree of the Scripting Editor (*screenshot below*).



Given below is a brief description of each of these components.

Global Declarations

The Global Declarations component is a script that contains variables and functions that can be used by Forms, Event Handlers, and Macros. The functions make use of the UModel API to access UModel functionality. Creating a variable or function in the Global Declarations module enables it to be accessed from all the Forms, Event Handlers and Macros in the scripting project.

To add a variable or function, open the Global Declarations component (by right-clicking it in the Scripting Project Tree and selecting **Open**) and edit the Global Declarations script in the Main Window. In this script, add the required variable or function.

Forms

In the Scripting Editor, you can build a Form graphically using a palette of Form objects such as text input fields and buttons. For example, you can create a Form to accept the input of an element name and to then remove all occurrences of that element from the active UMP project.

For such a Form, a function script can be associated with a text box so as to take an input variable, and an Event Handler can be associated with a button to start execution of the delete functionality, which is available in the UModel API. A Form is invoked by a call to it either within a function (in the Global Declarations script) or directly in a Macro. For details of how to create and edit Forms, see the [Forms](#) section.

Event handling

Event Handler scripts can be associated with a variety of available events. You can control events that occur both within Forms ([Form events](#)) and within the general application interface ([application events](#)). The script associated with an event is executed immediately upon the triggering of that event.

Most events have parameters which provide detailed information about the event. The return value from the script typically instructs the application about how to continue its processing (for example, the application may not allow editing).

An Event Handler runs when the relevant event occurs in the Form or in UModel. For details about how to create event handlers, see [Event Handlers](#).

Macros

Macros are used to implement complex or repetitive tasks. Macros do not use either parameters or return values.

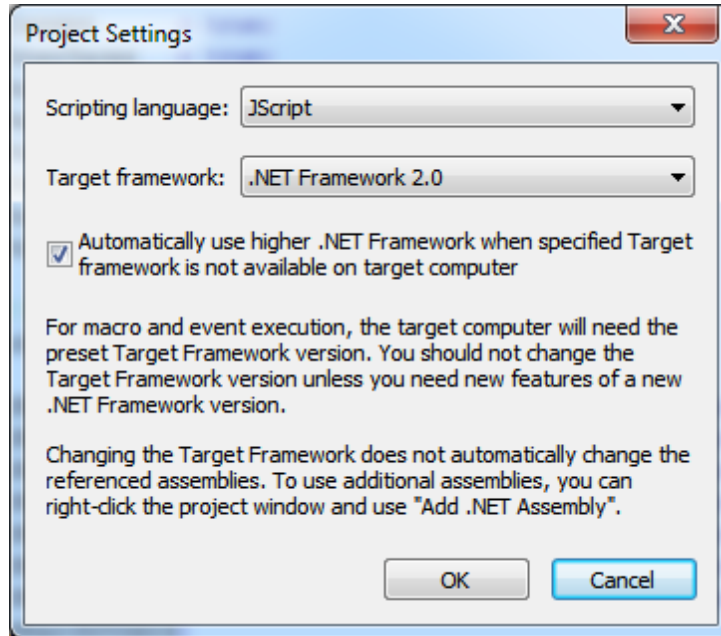
In a Macro, it is possible to access all variables and functions declared in the Global Declarations and to display Forms for user input.

For a simple example of creating a Macro, see [Writing a Macro](#). Also see [Running Macros](#) for a description of the ways in which a Macro can be called. A Macro is run from within the UModel interface by clicking **Tools | Macros | [MacroName]**

20.2.2 Creating a Scripting Project

The broad steps for creating a Scripting Project are as follows:

1. Open the Scripting Editor by clicking the command **Tools | Scripting Editor**.
2. In the Scripting Editor, open a new scripting project by clicking the **New** icon in the Scripting Editor toolbar. The Project Settings dialog (*screenshot below*) pops up. You can also access this dialog by right-clicking a project in the Scripting Project Tree pane (in the top left part of the Scripting Editor window) and clicking the command **Project Settings**.



Select either JScript or VBScript in the first combo box and the .NET Framework in the second combo box. To enable higher .NET Frameworks (such as .NET Framework 4.5 on Windows 8), check the *Automatically use higher .NET Framework* check box. Then click **OK**. The new Scripting Project is created.

3. Click the **Save** icon in the Scripting Editor toolbar to save the Scripting Project as a .asprj file.
4. A Scripting Project can be considered to be made up of several components that work together. These components will typically be a combination of: Global Declarations, Forms, Events, and Macros. They can be created in any order, but you should clearly understand how they work together. The way each type of component is called and executed is [described below](#). How to create each type of component is described in the respective sections about the component type.
5. After you have finished creating all the required components, save the Scripting Project (by clicking the **Save** icon in the Scripting Editor toolbar).
6. Close the Scripting Editor.

Note: Right-clicking the Project folder and selecting **Project Settings** lets you change the scripting language at any time.

How Forms, Event Handlers, and Macros are called and executed

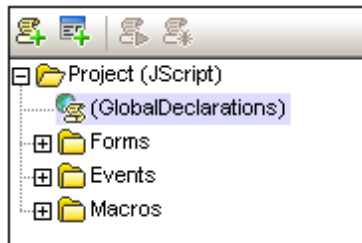
Forms, Event Handlers, and Macros are all created in the Scripting Editor. However, the way they are called and executed is different for each and has a bearing on how you create your scripting

projects.

- A Form is invoked by a call to it either within a function in the Global Declarations script or directly in a Macro.
- An Event Handler runs when the relevant event occurs in UModel. If an Event Handler for a single event is defined in both the Global Scripting Project and the UModel-project-specific Scripting Project, then the event handler for the project-specific Scripting Project is executed first and that for the Global Scripting Project immediately afterwards.
- A Macro is executed from within the UModel interface by clicking **Tools | Macros | [MacroName]**. In a Macro, it is possible to access all variables and functions declared in the Global Declarations and to display Forms for user input.

20.2.3 Global Declarations

The Global Declarations component is present by default in every Scripting Project (see *screenshot below*), and therefore does not have to be created. In order to add variables and functions to the Global Declarations script of a Scripting Project, you need to open the Global Declarations script and add the code fragment to the Global Declarations script. See [Components of a Scripting Project](#) and [Creating a Scripting Project](#) for more information.



To open the Global Declarations script of a Scripting Project, right-click the *Global Declarations* item in the Scripting Project Tree (*screenshot above*), and select **Open**. The Global Declarations script opens in the Main Window.

Note: Every time a macro is executed or an event handler is called, global declarations are re-initialized.

20.2.4 Forms

Creating and editing Forms in the Scripting Editor consists of the following steps:

1. [Creating a New Form](#). The new Form is created and named, and has properties defined for it.
2. [Designing the Form](#). A Form is designed by adding Form Objects to it and assigning values for the different Form Objects.
3. [Scripting Form Events](#). Scripts are assigned to Form-related events.

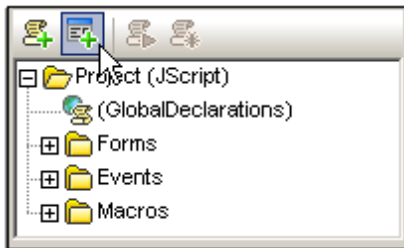
Creating a New Form

Creating a new Form in the Scripting Editor involves the following steps:

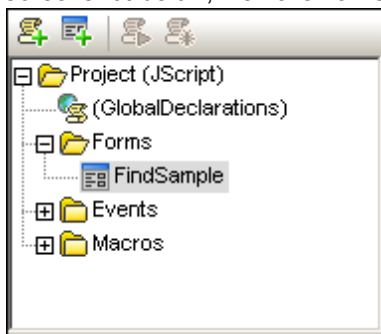
1. [Creating a new Form and naming it](#)
2. [Specifying the properties of the Form](#)

Creating a new Form and naming it

To add a new Form to a scripting project, click the **Add Form** icon (*highlighted in screenshot below*) in the toolbar of the Project Overview pane. Enter the name of the new Form.

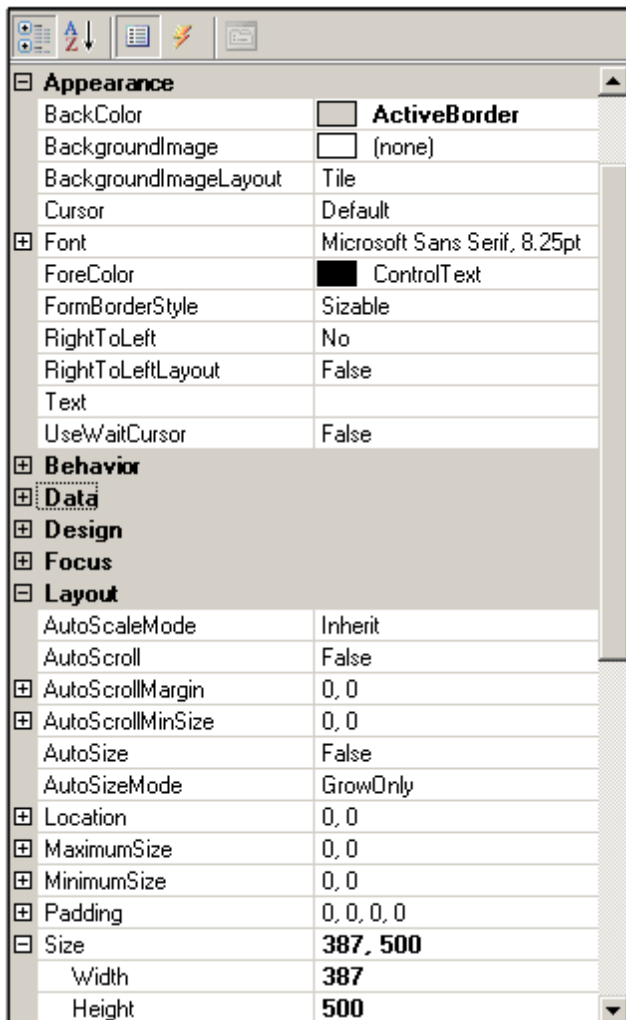


A new Form is added to the project. It appears in the Main Window and an entry for it is created in the Scripting Project Tree pane, under the Forms heading. Press the F2 function key to rename the form, or right click the form name and select Rename from the context menu. In the screenshot below, we have named the new Form *FindSample*.



Form properties

The properties of the Form, such as its size, background color, and font properties, can be set in the Properties pane. The screenshot below shows the size and background-color property values in bold, in the *Layout* and *Appearance* categories, respectively.



Testing a Form

You can test a form in the Scripting Editor by right-clicking it in the Project Overview pane and selecting the **Test Form** Command.

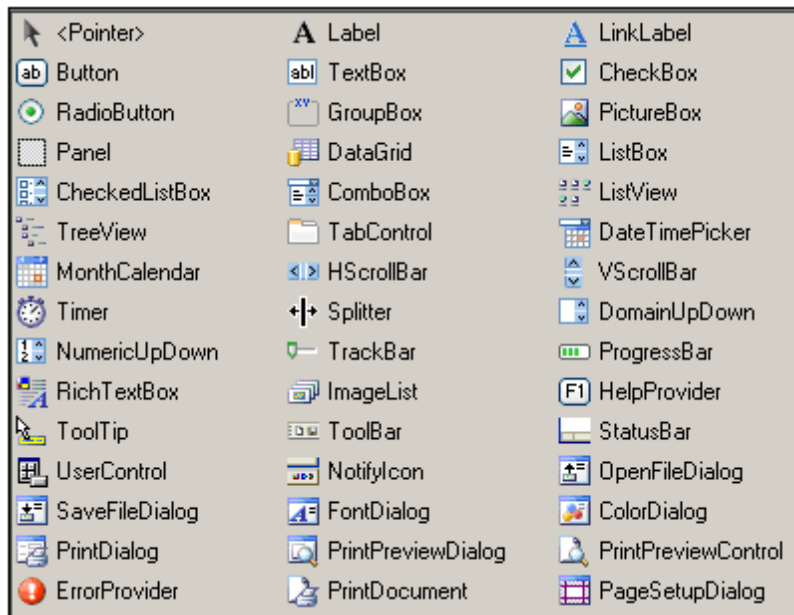
Form Design and Form Objects

Designing a Form consists of the following steps:

- Placing an object from the [Form Object Palette](#) in the Form design.
- Assigning values for the [properties of individual Form Objects](#).
- [Assigning scripts for Form-based events](#).

The Form Object Palette

The Form Object Palette contains all the objects that are available for designing Forms and looks something like the screenshot below. Registered ActiveX controls can be added to the Form Object Palette by right-clicking the pane and selecting the **Add ActiveX Control** command



To insert an object from the Form Object Palette click the object you want in the palette, then click at the location in the Form where you wish to insert the object. The object will be placed at this location. In many cases you will need to supply some properties of the object via the Properties and Events pane. You can drag the object to other locations as well as resize it. Further, a number of editing commands, such as centering and stacking objects, can be accessed via the context menu of the selected Form object.

Some Form objects, such as `Timer`, are not added to the Form but are created as Tray Components in a tray at the bottom of the Main Window. You can select the object in the tray and set properties and event handlers for the object via the Properties and Events pane. For an example of how Tray Components are handled, see [Form usage and commands](#).

Some of the most commonly used objects are described below:



Label: Adds text fields such as captions or field descriptions.



Button: Adds a button. It is possible to assign bitmaps as background images for these buttons.



Check Box: Adds a check box, which enables `Yes/No` type selections.



Combo Box: Adds a combo box, which allows the user to select an option from a drop-down menu.



List Box: Adds a list box, which displays a list of items for selection.



TextBox: Enables the user to enter a single line of text.



Rich TextBox: Enables the user to enter multiple lines of text.

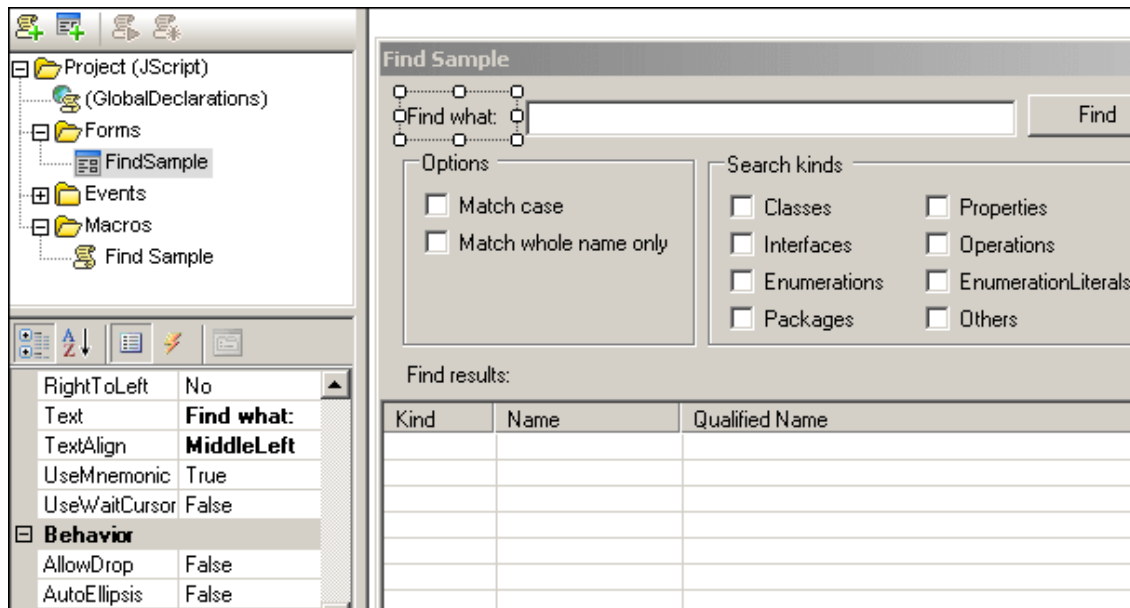
Creating objects and setting their properties

To create an object in the Form, first select the required object in the Form Object Palette and then click the location in the Form where you want to insert it. After the object has been inserted, you can resize it as well as drag it to another location in the Form.

When an object is selected in the design, you can specify its properties in the Properties and Events pane. In the toolbar of the Properties and Events pane, click the Properties icon to display a list of the object's properties.

For example, in the screenshot below, the Label object with the text *Balance* has been selected in the design. In the Properties and Events pane, the name of the object (which is the name that is to be used to identify the object in code, `Label1` in the screenshot below) is given in the *Design* category of properties; in this case, the name of the object is `Label1`.

The text of the label (which is what appears in the Form) must be entered as the value of the *Text* property in the *Appearance* category of properties.



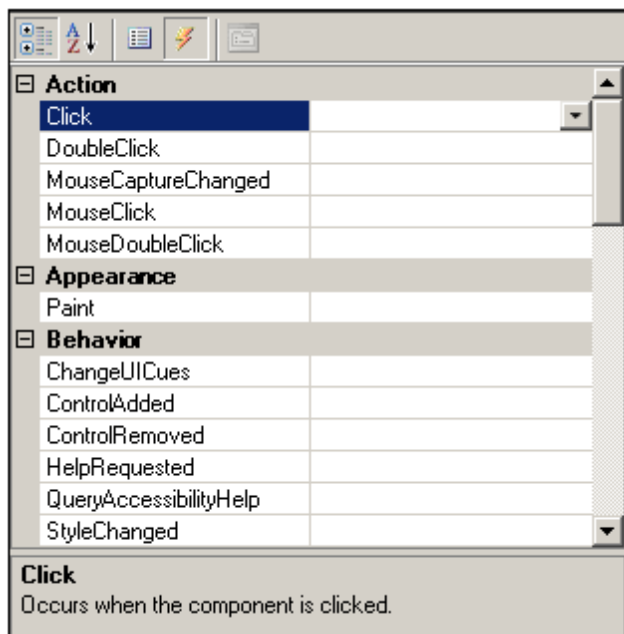
To assign other object properties, enter values for them in the Properties and Events pane.

Testing a Form

You can test a form in the Scripting Editor by right-clicking it in the Project Overview pane and selecting the **Test Form** Command.

Form Events

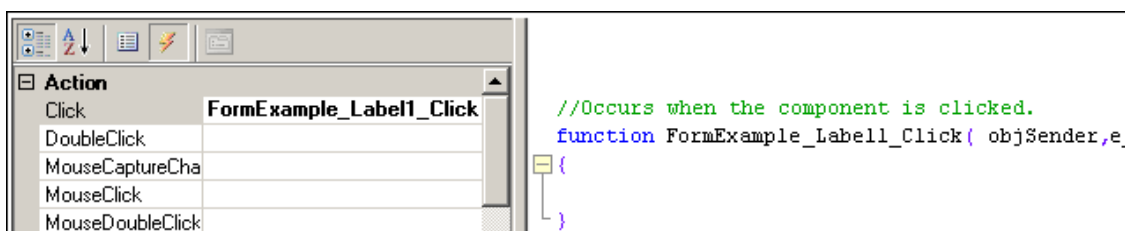
When an object is selected in the design, clicking on the Events icon in the toolbar of the Properties and Events pane (*fourth icon from left*), displays all the events available for that object (see screenshot below). These can be displayed either by category (screenshot below) or alphabetically.



For each event, you can enter the name of an existing event handler or function. Alternatively:

- you can double click on an event to create: (i) an empty function script in the *Source* tab of the Main Window, and (ii) an association of the newly created function with the selected event.
- double click a button in the design tab, to directly generate the handler stub in the code window.

The screenshot below was taken after the *Click* event was double-clicked. Notice that an empty event handler function called `FormExample_Label1_Click` has been created in the Main Window and that, in the Properties and Events pane, this function has been associated with the *Click* event.



Enter the required scripting code and save the project.

Writing the required scripts

After the visual design of the form is complete, form objects will typically be associated with suitable scripts. The example below is a script that adds colors when a button is clicked. The script is inserted as an event handler for the *Click* event of the button `Button1` (the event is available in the Properties and Events pane when the button is selected in the design):

```
function FormExample_Button1_Click( objSender, e_EventArgs )
{
    // Sets the ForeColor (red) of the button.
```

```
objSender.ForeColor = CLR.Static( "System.Drawing.Color" ).Red;
// Sets the BackColor (blue) of the button.
objSender.BackColor = CLR.Static( "System.Drawing.Color" ).Blue;
// Sets the form BackColor (green).
objSender.FindForm().BackColor =
CLR.Static( "System.Drawing.Color" ).Green;
}
```

20.2.5 Events

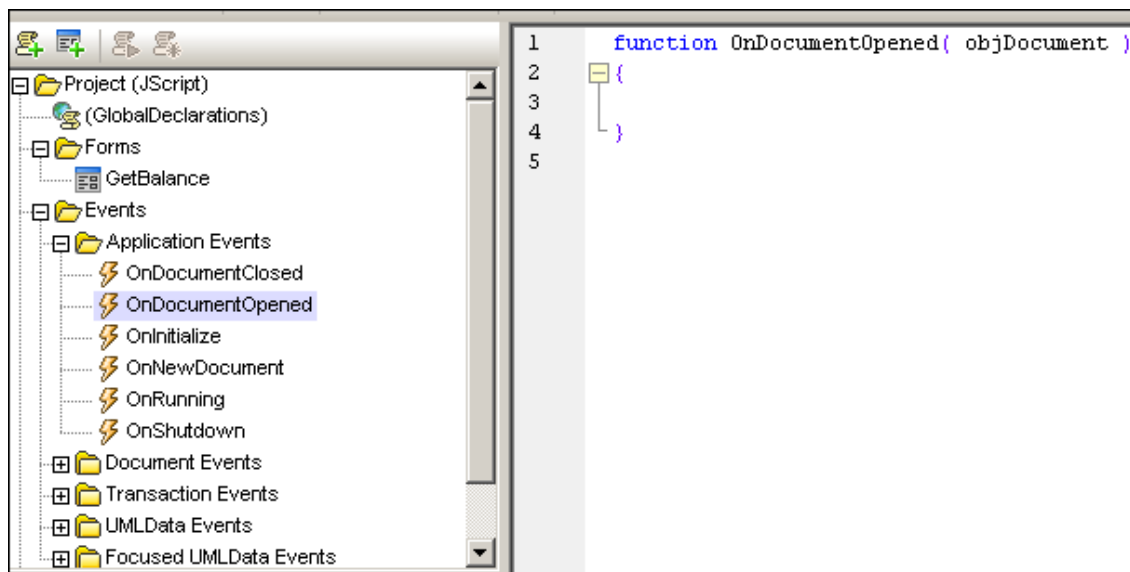
The Events folder of the scripting project (see *screenshot below*) contains folders for the following type of events:

- Application Events
- Document Events
- Transaction Events
- UMLData Events
- Focused UMLData Events

Note that these events are UModel-specific, as opposed to Form-based events. Each of the folders listed above contains a set of events for which Event Handler scripts can be written.

Application Events, for example, are shown in the screenshot below.

To access the event handler script of any of these events, right-click the event and select **Open** from the context menu. The script will be displayed in the Main Window (see *screenshot below*) and can be edited there. After you have finished editing the script, save changes by clicking the **Save** command in the toolbar of the Scripting Editor.



Note the following points:

- Event Handlers need function headers with the correct spelling of the event name. Otherwise the Event Handler will not be called.
- It is possible to define local variables and helper functions within Macros and Event Handlers. In order for events to be processed, the Process Events options must be toggled on in the Scriptings options of UModel. See [Scripting Projects in UModel](#) for details.
- Also see [Programming Points](#).

20.2.6 Macros

Macros automate repetitive or complex tasks. In the Scripting Environment, you can create a script that calls application functions as well as custom functions that you have defined. This flexibility provides you with a powerful method of automating tasks within UModel. This section about macros is organized as follows:

- [Creating and Editing a Macro](#) describes how to create a new macro and edit an existing one.
- [Running a Macro](#) explains how a macro can be run from the Scripting Editor and from the broader UModel environment as well.
- [Debugging](#) describes how macros can be debugged.

Key points about macros

Given below is a summary of important points about macros.

- Any number of macros can be added to the active scripting project. These macros are saved in the Altova Scripting Project file (.asprj file).
- Functions that are used in a macro can be saved as a Global Declaration. All Global Declarations are also saved in the Altova scripting project file (.asprj file).
- The macro can be tested by running it from within the Scripting Editor, and it can be debugged from within the Scripting Editor.
- UModel can have one global Scripting Project, and a second scripting project, assigned to the currently loaded project, active at any one time; the macros are available to both of them. See [Running a Macro](#) for details.

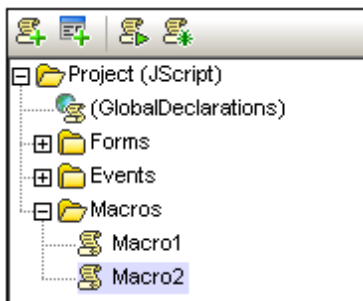
Creating and Editing a Macro

The following operations enable you to create a new macro and edit an existing macro.

Creating a new macro

Right-click the Macro folder in the Scripting Projects tree and select **Add Macro** from the context menu. (The **Add Macro** command can also be selected from the context menu of any item in the Scripting Projects tree.) Alternatively, click the **New Macro** icon in the toolbar of the Scripting Projects tree.

The newly created (and empty) macro document is displayed in the Main Window, and the name of the macro is displayed in the title bar of the Scripting Editor (*screenshot below*).



Naming or renaming a macro

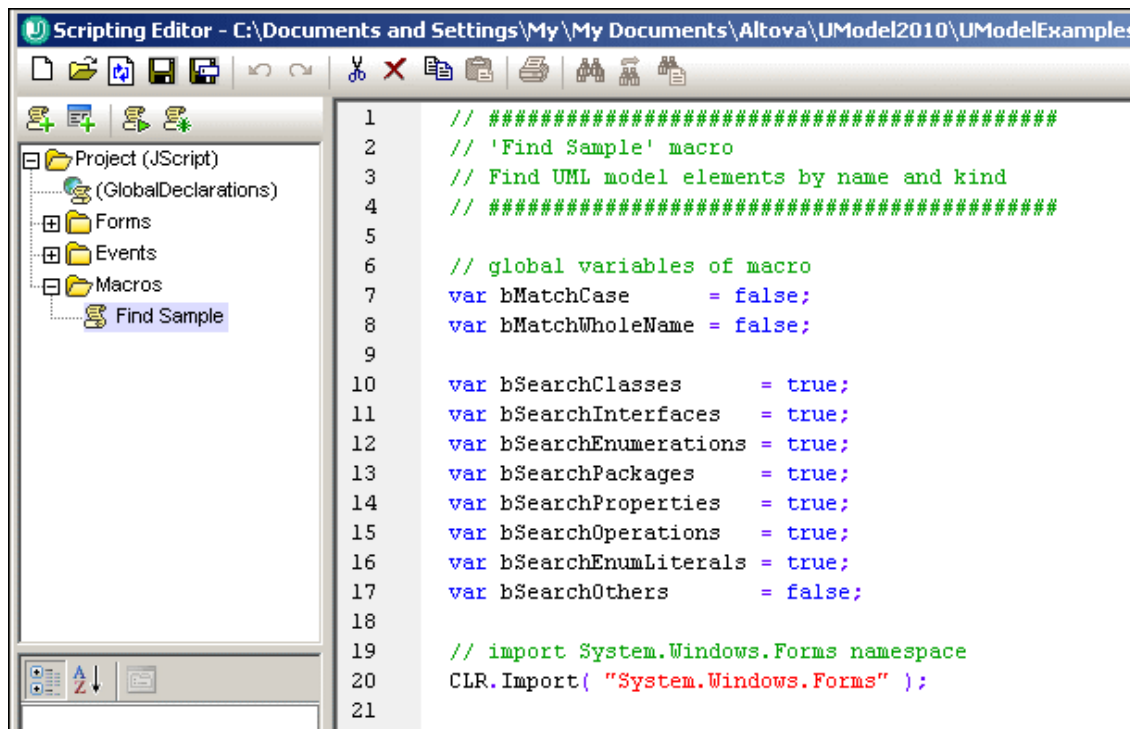
To name or rename a macro, click the macro name in the Scripting Project tree and press the **F2**

function key, or right click the name and select **Rename** from the context menu.

Opening a macro

To open a macro, right-click the macro in the Macros folder of the Scripting Project tree (see *screenshot above*), and select the **Open** command. The macro is displayed in the Main Window and its name is displayed in the title bar of the Scripting Editor (*screenshot below*). Alternatively, double-clicking a macro in the Scripting Project tree opens it in the Main Window.

The screenshot shows the "Find Sample" macro of the **Bank_MultiLanguage.ump** project file available in the ...\\UmodelExamples folder.



Editing the macro

To edit a macro, enter or edit its code in the Main Window.

An example of a macro is given below. This macro finds UML elements by name and type.

```

// #####
// 'Find Sample' macro
// Find UML model elements by name and type
// #####

// global variables of macro
var bMatchCase      = false;
var bMatchWholeName = false;

var bSearchClasses   = true;
var bSearchInterfaces = true;
var bSearchEnumerations = true;

```

```
var bSearchPackages          = true;
var bSearchProperties        = true;
var bSearchOperations        = true;
var bSearchEnumLiterals     = true;
var bSearchOthers            = false;

// import System.Windows.Forms namespace
CLR.Import( "System.Windows.Forms" );

// verify if UML element kind should be considered
function MatchKind( objUMLElement )
{
    if ( objUMLElement.KindName == "Class" ) return
bSearchClasses;
    if ( objUMLElement.KindName == "Interface" ) return
bSearchInterfaces;
    if ( objUMLElement.KindName == "Enumeration" ) return
bSearchEnumerations;
    if ( objUMLElement.KindName == "Package" ) return
bSearchPackages;
    if ( objUMLElement.KindName == "Property" ) return
bSearchProperties;
    if ( objUMLElement.KindName == "Operation" ) return
bSearchOperations;
    if ( objUMLElement.KindName == "EnumerationLiteral" ) return
bSearchEnumLiterals;
    return bSearchOthers;
}

// verify if UML element name matches search options
function MatchName( objUMLNamedElement, sFindText )
{
    var sName = objUMLNamedElement.Name;
    if ( !bMatchCase )
        sName = sName.toUpperCase();

    if ( bMatchWholeName )
        return ( sName == sFindText )
    else
        return ( sName.indexOf( sFindText ) != -1 )
}

// recursively search all elements
function Find( objUMLElement, sFindText )
{
    // we are only interested in named elements:
    if ( objUMLElement.IsKindOf( "NamedElement" ) )
    {
        var objUMLNamedElement = objUMLElement;

        // only list elements where kind and name match:
        if ( MatchKind( objUMLNamedElement ) &&
            MatchName( objUMLNamedElement, sFindText ) )
        {
            // create a new ListViewItem class

```

```

        var newItem = CLR.Create( "ListViewItem",
objUMLNamedElement.KindName );
        // and add it to the listview - items:
        lastForm.lvResults.Items.Add( newItem );
        // append name and qualified name in sub-items:
        newItem.SubItems.Add( objUMLNamedElement.Name );
        newItem.SubItems.Add( objUMLNamedElement.QualifiedName );
    }
}

// also iterate over all children (ownedElements):
var ownedElements = objUMLElement.OwnedElements;
var itr = new Enumerator( ownedElements );
for ( ; !itr.atEnd(); itr.moveNext() )
    Find( itr.item(), sFindText );
}

// create and show the form:
ShowForm( "FindSample" );

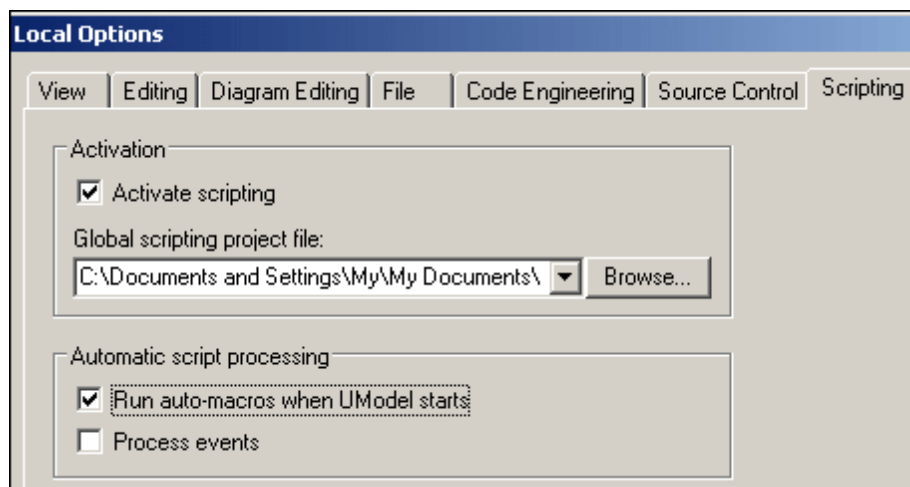
```

Note: Macros do not support parameters or return values.

Setting a macro as an Auto-Macro

When a macro is set as an Auto-Macro it can be run automatically when: (i) UModel is started, or (ii) an Altova UModel project is loaded in UModel. To specify whether Auto-Macros should be run in each of these two events, check the *Run Auto-Macros* option in the Automatic Script Processing pane of the relevant dialogs:

- *When UModel is started:* the Scripting tab of the UModel Options dialog (**Tools | Options** menu command).
- *When an UModel project is loaded into UModel:* the Scripting dialog (screenshot below, **Project | Scripting Settings** menu command).



To set a macro as an Auto-Macro, right-click the macro in the Scripting Project tree and select the command **Set as Auto-Macro**. This is a toggle command; so to remove the Auto-Macro setting of a macro, select the command again.

Running a Macro

To run a macro in the Scripting Editor, right-click the macro in the Scripting Project tree and select the command **Run Macro**.

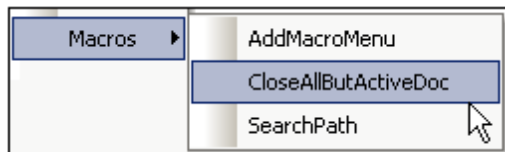
There are different ways to run a macro from UModel:

- [Via the Tools | Macros menu](#) of UModel.
- [By creating and using a toolbar button](#) for a macro.

Note that only one macro can be run at a time. After a macro (or event) is executed, the script is closed and global variables lose their values.

The UModel command to run Macros

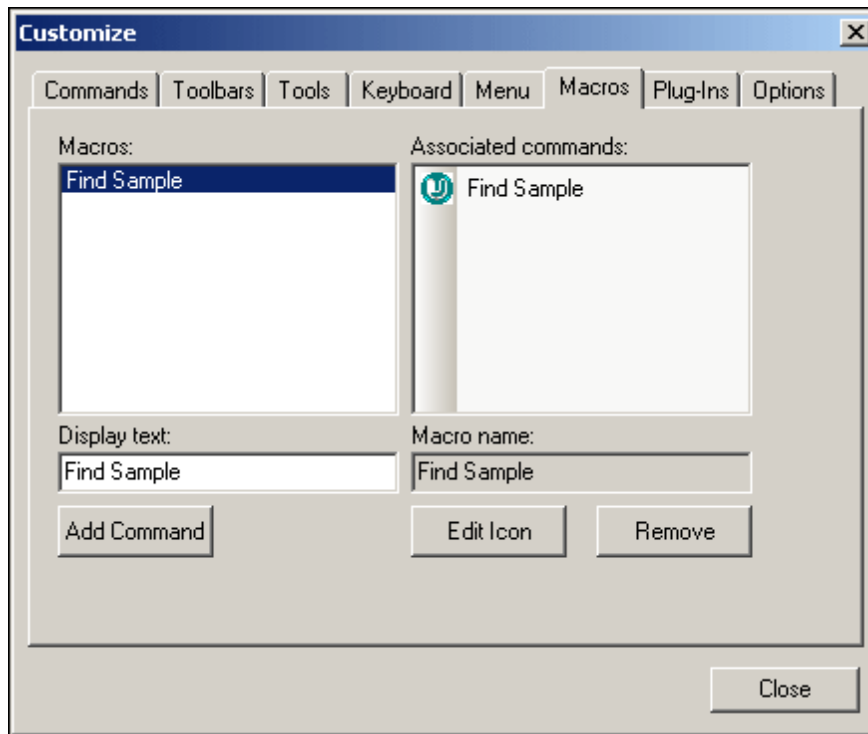
The **Tools | Macros** menu command (*screenshot below*) opens a submenu containing the macros defined in the Scripting Project that is currently active in UModel. The active Scripting Projects are specified in the Scripting tab of the Options dialog, or in the Scripting tab of the project settings.



From the submenu of available macros, select the macro to run. The macro will be executed.

Toolbar icon

You can create an icon in the toolbar or a menu item that runs a selected macro. To do this, click **Tools | Customize | Macros**. This causes the Customize dialog to be displayed (*screenshot below*).



Now do the following:

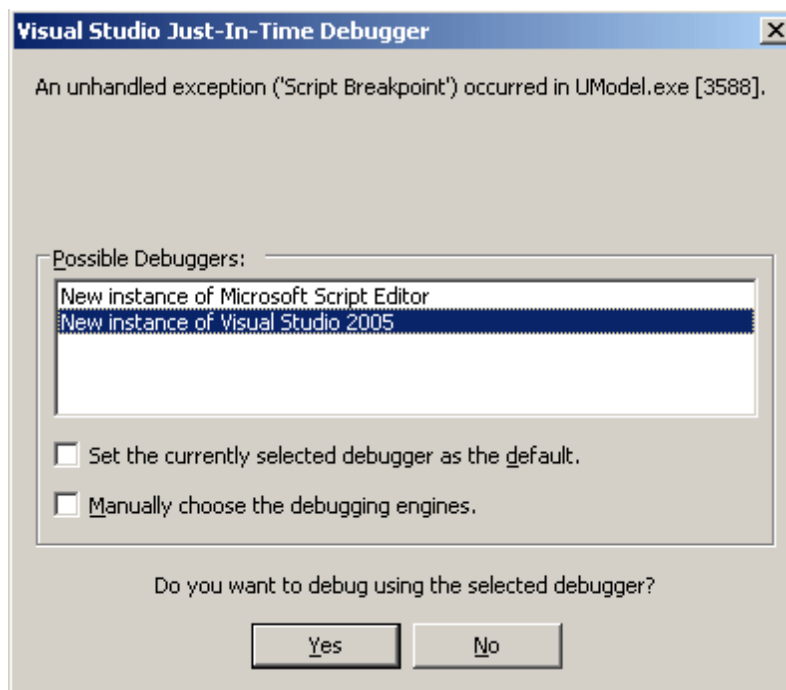
1. In the Macros tab of the Customize dialog, select the required macro from the Macros pane. The macros in the Macros pane are those in the active Scripting Project (which is specified in the Scripting tab of the Options dialog).
2. In the *Display Text* input field enter the name of the icon. This name will appear when the cursor is placed over the icon when it is in the toolbar.
3. Click **Add Command** to add it to the list of commands.
4. Select the command and click **Edit Icon** to create a new icon.
5. Drag the finished icon from the *Associated Commands* pane and drop it on to the toolbar or menu when the cursor changes from an arrow to an I-beam or line.
6. Macros can even be assigned their own shortcuts in the Keyboard tab of the Customize dialog (see *screenshot above*).

To remove the toolbar icon, open the Macros tab of the Customize dialog and drag the icon out of the toolbar and into the *Associated Commands* pane. Select the command in the *Associated Commands* pane and click **Remove** to remove the command from the pane.

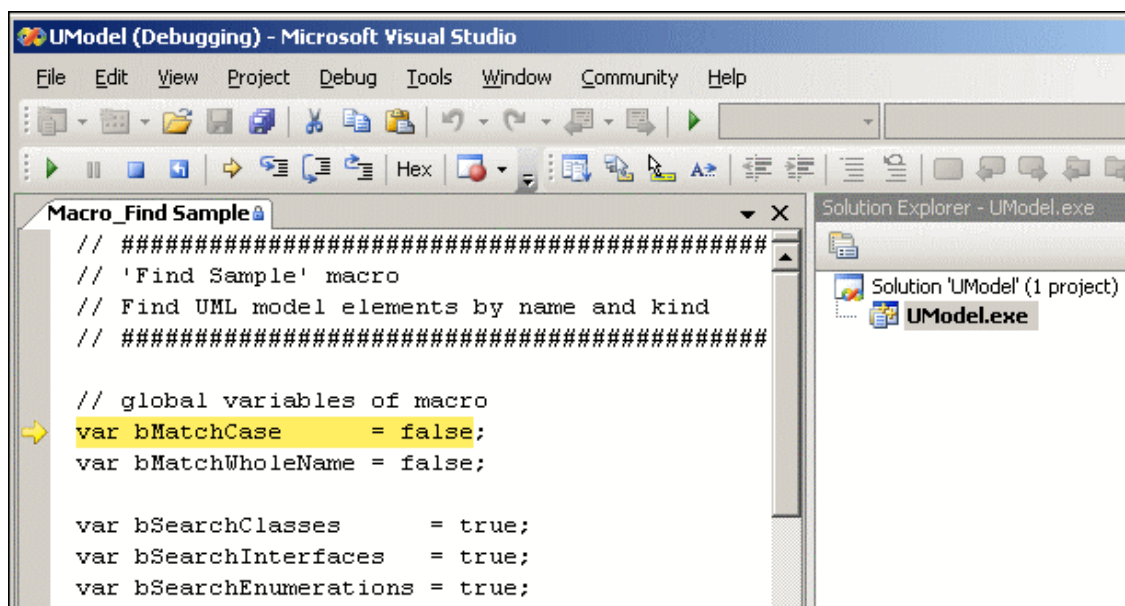
Debugging a Macro

You can debug a macro using an installed debugger. To do this, right-click the macro in the Scripting Project tree and select the command **Debug Macro**.

This pops up the Just-In-Time Debugging dialog (*screenshot below*), which lists the debuggers available on the machine. Select the debugger you wish to use and click **Yes**.



The selected debugger starts.



20.2.7 Programming Points

The following programming points should be noted:

- All namespaces and types of the following .NET assemblies can be accessed in the Microsoft .NET Framework per default:

```
System
System.Data
System.Design
System.Drawing
System.Windows.Forms
System.XML
```

Additional assemblies can be added to the scripting project via the [project's context menu](#), or dynamically (at runtime) in the source code by using [CLR.LoadAssembly](#).

Out-parameters from methods of the .NET Framework require special variables in JScript. For example:

```
var dictionary =
CLR.Create( "System.Collections.Generic.Dictionary< System.String,
System.String >" );
dictionary.Add("1", "A");
dictionary.Add("2", "B");

// use JScript method to access out-parameters
var strOut = new Array(1);
if ( dictionary.TryGetValue("1", strOut) ) // TryGetValue will set
the out parameter
    alert( strOut[0] ); // use out parameter
```

- .NET Methods that require integer arguments should not be called directly with JScript Number Objects which are Floating Point Values.

For example, instead of:

```
var objCustomColor = CLR.Static( "System.Drawing.Color" ). FromArgb( 128,
128, 128 );
```

use:

```
var objCustomColor = CLR.Static( "System.Drawing.Color" ).
FromArgb( Math.floor( 128 ), Math.floor( 128 ), Math.floor( 128 ) );
```

- To iterate .NET collections the JScript Enumerator as well as the .NET iterator technologies can be used:

For example:

```
// iterate using the JScript iterator
var itr = new Enumerator( coll );
for ( ; !itr.atEnd(); itr.moveNext() )
    alert( itr.item() );

// iterate using the .NET iterator
var itrNET = coll.GetEnumerator();
```

```
while( itrNET.MoveNext() )  
    alert( itrNET.Current );
```

- .NET templates can be instantiated as shown below:

```
var coll =  
CLR.Create( "System.Collections.Generic.List<System.String>" );
```

or

```
CLR.Import( "System" );  
CLR.Import( "System.Collections.Generic" );  
var dictionary = CLR.Create( "Dictionary< String, Dictionary<  
String, String > >" );
```

- .NET Enum values are accessed as shown below:

```
var enumValStretch =  
CLR.Static( "System.Windows.Forms.ImageLayout" ).Stretch;
```

- Enumeration literals, as defined in the Altova type libraries, can now be used instead of numerical values.

```
objExportXMIFileDlg.XMIType = eXMI21ForUML23;
```

Built-in Commands

This section lists:

- [Built-in commands](#)
[alert](#)
[conform](#)
[doevents](#)
[createForm](#)
[lastform](#)
[prompt](#)
[showform](#)
[watchdog](#)
- [.NET interoperability](#) commands
[CLR.Create](#)
[CLR.Import](#)
[CLR.LoadAssembly](#)
[CLR.ShowImports](#)
[CLR.ShowLoadedAssemblies](#)
[CLR.Static](#)

Built-in commands

The following built-in commands are available.

ShowForm(strFormName : String)

Instantiates a New Form object from the given form name and immediately shows it as Dialog.

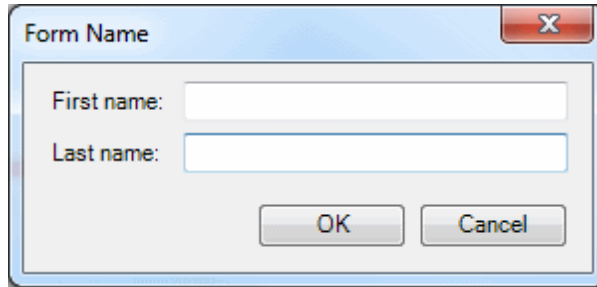
Return Value: A Number that represents the generated DialogResult

(System.Windows.Forms.DialogResult).

Example:

```
var dialogResult = ShowForm( "FormName" );
```

Shows Form "FormName" as Dialog:



The DialogResult can be evaluated e.g. by:

```
if ( dialogResult ==  
CLR.Static( "System.Windows.Forms.DialogResult" ).OK )  
    alert( "ok" );  
else  
    alert( "cancel" );
```

CreateForm(strFormName : String)

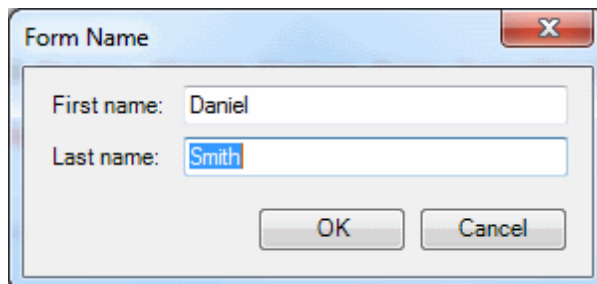
Instantiates a New Form object from the given Form name.

Return Value: The Form object (System.Windows.Forms.Form) of the given name, or null if no Form with such name exists.

Example:

```
var myForm = CreateForm( "FormName" );  
if ( myForm != null )  
{  
    myForm.textboxFirstName.Text = "Daniel";  
    myForm.textboxLastName.Text = "Smith";  
    var dialogResult = myForm.ShowDialog();  
}
```

Shows Form "FormName" as Dialog - TextBoxes are initialized:



The DialogResult can be evaluated e.g. by:

```
if ( dialogResult ==  
CLR.Static( "System.Windows.Forms.DialogResult" ).OK )  
    alert( "ok" );  
else  
    alert( "cancel" );
```

lastform

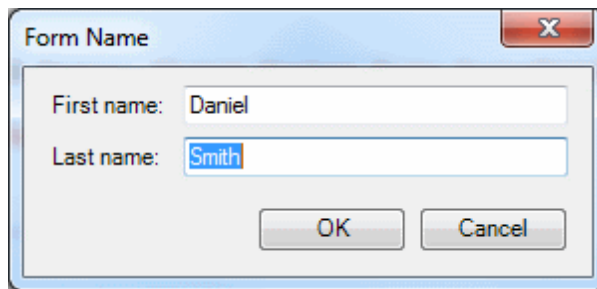
This global field can be used to conveniently access the last form object that was created.

Return Value: Returns a reference to the last form object (`System.Windows.Forms.Form`) that was successfully instantiated via `CreateForm()` or `ShowForm()`.

Example:

```
CreateForm( "FormName" );  
if ( lastform != null )  
{  
    lastform.textboxFirstName.Text = "Daniel";  
    lastform.textboxLastName.Text = "Smith";  
    var dialogResult = lastform.ShowDialog();  
}
```

Shows Form "FormName" as Dialog - TextBoxes are initialized (similar to the `CreateForm` example above):

**doevents()**

Processes all Windows messages currently in the message queue.

Return Value: None

Example:

```
for ( i=0; i < nLongLastingProcess; ++i )  
{  
    // do long lasting process  
  
    doevents(); // process windows messages; give UI a chance to update  
}
```

watchdog(bEnable : boolean)

Long running CPU-intensive scripts cause the watchdog to ask the user if the script should be terminated. The `watchdog()` method is used to disable or enable this behavior.

Per default the watchdog is enabled.

Return Value: None

Example:

```
watchdog( false ); // disable watchdog - we know the next statement is
CPU intensive but it will terminate for sure
doCPUIntensiveScript();
watchdog( true ); // re-enable watchdog
```

Usage tip:

Calling watchdog(true) can also be used to reset the watchdog. This can be useful before executing long running (CPU intensive) tasks to ensure they have the maximum allowed script processing quota.

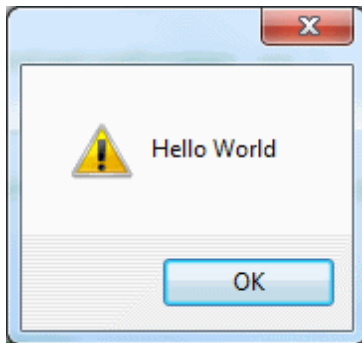
alert(strMessage : String) or MsgBox(strMessage : String)

An alert box is used to show a given message. The user will have to click "OK" to proceed.

Return Value: None

Example:

```
alert( "Hello World" );
```



confirm(strMessage : String)

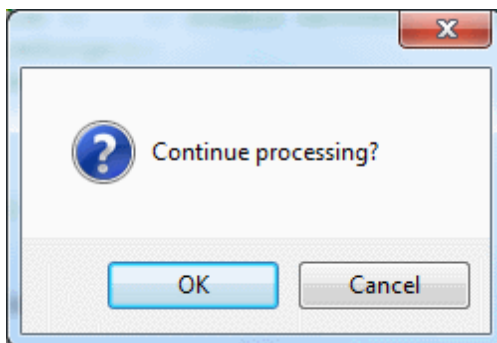
Opens a dialog that shows the given confirm message.

A confirm box is often used to verify or accept something. The user will have to click either "OK" or "Cancel" to proceed.

Return Value: A Boolean that represents the users answer. If the user clicks "OK", the dialog returns true, if the user clicks "Cancel", the dialog returns false.

Example:

```
if ( confirm( "Continue processing?" ) == false )
    return;
```



prompt(strMessage : String, strDefault : String)

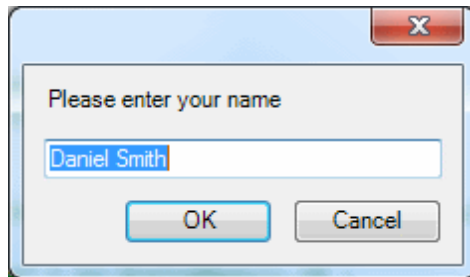
Opens a dialog that shows the given prompt message and a TextBox control with a default answer.

A prompt box is often used to input a simple string value.

Return Value: A String that contains the TextBox value or null if the user selected "Cancel".

Example:

```
var name = prompt( "Please enter your name", "Daniel Smith" );
if ( name != null )
    alert( "Hello " + name + "!" );
```



.NET interoperability commands

To allow further interoperability with the .NET Framework additional functions are provided under CLR.

CLR.Import(strNamespaceCLR : String)

This is the scripting equivalent to the C# *using* / VB.Net *imports* keyword. This allows to leave out the given namespaces in successive calls like `CLR.Create()` and `CLR.Static()`.

Return Value: None

Example:

Instead of always having to use full qualified names:

```
if ( ShowForm( "FormName" ) ==
CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
{
    var sName = lastform.textboxFirstName.Text + " " +
lastform.textboxLastName.Text;
    CLR.Static( "System.Windows.Forms.MessageBox" ).Show( "Hello " +
sName );
}
```

one can import namespaces and use the short form:

```
CLR.Import( "System.Windows.Forms" );
if ( ShowForm( "FormName" ) == CLR.Static( "DialogResult" ).OK )
{
    var sName = lastform.textboxFirstName.Text + " " +
lastform.textboxLastName.Text;
    CLR.Static( "MessageBox" ).Show( "Hello " + sName );
}
```

Please note:

Importing a namespace does not add or load the corresponding assembly to the scripting project! Assemblies can be added to the scripting project dynamically (at runtime) in the source code by using [CLR.LoadAssembly](#).

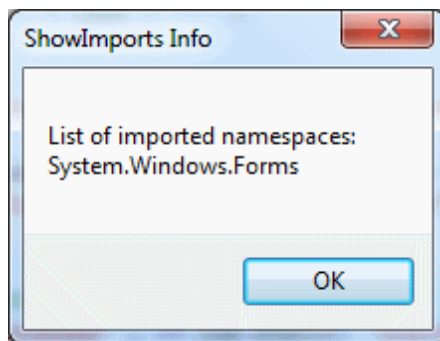
CLR.ShowImports()

Opens a MessageBox dialog that shows the currently imported namespaces. The user will have to click "OK" to proceed.

Return Value: None

Example:

```
CLR.ShowImports();
```

**CLR.LoadAssembly(strAssemblyNameCLR : String)**

Loads the .NET assembly with the given long assembly name or file path.

Return Value: A Boolean value. True if the assembly could be loaded, false otherwise.

Example:

```
// set clipboard text (if possible)
// System.Windows.Clipboard is part of the PresentationCore assembly, so
// load this assembly first:
if ( CLR.LoadAssembly( "PresentationCore, Version=3.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35", true ) )
{
    var clipboard = CLR.Static( "System.Windows.Clipboard" );
    if ( clipboard != null )
        clipboard.SetText( "HelloClipboard" );
}
```

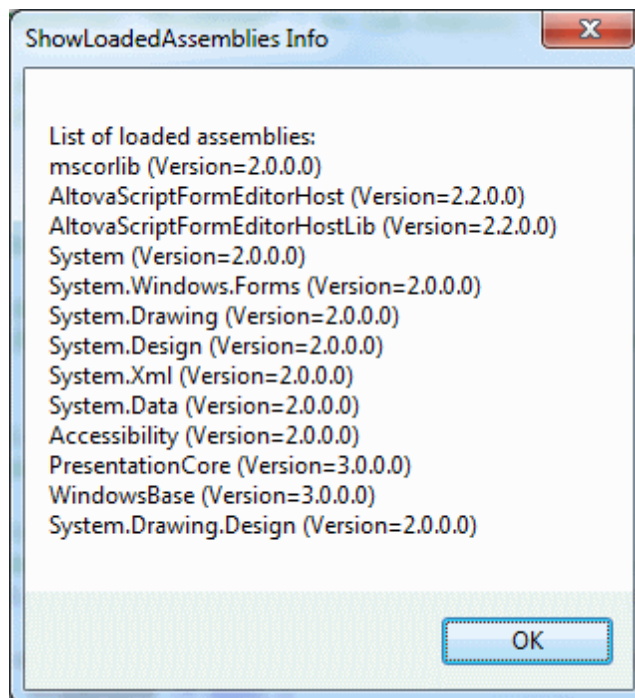
CLR.ShowLoadedAssemblies()

Opens a MessageBox dialog that shows the currently loaded assemblies. The user will have to click "OK" to proceed.

Return Value: None

Example:

```
CLR.ShowLoadedAssemblies();
```



CLR.Create(strTypeNameCLR : String, constructor arguments ...)

Creates a new .NET object instance for the given typename. If more than one argument is passed the successive arguments are interpreted as the arguments for the constructor of the .NET object.

Return Value: A reference to the created .NET object

Examples:

```
var objArray = CLR.Create("System.Collections.ArrayList");

var newItem = CLR.Create( "System.Windows.Forms.ListViewItem",
    "NewItemText" );

var coll =
CLR.Create( "System.Collections.Generic.List<System.String>" );

CLR.Import( "System" );
CLR.Import( "System.Collections.Generic" );
var dictionary = CLR.Create( "Dictionary< String, Dictionary< String,
String > >" );
```

CLR.Static(strTypeNameCLR : String)

Gives access to .NET types that have no instances and contain only static members.

Return Value: A reference to the static .NET object

Examples:

```
var enumValStretch =
CLR.Static( "System.Windows.Forms.ImageLayout" ).Stretch

var clipboard = CLR.Static( "System.Windows.Clipboard" );
clipboard.SetText( "HelloClipboard" );
```

```

if ( ShowForm( "FormName" ) ==
CLR.Static( "System.Windows.Forms.DialogResult" ).OK )
    alert( "ok" );
else
    alert( "cancel" );

```

Form usage and commands

Form usage is as follows:

With Form objects, the Form Component Tree can be accessed naturally via field access:

For example, suppose there is a Form designed as follows:

```

MyForm
    ButtonPanel
        OkButton
        CancelButton
    TextEditor
    AxMediaPlayer1

TrayComponents:
    MyTimer

```

The Form can then be instantiated from script as:

```
var objForm = CreateForm("MyForm");
```

To access one its components the field access can be used:

```
objForm.ButtonPanel.OkButton.Enabled = false;
```

or

```
objForm.TextEditor.Text = "Hello World";
```

To access Tray Components use the following method on the Form object:

```
var objTrayComponent = <A form object>.GetTrayComponent(strComponentName :
String);
```

In our example to get a reference to the Timer Component to enable it use the following:

```
var objTimer = objForm.GetTrayComponent("MyTimer");
objTimer.Enabled = true;
```

For ActiveX Controls the underlying COM object can be accessed via the OCX property:

```

var ocx = lastform.AxMediaPlayer1.OCX; // get underlying COM object
ocx.enableContextMenu = true;
ocx.URL = "mms://apasf.apa.at/fm4_live_worldwide";

```

20.2.8 Usage example (UModel Find sample)

This scripting example is available in the ...\UModelExamples folder as **ScriptSampleFind.asprj**.

Global Declarations:

```
// import System.Windows.Forms namespace for all macros, forms and events:
CLR.Import( "System.Windows.Forms" );
```

Form FindSample

```
//Occurs whenever the user loads the form.
function FindKinds_Load( objSender, e_EventArgs )
{
    // initialize all controls:
    objSender.btnFind.Enabled = false;

    objSender.gbOptions.cbMatchCase.Checked = bMatchCase;
    objSender.gbOptions.cbWholeName.Checked = bMatchWholeName;

    objSender.gbSearchKinds.cbClasses.Checked =
bSearchClasses;
    objSender.gbSearchKinds.cbInterfaces.Checked =
bSearchInterfaces;
    objSender.gbSearchKinds.cbEnumerations.Checked =
bSearchEnumerations;
    objSender.gbSearchKinds.cbPackages.Checked =
bSearchPackages;
    objSender.gbSearchKinds.cbProperties.Checked =
bSearchProperties;
    objSender.gbSearchKinds.cbOperations.Checked =
bSearchOperations;
    objSender.gbSearchKinds.cbEnumerationLiterals.Checked =
bSearchEnumLiterals;
    objSender.gbSearchKinds.cbOthers.Checked =
bSearchOthers;
}

//Event raised when the value of the Text property is changed on Control.
function FindKinds_tbFindText_TextChanged( objSender, e_EventArgs )
{
    // objSender is the 'Find' TextBox; also get the form:
    var objForm = objSender.FindForm();

    // to enable the find button, searchstring should at least have one
character
    objForm.btnFind.Enabled = ( objSender.Text.length > 0 );
}

//Occurs when the component is clicked.
function FindType_btnFind_Click( objSender, e_EventArgs )
{
    // objSender is the 'Find' Button; also get the form:
```

```

var objForm = objSender.FindForm();

// save chosen settings in global macro variables:
bMatchCase      = objForm.gbOptions.cbMatchCase.Checked;
bMatchWholeName  = objForm.gbOptions.cbWholeName.Checked;

bSearchClasses   = objForm.gbSearchKinds.cbClasses.Checked;
bSearchInterfaces = objForm.gbSearchKinds.cbInterfaces.Checked;
bSearchEnumerations = objForm.gbSearchKinds.cbEnumerations.Checked;
bSearchPackages  = objForm.gbSearchKinds.cbPackages.Checked;
bSearchProperties = objForm.gbSearchKinds.cbProperties.Checked;
bSearchOperations = objForm.gbSearchKinds.cbOperations.Checked;
bSearchEnumLiterals =
objForm.gbSearchKinds.cbEnumerationLiterals.Checked;
bSearchOthers     = objForm.gbSearchKinds.cbOthers.Checked;

var sFindText = objForm.tbFindText.Text;

if ( !bMatchCase )
    sFindText = sFindText.toUpperCase();

// show wait cursor during search; load static "Cursors" class:
var objCursors = CLR.Static( "Cursors" );
objForm.Cursor = objCursors.WaitCursor;
// start update of ListView:
objForm.lvResults.BeginUpdate();
// reset Items:
objForm.lvResults.Items.Clear();
// start find with root-package:
Find( Application.ActiveDocument.RootPackage, sFindText );
// ListView has been updated:
objForm.lvResults.EndUpdate();
// reset cursor to default:
objForm.Cursor = objCursors.Default;

// inform user when nothing has been found:
if ( objForm.lvResults.Items.Count == 0 )
    alert( "Nothing found!" );
}

//Occurs whenever the 'SelectedIndex' property for this ListView changes.
function FindKinds_lvResults_SelectedIndexChanged( objSender, e_EventArgs )
{
    // focus selected UML element in Model Tree:
    if ( objSender.SelectedItems.Count == 1 )
    {
        // get qualified name of selected item:
        var selItem = objSender.SelectedItems.Item( 0 );
        var sQualified_name = selItem.SubItems.Item( 2 ).Text;

        var objDocument = Application.ActiveDocument;
        // find UML element by help of the qualified name:
        var objUMLElement =

```

```

objDocument.RootPackage.FindOwnedMemberWithQualifiedName( sQualifiedName );
    // focus UML element in Model Tree:
    objDocument.FocusUMLDataInModelTree( objUMLElement, false, true );
}
}

```

Macro Find Sample:

```

// #####
// 'Find Sample' macro
// Find UML model elements by name and kind
// #####

// global variables of macro
var bMatchCase           = false;
var bMatchWholeName      = false;

var bSearchClasses       = true;
var bSearchInterfaces     = true;
var bSearchEnumerations  = true;
var bSearchPackages      = true;
var bSearchProperties     = true;
var bSearchOperations     = true;
var bSearchEnumLiterals  = true;
var bSearchOthers        = false;

// verify if UML element kind should be considered
function MatchKind( objUMLElement )
{
    if ( objUMLElement.KindName == "Class" ) return
bSearchClasses;
    if ( objUMLElement.KindName == "Interface" ) return
bSearchInterfaces;
    if ( objUMLElement.KindName == "Enumeration" ) return
bSearchEnumerations;
    if ( objUMLElement.KindName == "Package" ) return
bSearchPackages;
    if ( objUMLElement.KindName == "Property" ) return
bSearchProperties;
    if ( objUMLElement.KindName == "Operation" ) return
bSearchOperations;
    if ( objUMLElement.KindName == "EnumerationLiteral" ) return
bSearchEnumLiterals;
    return bSearchOthers;
}

// verify if UML element name matches search options
function MatchName( objUMLNamedElement, sFindText )
{
    var sName = objUMLNamedElement.Name;
    if ( !bMatchCase )
        sName = sName.toUpperCase();

    if ( bMatchWholeName )

```

```
        return ( sName == sFindText )
    else
        return ( sName.indexOf( sFindText ) != -1 )
    }

    // recursively search all elements
    function Find( objUMLElement, sFindText )
    {
        // we are only interested in named elements:
        if ( objUMLElement.IsKindOf( "NamedElement" ) )
        {
            var objUMLNamedElement = objUMLElement;

            // only list elements where kind and name match:
            if ( MatchKind( objUMLNamedElement ) &&
                MatchName( objUMLNamedElement, sFindText ) )
            {
                // create a new ListViewItem class
                var newItem = CLR.Create( "ListViewItem",
objUMLNamedElement.KindName );
                // and add it to the listview - items:
                lastForm.lvResults.Items.Add( newItem );
                // append name and qualified name in sub-items:
                newItem.SubItems.Add( objUMLNamedElement.Name );
                newItem.SubItems.Add( objUMLNamedElement.QualifiedName );
            }
        }

        // also iterate over all children (ownedElements):
        var ownedElements = objUMLElement.OwnedElements;
        var itr = new Enumerator( ownedElements );
        for ( ; !itr.atEnd(); itr.moveNext() )
            Find( itr.item(), sFindText );
    }

    // create and show the form:
    ShowForm( "FindSample" );
```


20.3 IDE PlugIns

UModel allows you to create your own plug-ins and integrate them into UModel.

Use plug-ins to:

- Configure your version of UModel, add commands through menus, icons, buttons etc.
- React to events from UModel.
- Run your specific code within UModel with access to the complete API
- integrate your own ActiveX controls into UModel

UModel expects your plug-in to implement the [IUModelPlugIn](#) interface.

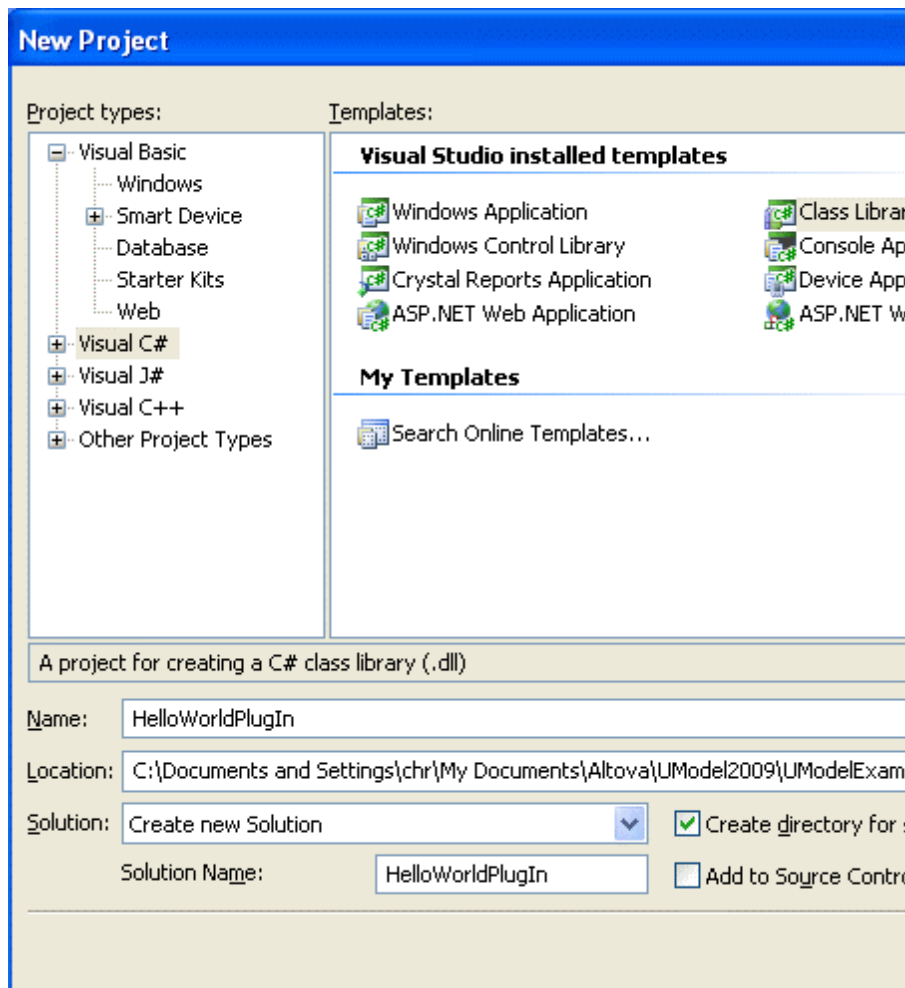
A simple implementation of this interface can be found [here](#), or see the folder "UModelExamples\UModelAPI" of your UModel installation for several further samples using C#.

20.3.1 How to create a plug-in in MS Visual Studio .NET

This section shows how to create a simple UModel IDE plug-in DLL using C#.

Having installed UModel and Microsoft Visual Studio .NET on your computer:

1. Start MS Visual Studio .NET and create a new, empty project in the coding language of your choice (C# in this sample).
Choose the Class Library template to create a class library (.dll).

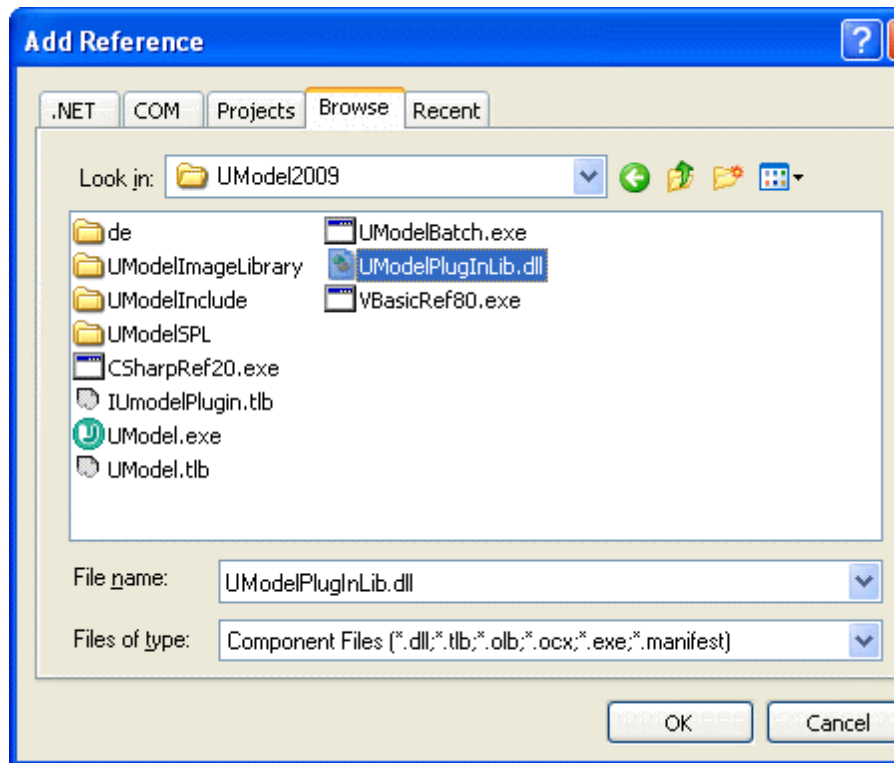


Add reference to plugin library

Add a reference to the UModel plug-in library

If a DLL is added to UModel as a plug-in, it is necessary that it implements the [IUModelPlugIn](#) interface, which is defined in UModelPlugInLib.dll.

1. Right-click "**References**" in the Solution Explorer.
2. Select the "**Browse**" tab and browse to the UModel installation directory (e.g. C:\Program Files\Altova\UModel2010).
3. Select **UModelPlugInLib.dll**:



UModelPlugInLib.dll is a .NET assembly and has been created from IUModelPlugIn.tlb. You can also create your own assembly using the type library importer of your choice.

Note:

The UModelPlugInLib.dll which is installed, was generated using the Microsoft .NET Framework v2.0. If you plan to install your plug-in on a .NET Framework prior to 2.0 (e.g. 1.1), you must create your own UModelPlugInLib.dll.

How to create your own UModelPlugInLib.dll

You can create your own UModelPlugInLib.dll by using e.g. the type library importer of the Microsoft .NET Framework SDK:

```
tlbimp.exe IUModelPlugIn.tlb
```

or to create the assembly with a strong name key pair and a specific version:

```
tlbimp.exe IUModelPlugIn.tlb /keyfile:UModelPlugIn.snk /  
asmversion:1.0.0.0
```

where UModelPlugIn.snk is a key file created by sn.exe (e.g. "sn.exe -k UModelPlugIn.snk")

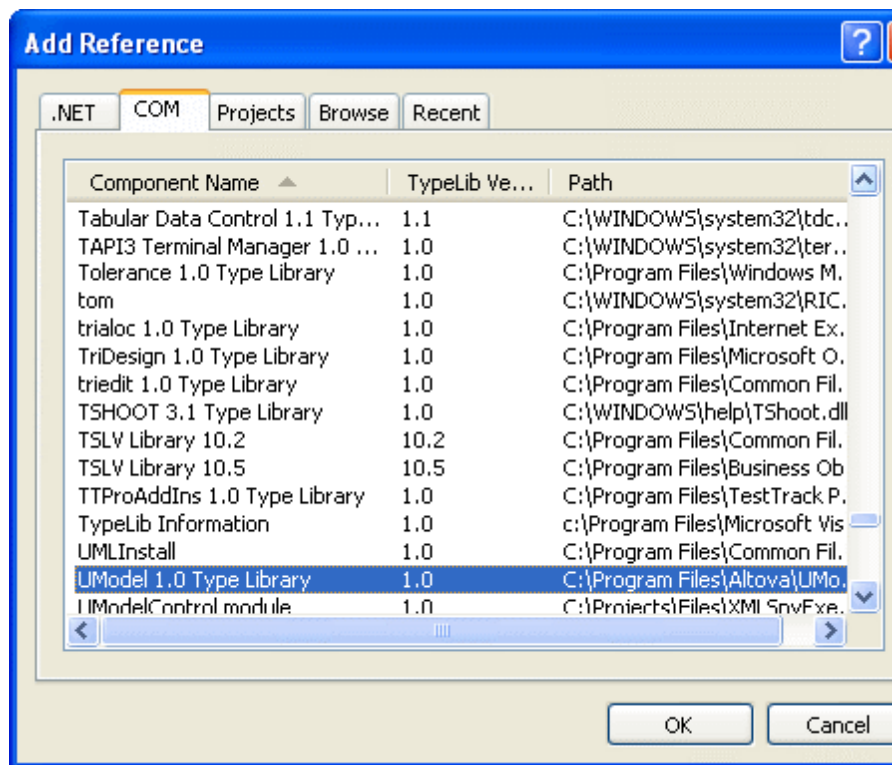
Note:

sn.exe is part of the Microsoft .NET Framework SDK

Add reference to UModel type library

To use all UModel API interfaces, you need to add a reference to the UModel type library:

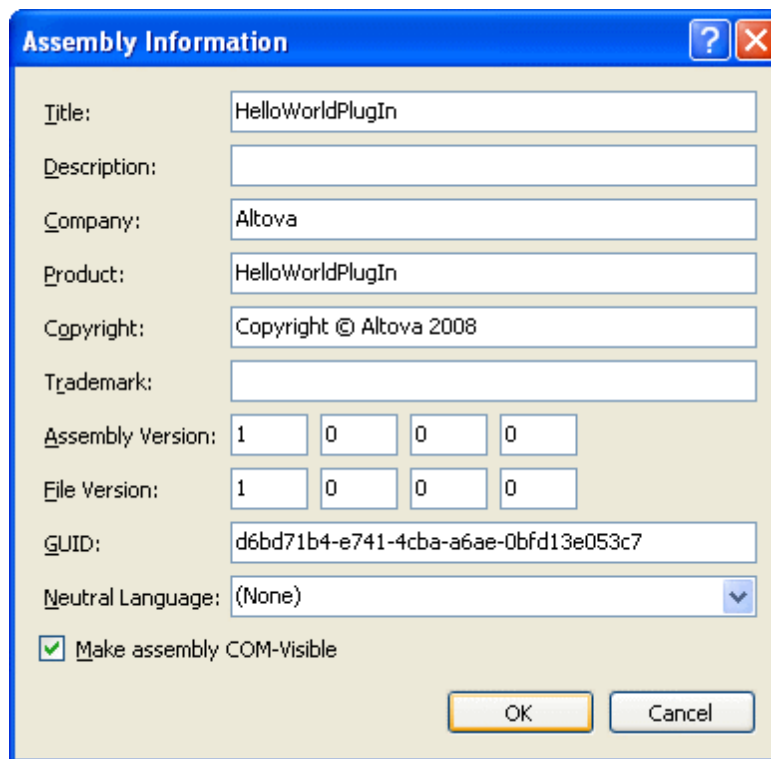
1. Right-click **"References"** in the Solution Explorer
2. Select the **"COM"** tab and choose the UModel type library:



Change compiler settings - make COM-visible

To make you code accessible to COM, you need to change your compiler settings.

1. Right click your C# project and select **"Properties"**.
2. On the "Application" tab, open the **"Assembly Information..."** dialog box and check the **"Make assembly COM-Visible"** check box at the bottom of the dialog box.



The "Assembly Information" dialog box is shown with the following fields and values:

- Title: HelloWorldPlugIn
- Description: (empty)
- Company: Altova
- Product: HelloWorldPlugIn
- Copyright: Copyright © Altova 2008
- Trademark: (empty)
- Assembly Version: 1.0.0.0
- File Version: 1.0.0.0
- GUID: d6bd71b4-e741-4cba-a6ae-0bfd13e053c7
- Neutral Language: (None)
- ☒ Make assembly COM-Visible

Buttons: OK, Cancel

Change compiler settings - expose COM wrapper

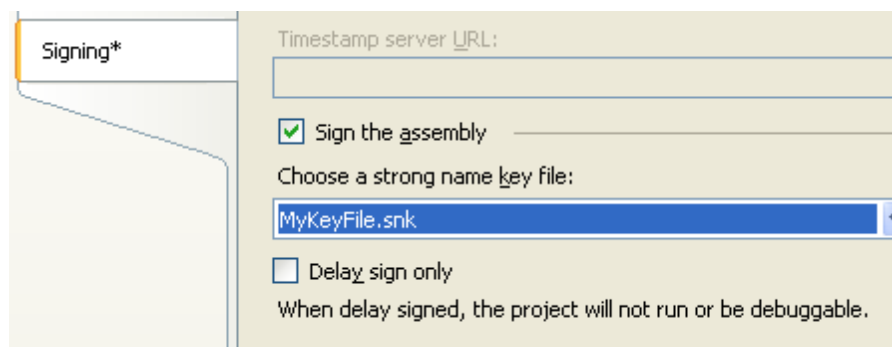
To expose a COM callable wrapper that can interact with COM objects:

1. Right click your C# project and select **"Properties"** (if properties are not still open from the previous step).
2. In the "Build" tab, check the **"Register for COM interop"** check box for all build configurations.

How to sign your plug-in with a strong name (optional)

To sign your assembly with a strong name key pair (e.g. for [deployment](#)):

1. Right click your C# project and select **"Properties"** (if properties are not still open from the previous step).
2. On the "Signing" tab, check **"Sign the assembly"** and use either "Browse..." to choose an existing key file or "New..." to create a new one.



The "Signing*" dialog box is shown with the following fields and values:

- Timestamp server URL: (empty)
- ☒ Sign the assembly
- Choose a strong name key file: MyKeyFile.snk
- ☐ Delay sign only
- When delay signed, the project will not run or be debuggable.

Implement IUModelPlugIn

UModel IDE plug-ins must implement the [IUModelPlugIn](#) interface.

The code below shows a simple implementation of this interface. It adds a menu item and a separator (available with UModel) to the Edit menu.

Note:

Inside the OnUpdateCommand() method, the command "new" is always enabled. The command opens a simple message box (ensure that your C# project also references System.Windows.Forms).

```
using System;
using System.Collections.Generic;
using System.Text;
using UModelPlugInLib;

namespace HelloWorldPlugIn
{
    public class MyHelloWorldUModelPlugIn : IUModelPlugIn
    {
        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "HelloWorldPlugIn;HelloWorldPlugIn demonstrates a
simple implementation of an IDE plug-in for UModel";
        }

        public string GetUIModifications()
        {
            return "<ConfigurationData>" +
                "<Modifications>" +
                // add "Hello World..." to Edit menu
                "<Modification>" +
                "<Action>Add</Action>" +
                "<UIElement type=\"MenuItem\">" +
                "<ID>1</ID>" +
                "<Name>Hello world...</Name>" +
                "<Info>My hello world</Info>" +
                "<Place>0</Place>" +
                "<MenuID>101</MenuID>" +
                "<Parent>:Edit</Parent>" +
                "</UIElement>" +
                "</Modification>" +
                // add Separator to Edit menu
                "<Modification>" +
                "<Action>Add</Action>" +
                "<UIElement type=\"MenuItem\">" +
                "<ID>0</ID>" +
                "<Place>1</Place>" +
                "<MenuID>101</MenuID>" +
                "<Parent>:Edit</Parent>" +
```

```

        "</UIElement>"
        "</Modification>"
        // finish modification description
        "</Modifications>"
        "</ConfigurationData>";
    }

    public void OnInitialize(object pUModel)
    {
        // before processing DDE or batch commands
    }

    public void OnRunning(object pUModel)
    {
        // DDE or batch commands are processed; application is
        fully initialized
    }

    public void OnShutdown(object pUModel)
    {
        // application will shutdown; release all unused objects
    }

    public UModelUpdateAction OnUpdateCommand(int nID, object
pUModel)
    {
        if (nID == 1)
            return UModelUpdateAction.UModelUpdateAction_Enable;

        return UModelUpdateAction.UModelUpdateAction_Disable;
    }

    public void OnCommand(int nID, object pUModel)
    {
        System.Windows.Forms.MessageBox.Show("Hello world!");
    }

    #endregion
}
}

```

Please note:

Microsoft Visual Studio versions prior to version 2005 generate new GUIDs automatically when compiling. To always use the same GUID for your IUModelPlugIn implementation, you should define the GUID by using the GuidAttribute class of the System.Runtime.InteropServices namespace, e.g.

```

[ComVisible(true), GuidAttribute("332B1F8B-4415-44d7-BB6F-
176E2A52DFBD")]
public class MyHelloWorldUModelPlugIn : IUModelPlugIn
{
    ...
}

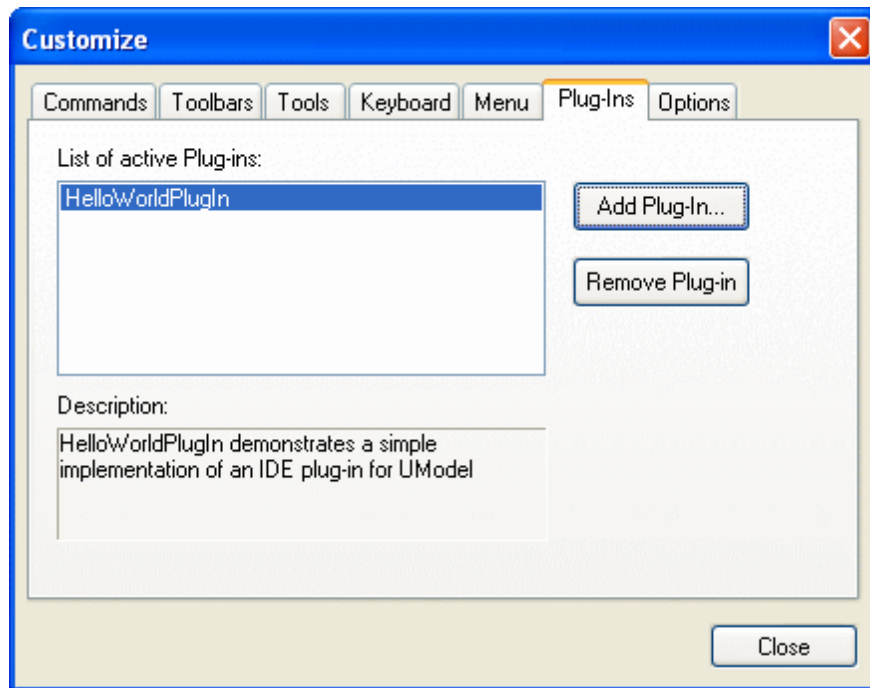
```

Test UModel plug-in

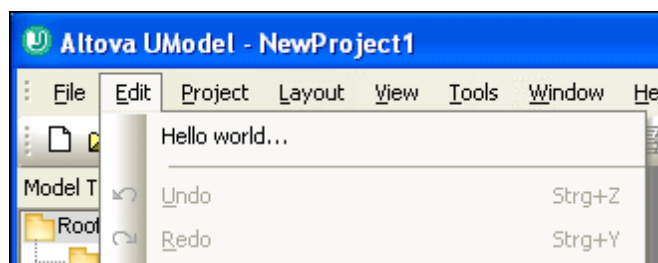
After building your C# project you can test the new UModel plug-in.

Note that there is no additional need to register/install the plug-in before testing:

1. Start UModel and choose "**Tools | Customize...**".
2. Click the "Add Plug-In..." button in the Plug-Ins" tab, to specify the DLL that implements your plug-in (HelloWorldPlugIn.dll):



3. Close the customize dialog and verify that the "Edit" menu has been changed according to the specified UI modification (`GetUIModifications()`):



4. Check that the command is executed as specified by the plug-in:



20.3.2 Deployment of PlugIns

Deployment Prerequisites:

Note that the plug-in is automatically registered on the development machine during the build process. There is also no need to deploy the plug-in.

To successfully install your plug-in on your **target system/deployment PC**, you need the following:

- UModel Professional or Enterprise edition
- If your plug-in is written in .NET, the corresponding Microsoft .NET Framework

Deploying UModel IDE plug-ins using an installer

UModel IDE plug-ins can be deployed by creating an installer.

The [SetupStyles](#) sample shows how to create a setup project in order to install and register an IDE plug-in on your target system.

Some hints for manual installation/registration:

Hints for registering UModelPlugInLib.dll

Register the assembly for COM:

```
regasm.exe UModelPlugInLib.dll /codebase
```

and install it to the global assembly cache (GAC):

```
gacutil.exe /i UModelPlugInLib.dll
```

Notes:

regasm.exe and gacutil.exe are part of the Microsoft .NET Framework SDK.

The UModelPlugInLib.dll which is installed, was generated using the Microsoft .NET Framework v2.0. If you plan to install your plug-in on a .NET Framework prior to 2.0 (e.g. 1.1), you must create your own UModelPlugInLib.dll (also see [How to create your own UModelPlugInLib.dll](#))

Hints for deploying your UModel IDE plug-in written in .NET

Ensure UModelPlugInLib.dll is registered (see above)!

- **How to sign your assembly (plug-in) with a strong name key pair**
You can either sign your assembly by using the assembly linker (this is recommended! - also see [How to sign your plug-in with a strong name](#)), or by using the AssemblyKeyFileAttribute:

create a key file, e.g.

```
sn.exe -k MyKeyFile.snk
```

and add following line to the assembly file (e.g. assembly.cs) of your project:

```
[assembly: AssemblyKeyFile("MyKeyFile.snk")]
```

- **How to register your assembly (plug-in) for COM**

```
regasm.exe MyPlugin.dll /codebase /tlb
```

- **How to install your assembly (plug-in) to the global assembly cache (GAC)**

```
gacutil.exe /i MyPlugin.dll
```

Notes:

sn.exe, regasm.exe and gacutil.exe are part of the Microsoft .NET Framework SDK

When deploying your UModel IDE plug-in on .NET framework versions prior to 2.0, the plug-in DLL must either be installed in the same directory as UModel.exe or signed with a strong name key and registered into the global assembly cache (GAC).

Registration of plug-ins in UModel

UModel maintains a specific key in the Registry where it stores all registered plug-ins:

```
HKEY_CURRENT_USER\Software\Altova\UModel\PlugIns
```

All values of this key are treated as references to registered plug-ins and must conform to the following format:

Value name:	ProgID of the plug-in
Value type:	must be REG_SZ
Value data:	CLSID of the component

Every time the application starts the values of the "PlugIns" key are scanned, and the registered plug-ins are loaded.

- **Register the plug-in manually**

To register a plug-in manually, use the "Customize" dialog box of the UModel "Tools" menu. Use the "Add Plug-In..." button to specify the DLL that implements your plug-in. UModel registers the DLL as a COM server and adds the corresponding entry in its "PlugIns" key.

If you experience problems with manual registration, you can check if the CLSID of your plug-in is correctly registered in the "PlugIns" key. If this is not the case, the name of your plug-in DLL was probably not sufficiently unique. Use a different name or perform a direct registration.

A sample of how to manually register a plug-in can be found [here](#).

- **Register plug-in directly**

A plug-in can be directly registered as an plug-in by first registering the DLL and then adding the appropriate value to the "PlugIns" key of UModel during plug-in setup. The new plug-in will be activated the next time UModel is launched.

20.3.3 Configuration XML

The plug-in allows you to change the user interface (UI) of UModel. This is done by describing each separate modification using an XML data stream. The XML configuration is passed to UModel using the [GetUIModifications](#) method of the IUModelPlugIn interface.

The XML file containing the UI modifications for the plug-in must have the following structure:

```
<ConfigurationData>
  <ImageFile>path To image file</ImageFile>
  <Modifications>
    <Modification>
      ...
    </Modification>
    ...
  </Modifications>
</ConfigurationData>
```

You can define icons, or toolbar buttons for the new menu items which are added to the UI of UModel by the plug-in. The path to the file containing the images is set using the **ImageFile** element. Each image must be 16 x 16 pixels. The image references must be arranged from left to right in a single (<ImageFile>...) line. The rightmost image index value, is zero.

The **Modifications** element can have any number of **Modification** child elements. Each **Modification** element defines a specific change to the standard UI of UModel. It is also possible to remove UI elements from UModel.

Structure of Modification elements

All Modification elements consist of the following two child elements:

```
<Modification>
  <Action>Type of action</Action>
  <UIElement Type="type of UI element">
  </UIElement>
</Modification>
```

Valid values for the **Action** element are:

- Add - to add the following UI element to UModel.
- Hide - to hide the following UI element in UModel.
- Remove - to remove the UI element from the "Commands" list box, in the customize dialog

You can combine values of the **Action** element e.g. "Hide Remove"

The **UIElement** element describes any new, or existing UI element for UModel. Possible elements are currently: new toolbars, buttons, menus, or menu items. The **type** attribute, defines which UI element is described by the XML element.

Common UIElement children

The ID and Name elements are valid for all different types of XML UIElement fragments. It is however, possible to ignore one of the values for a specific type of UIElement e.g. Name is ignored for a separator.

```
<ID></ID>
```

```
<Name></Name>
```

If **UIElement** describes an existing element of the UI, the value of the ID element is predefined by UModel. Normally these ID values are not known to the public. If the XML fragment describes a new part of the UI, then the ID is arbitrary and the value should be less than 1000.

The **Name** element sets the textual value. Existing UI elements can be identified just by name, for e.g. menus and menu items with associated sub menus. For new UI elements, the **Name** element sets the caption e.g. the title of a toolbar, or text for a menu item.

Toolbars and Menus

To define a toolbar its necessary to specify the ID and/or the name of the toolbar. An existing toolbar can be specified using only the name, or by the ID if it is known. To create a **new** toolbar both values must be set. The **type** attribute must be equal to "ToolBar".

```
<UIElement Type="ToolBar">
  <ID>1</ID>
  <Name>Styles</Name>
</UIElement>
```

To specify an UModel menu you need two parameters:

- The ID of the menu bar which contains the menu. UModel's main menu bar ID is 101.
- The menu name. Menus do not have an associated ID value. The following example defines the "Edit" menu of the menu bar:

```
<UIElement Type="Menu">
  <ID>101</ID>
  <Name>Edit</Name>
</UIElement>
```

An additional element is used if you want to create a new menu. The **Place** element defines the position of the new menu in the menu bar:

```
<UIElement Type="Menu">
  <ID>101</ID>
  <Name>PlugIn Menu</Name>
  <Place>12</Place>
</UIElement>
```

A value of -1 for the **Place** element sets the new button or menu item at the end of the menu or toolbar.

Commands

If you add a new command, through a toolbar button or a menu item, the **UIElement** fragment can contain any of these sub elements:

```
<Info></Info>
<ImageID></ImageID>
```

The **Info** element contains a short description string which is displayed in the status bar, when the mouse pointer is over the associated command (button or menu item). **ImageID** defines the index of the icon in the external image file. Please note that all icons are stored in one image file.

To define a toolbar button create an **UIElement** with this structure:

```

<UIElement Type="ToolBarItem">
  <!--don't reuse local IDs even the commands do the same-->
  <ID>6</ID>
  <Name>Fill red</Name>
  <!--Set Place To -1 If this is the first button to be inserted-->
  <Place>-1</Place>
  <ImageID>0</ImageID>
  <ToolBarID>1</ToolBarID>
  <!--instead of the toolbar ID the toolbar name could be used-->
  <ToolBarName>Styles</ToolBarName>
</UIElement>

```

Additional elements to declare a toolbar button are **Place**, **ToolBarID** and **ToolBarName**. **ToolBarID** and **ToolBarName** are used to identify the toolbar which contains the new or existing button. The textual value of **ToolBarName** is case sensitive. The (UIElement) **type** attribute must equal "ToolBarItem".

To define a menu item, the elements **MenuID**, **Place** and **Parent** are available in addition to the standard elements used to declare a command. **MenuID** must be 101. Please see "Toolbars and Menus" for more information on these values.

The **Parent** element is used to identify the **menu** where the new menu entry should be inserted. As sub menu items have no unique Windows ID, we need some other way to identify the parent of the menu item.

The value of the **Parent** element is a path to the menu item.

The text value of the Parent element, must equal the **parent menu name** of the submenu, where the submenu name is separated by a colon. If the menu has no parent, because it is not a submenu, add a colon to the beginning of the name. The **type** attribute must be set to "MenuItem".

Example for an **UIElement** defining a menu item:

```

<UIElement Type="MenuItem">
  <!--the following element is a Local command ID-->
  <ID>3</ID>
  <Name>Fill red</Name>
  <Place>-1</Place>
  <MenuID>101</MenuID>
  <Parent>:PlugIn Menu1</Parent>
  <ImageID>0</ImageID>
</UIElement>

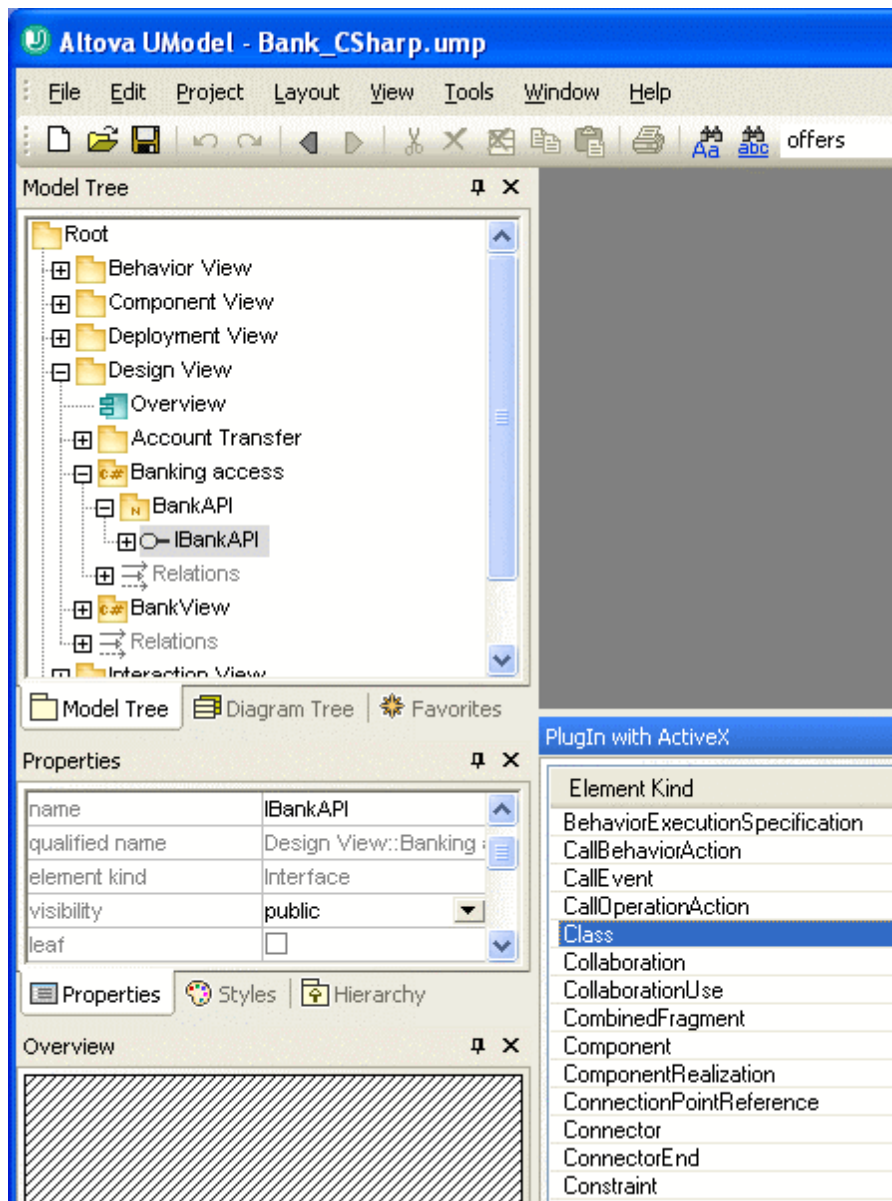
```

UModel makes it possible to add toolbar separators and menus if the value of the **ID** element is set to 0.

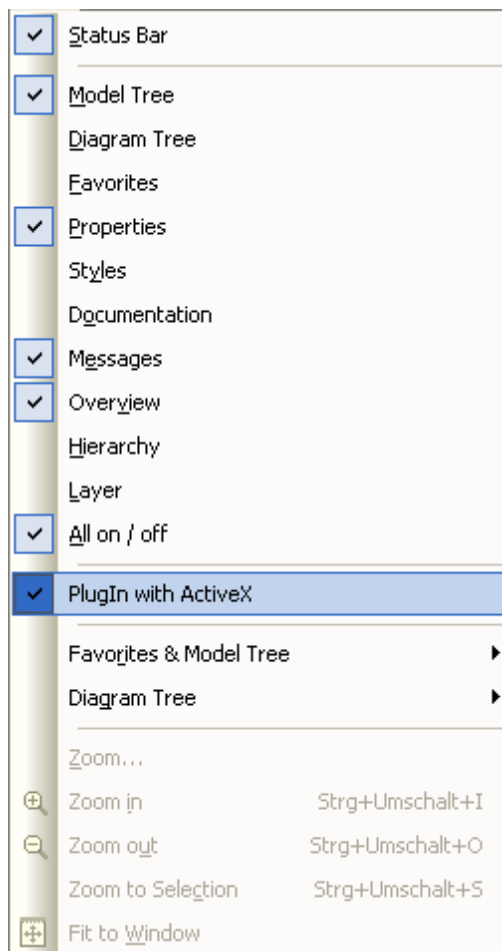
20.3.4 ActiveX Controls

Any IDE plug-in which is also an ActiveX control will be displayed in a Dialog Control Bar. To work as ActiveX control, the IDE plug-in must e.g. implement the `IOleControl` interface (C++) or derive from `System.Windows.Forms.UserControl` (C#, VB.NET).

Also see the [StatisticsActiveX](#) sample:



Every ActiveX IDE plug-in gets a new menu command in the 'View' menu to show/hide the corresponding Dialog Control Bar (pane):

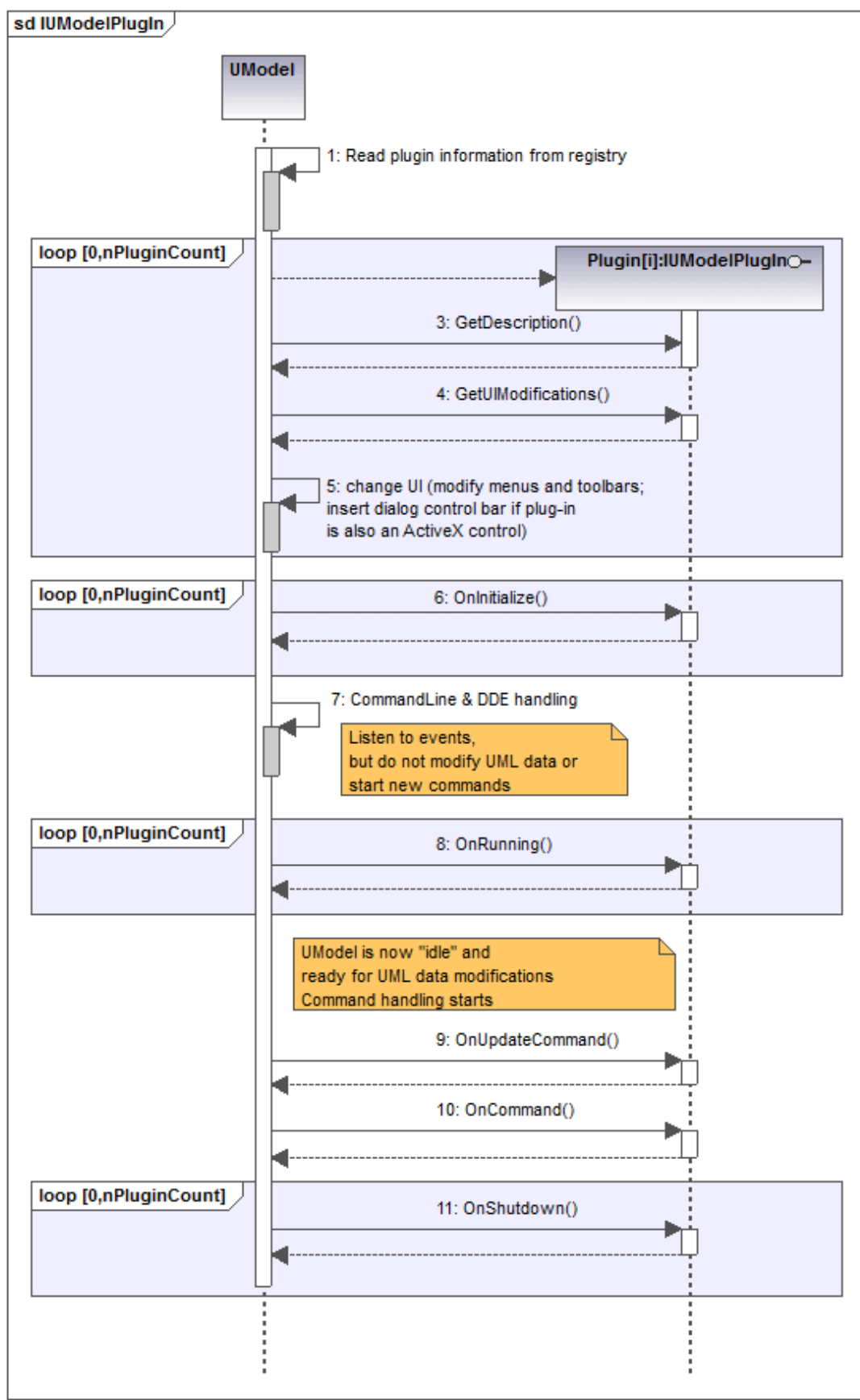


20.3.5 IModelPlugin

Methods[OnInititalize](#)[OnRunning](#)[OnShutdown](#)[GetUIModifications](#)[GetDescription](#)[OnCommand](#)[OnUpdateCommand](#)**Description**

If a DLL is added to UModel as a plug-in, it is necessary that it registers a COM component that answers to an [IModelPlugin](#) interface with the reserved UUID (6B893B43-BEDD-4eea-B6D2-C0F3A6402119), for it to be recognized as a plug-in.

The sequence diagram below shows how UModel interacts with [IModelPlugin\(s\)](#):



OnInitialize

Declaration: `OnInitialize(pUModel as IDispatch)`

Description

The `OnInitialize` method of the interface implementation is called when the plug-in is initialized and **before** DDE or batch commands are processed.

You can attach notifiers and listen to UModel events, but should not start new commands / modifications until the `OnRunning` method is called.

`pUModel` holds a reference to the dispatch interface of the `Application` object of UModel.

Example

```
public void IUModelPlugIn_OnInitialize(object pUModel)
{
    // before processing DDE or batch commands
}
```

OnRunning

Declaration: `OnRunning(pUModel as IDispatch)`

Description

The `OnRunning` method of the interface implementation is called when the plug-in is initialized and **after** DDE or batch commands are processed.

The application is now fully initialized and you can start new commands / modifications and modify UML data.

`pUModel` holds a reference to the dispatch interface of the `Application` object of UModel.

Example

```
public void IUModelPlugIn_OnRunning(object pUModel)
{
    // DDE or batch commands are processed; application is fully
    // initialized
    // and we can attach UModelApplicationEvents
    AttachAppEvents( (IApplication)pUModel );
}
```

OnShutdown

Declaration: `OnShutdown(pUModel as IDispatch)`

Description

The `OnShutdown` method of the interface implementation is called immediately before the plug-in is unloaded (e.g. because the application will shutdown).

`pUModel` holds a reference to the dispatch interface of the `Application` object of UModel.

Example

```
public void IUModelPlugIn_OnShutdown(object pUModel)
```

```

{
    // detach UModelApplicationEvents; stop receiving events
    DetachAppEvents();

    // application will shutdown; release all unused objects
    GC.Collect();
}

```

GetUIModifications

Declaration: `GetUIModifications()` as `String`

Description

The `GetUIModifications()` method is called during initialization of the plug-in, to get the configuration XML data that defines the changes to the UI of UModel.

The method is called when the plug-in is loaded for the first time, and at every start of UModel.

See also [Configuration XML](#) for a detailed description on how to change the UI.

Example

```

protected string GetPlugInPath()
{
    string sDLLPath =
System.Reflection.Assembly.GetExecutingAssembly().Location;
    return System.IO.Path.GetDirectoryName(sDLLPath);
}

public string IUModelPlugIn_GetUIModifications()
{
    try
    {
        string sPath = GetPlugInPath();

        System.IO.StreamReader myFile = new
System.IO.StreamReader(sPath + "\\config.xml");
        string sRet = myFile.ReadToEnd();
        myFile.Close();

        // this replaces the token "***path**" from the XML file
with
        // the actual installation path of the plug-in to get the
image file
        return sRet.Replace("***path**", sPath);
    }
    catch (System.Exception ex)
    {
        System.Windows.Forms.MessageBox.Show("Error in
GetUIModifications:" + ex.Message);
        throw ex;
    }
}

```

GetDescription

Declaration: `GetDescription()` as `String`

Description

`GetDescription()` is used to define the description string for the plug-in entries visible in the Customize dialog box.

Example

```
public string IUModelPlugIn_GetDescription()
{
    return "Styles sample Plug-in for UModel;This Plug-in
demonstrates how to change several styles of the selected diagram
elements.";
}
```

OnCommand

Declaration: `OnCommand(nID as long, pUModel as IDispatch)`

Description

The `OnCommand()` method of the interface implementation, is called each time a command, added by the plug-in (menu item or toolbar button), is processed.

`nID` stores the command ID defined by the `ID` element of the respective `UIElement`.

`pUModel` holds a reference to the dispatch interface of the `Application` object of `UModel`.

Example

```
public void IUModelPlugIn_OnCommand(int nID, object pUModel)
{
    // fill red
    if (nID == 3 || nID == 6)
        OnSetStyles((IApplication)pUModel, "red");

    // fill green
    if (nID == 4 || nID == 7)
        OnSetStyles((IApplication)pUModel, "green");

    // release unused objects
    GC.Collect();
}
```

OnUpdateCommand

Declaration: `OnUpdateCommand(nID as long, pUModel as IDispatch)` as `UModelUpdateAction`

Description

The `OnUpdateCommand()` method is called each time the visible state of a button, or menu item, needs to be set.

`nID` stores the command ID defined by the `ID` element of the respective `UIElement`.

`pUModel` holds a reference to the dispatch interface of the `Application` object.

Possible return values (as defined in [UModelUpdateAction](#)) to set the update state are:

<code>UModelUpdateAction_Enable</code>	<code>= 1</code>
<code>UModelUpdateAction_Disable</code>	<code>= 2</code>
<code>UModelUpdateAction_Check</code>	<code>= 4</code>
<code>UModelUpdateAction_Uncheck</code>	<code>= 8</code>

Values can be combined using the bitwise OR operator (e.g. `UModelUpdateAction_Enable | UModelUpdateAction_Check`).

Example

```
public UModelUpdateAction IUModelPlugIn_OnUpdateCommand(int nID, object
pUModel)
{
    UModelUpdateAction action =
    UModelUpdateAction.UModelUpdateAction_Disable;

    // check for "fill red"
    if (nID == 3 || nID == 6)
        action = OnUpdateSetStyles((IApplication)pUModel);

    // check for "fill green"
    if (nID == 4 || nID == 7)
        action = OnUpdateSetStyles((IApplication)pUModel);

    // release unused objects
    GC.Collect();

    return action;
}
```

20.4 The UModel API

The COM-based API of UModel enables clients to easily access the functionality of UModel. As a result, it is now possible to automate a wide range of tasks.

UModel follows the common specifications for automation servers set out by Microsoft. It is possible to access the methods and properties of the UModel API from common development environments, such as those using C, C++, C# and VisualBasic, and with scripting languages like JavaScript and VBScript.

The following guidelines should be considered in your client code:

- Do not hold references to objects in memory longer than you need them. If a user interacts between two calls of your client, then there is no guarantee that these references are still valid.
- Be aware that if your client code crashes, instances of UModel may still remain in the system.
- See [Error handling](#) for details of how to avoid annoying error messages.
- Free references explicitly, if using languages such as C or C++; `"GC.Collect()"` can be used in C# and VisualBasic to force garbage collection.
- Umodel API collections are zero-based (e.g. `myPackage.InsertPackagedElementAt( 0, "Interface" );` will insert a new interface as first child of the package)

20.4.1 Overview

This overview of the UModel API provides you with the object model for the API and a description of the most important API concepts.

The following topics are covered:

- [The object model](#)
- [How to...](#)
- [Error handling](#)

Object model

The starting point for every application which uses the UModel API is the [IApplication](#) interface.

To create an instance of the application object, call `WScript.GetObject("", "UModel.Application")` from JScript, or a similar function from your preferred development environment, to create a COM object.

There is no need (or other way) to create any objects other than to use the complete UModel API. All interfaces are accessed through other objects, with the application object ([IApplication](#)) as the starting point.

Note: When implementing a UModel IDE plugin, there is no need to create an instance of the application object, because UModel is already running and the current instance of the application object is provided by [IApplication](#) as parameter for all important methods of [IUModelPlugin](#).

The application object consists of the following parts (each indentation level indicates a child-parent relationship with the level directly above):

```

IApplication
  IDocument
    IDiagramWindows
      IDiagramWindow
    IFocusedUMLDataNotifier
    ITransactionNotifier
    IUMLData (and all other derived UML data interfaces)
      IUMLDataList
  IDialogs
    IExportXMIFileDlg
    IGenerateDocumentationDlg
      IKindSelectionList
        IKindSelection
    IGenerateSequenceDiagramDlg
    IGenerateStateMachineCodeDlg
    IImportBinaryTypesDlg
      IImportBinaryTypeEntries
        IImportBinaryTypeEntry
    IImportDatabaseDlg
    IImportSourceDirectoryDlg
    IImportSourceProjectDlg
    IImportXMLSchemaDirectoryDlg
    IImportXMLSchemaFileDlg
    IIncludeSubprojectDlg
      IModelTransformationDlg
  
```

[IModelTransformationTypeMappings](#)
[IModelTransformationTypeMapping](#)
[IProjectSettingsDlg](#)
[ISaveAllDiagramsAsImagesDlg](#)
[ISynchronizationSettingsDlg](#)
[IMatchRenamedDlg](#)
[IMatchRenamedEntries](#)
[IMatchRenamedEntry](#)
[IURLDlg](#)
[ILocalOptions](#)
[ILocalOptionsCodeEngineering](#)
[ILocalOptionsDiagramEditing](#)
[ICollectionTemplates](#)
[ICollectionTemplate](#)
[ILocalOptionsEditing](#)
[ILocalOptionsFile](#)
[ILocalOptionsView](#)

[Enumerations](#)

[Events](#)

Once you have created an application object, you can start using the functionality of UModel. You will generally either open an existing document, create a new one, or access the active document ([IDocument](#)).

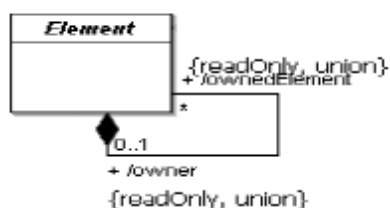
[IDocument](#) corresponds to a UModel project and can be used to include sub-projects, generate documentation, synchronize model and code..., while also giving access to the main UMLData objects (root package, GUI root element, also see [Object model UMLData](#)).

Object model UMLData

The starting point to access UML elements is the root package ([IUMLPackage](#)), which is a property of the [IDocument](#) interface.

All children of the root package are a subtype of [UMLElement](#) and are stored as defined by the OMG in the UML 2.4 Superstructure Specification (also see <http://www.uml.org>).

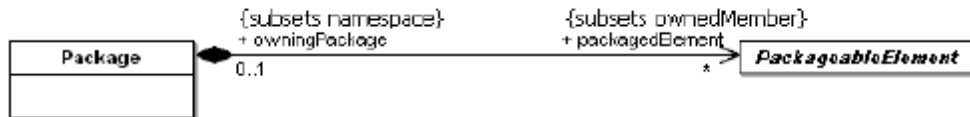
E.g. the UML Superstructure Specification defines the following relationship for UML Element:



Which means that every UML element can have a list of owned elements, and every UML element (apart from the root-package) has an owner:

In the [UModel API](#) this is mapped to [UMLElement](#) having the properties "OwnedElement" and "Owner". Since these relationships are "read only" in the UML specification, both properties cannot be modified in the [UModel API](#).

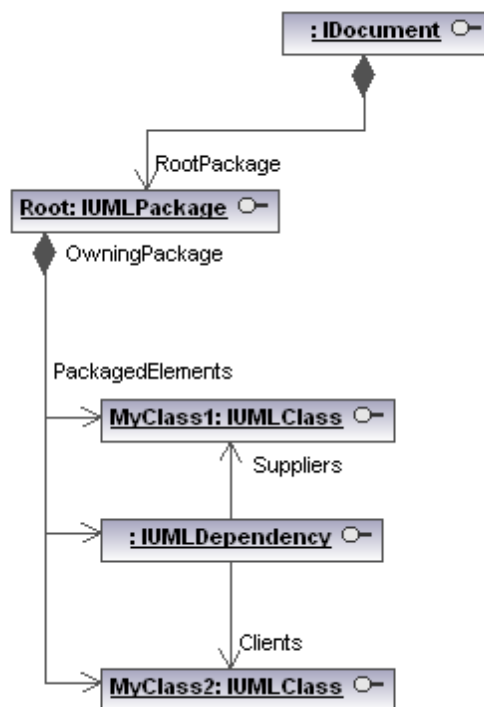
The UML superstructure specification also defines the following relationship between Package and PackageableElement:



This is mapped to [IUMLPackageableElement](#) having a property "OwningPackage" and an [IUMLPackage](#), which not only has a property "PackagedElements", but also a method "InsertPackagedElementAt" to insert new [IUMLPackageableElement](#)s (at the specified position).

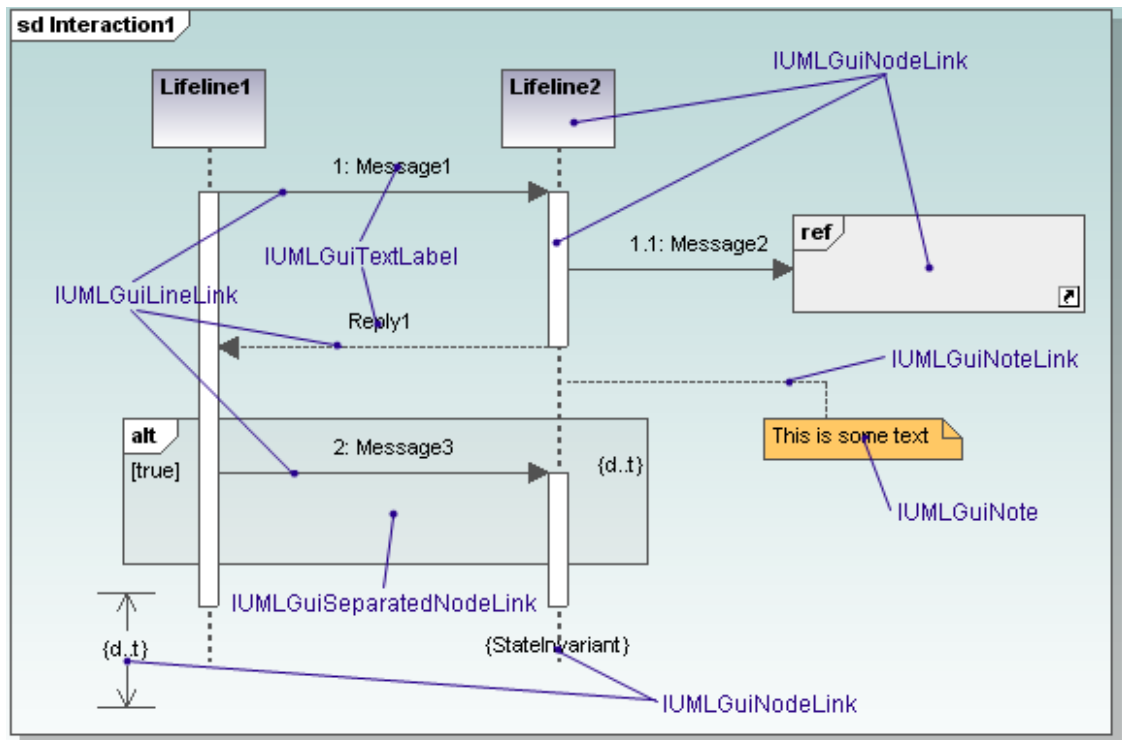
Note: "EraseFromModel" deletes any [IUMLElement](#) (and all its children) from the model.

The sample below shows the mapping of a project which consists of two classes ([IUMLClass](#)) with a dependency ([IUMLDependency](#)) between them:



This structure is independent of whether these elements are shown on any diagram or not.

The representation of graphical objects on diagrams (as shown in the image below) is stored in a second structure with elements of kind [IUMLGuiElement](#) (also see [Graphical Objects](#)).



How to...

How to create Sequence Diagrams

There are two ways to create sequence diagrams programmatically using the UModel API:

- [Generating a sequence diagram from existing source code](#) when there is code available that you want to be reverse engineered and displayed as UML diagram
- [Manually create a sequence diagram](#) from scratch using IUMLGuiElements directly

How to generate Sequence Diagrams from code

Sequence diagrams in UModel can be generated programmatically from an [IUMLOperation](#) element. The operation needs to exist in the model and have some source code associated to it.

The operation could possibly have been previously "read" by the reverse engineering functionality of UModel. Creating new Sequence Diagrams programmatically by reverse engineering source code using the [UModel API](#) involves two short steps:

- Setting up the options for diagram generation
- Invoking the diagram generation function

The following C# code shows how to set up the options and start the generation of the sequence diagram:

```
// starts the sequence diagram generation process based on an
// operation given as parameter
public static void
reverseEngineerAndCreateSequenceDiagram(IApplication application,
```

```
IUMLOperation operation)
{
    GenerateSequenceDiagramDlg dialog =
application.Dialogs.GenerateSequenceDiagramDlg;

    // set some options

    dialog.ShowEmptyCombinedFragments = false;
    dialog.UseDedicatedLineForStaticCalls = true;
    dialog.ShowCodeOfMessagesDisplayedDirectlyBelow = true;
    dialog.ShowCodeInNotes = true;

    dialog.ShowDialog = true; // set this to true if you want the
dialog to be displayed

    // generated the sequence diagram now

    application.ActiveDocument.GenerateSequenceDiagram(dialog,
operation);
}
```

How to create Sequence diagrams manually

Creating new Sequence Diagrams programmatically from scratch using the [UModel API](#) is basically nothing more than placing interaction fragments, such as Lifelines on a diagram and connecting them with messages.

Messages can easily be created using the `AddUMLLineElement()` method of [IUMLGuiLineLink](#), which removes the necessity of creating multiple underlying UML Elements such as MessageEnds, ExecutionOccurrences and similar manually.

To make it simple to create Messages between two interaction fragments such as Lifelines, create a small helper function which calls `AddUMLLineElement()` and layouts the created line:

```
// Creates a message between two interaction fragments (i.e.
lifelines, interaction uses,
// combined fragments or gates) and attaches all necessary elements
like events and activation bars.
// Possible values for 'kind': "Message", "Reply", "Create",
"Destruct"
protected static IUMLMessage addMessage(int ypos, string kind,
IUMLGuiNodeLink from, IUMLGuiNodeLink
to,
DiagramWindow wnd)
{
    // add message

    IUMLGuiLineLink line = wnd.Diagram.AddUMLLineElement(kind, from,
to);
```

```

    if (line == null)
        return null;

    // set position of the line where we want it to show up

    wnd.UpdateWindow();

    if (from == to && line.Waypoints.Count > 3)
    {
        // self-message

        ((IUMLGuiWaypoint) line.Waypoints[1]).SetPos(0, ypos);
        ((IUMLGuiWaypoint) line.Waypoints[4]).SetPos(0, ypos + 25);
    }
    else
    if (line.Waypoints.Count > 1)
    {
        // normal message

        ((IUMLGuiWaypoint) line.Waypoints[1]).SetPos(0, ypos);
        ((IUMLGuiWaypoint) line.Waypoints[2]).SetPos(0, ypos);
    }

    return (IUMLMessage) line.Element;
}

```

As you can see, [IUMLDiagram.AddUMLLineElement\(\)](#) accepts as a parameter not only the string "Message", to create a Message Line; but also "Reply", "Create" and "Destruct", for Reply Messages, Creation Messages and Destruction Messages.

In order to create a simple diagram it is only necessary to create a Sequence Diagram in the GuiRoot object, open the diagram, add a handful of lifelines and connect them with messages using this helper function:

```

IDocument document = theapplication.ActiveDocument;

// create diagram and open it

IUMLGuiSequenceDiagram sequenceDiagram =
    (IUMLGuiSequenceDiagram) document.GuiRoot.InsertOwnedDiagramAt(0,
    document.RootPackage, "SequenceDiagram");

DiagramWindow wnd = document.OpenDiagram(sequenceDiagram);

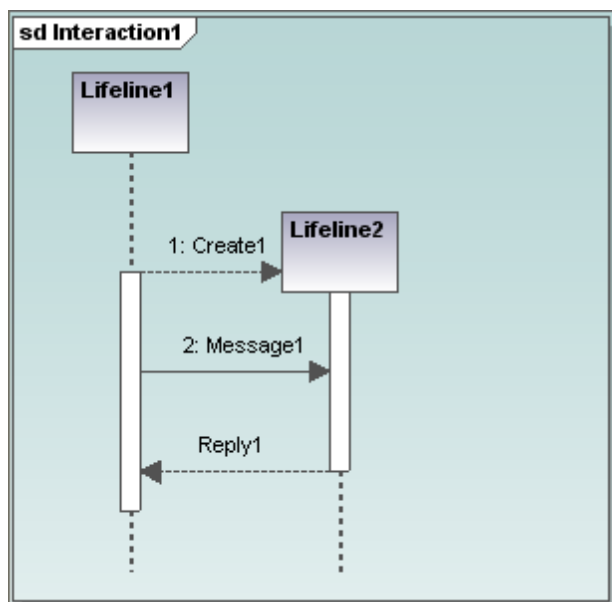
// create two lifelines

IUMLGuiNodeLink lifeline1 = sequenceDiagram.AddUMLElement("Lifeline",
0, 0);
IUMLGuiNodeLink lifeline2 = sequenceDiagram.AddUMLElement("Lifeline",
100, 70);

```

```
// connect these lifelines using some messages  
  
addMessage(100, "Create", lifeline1, lifeline2, wnd);  
addMessage(150, "Message", lifeline1, lifeline2, wnd);  
addMessage(200, "Reply", lifeline2, lifeline1, wnd);
```

The resulting created Diagram will look like this:



Setting the Type of a Lifeline

To display the Type represented by a Lifeline, be it be a Class, Interface, DataType or similar, this is done by using the [IUMLLifeline.Represents](#) property which references a [IUMLProperty](#). If the Type of this property is set, the Type will show up on the diagram as well.

The following code creates a Lifeline which references a class:

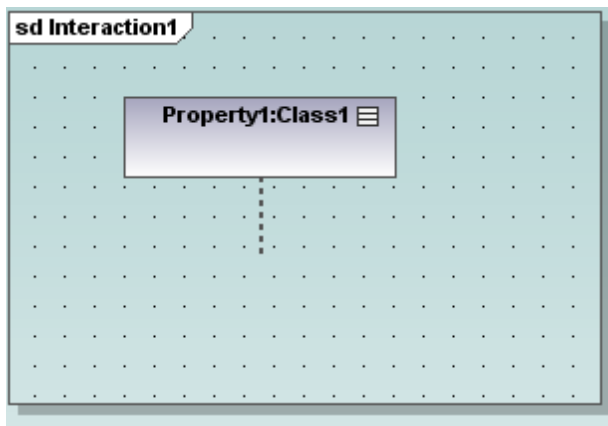
```
// create a class to be referenced by the lifeline  
  
IUMLClass someclass = (IUMLClass)  
document.RootPackage.InsertPackagedElementAt(0, "Class");  
  
// create a lifeline and a property with the class as type in the  
// interaction  
// of the sequence diagram to reference this class  
  
IUMLInteraction interaction = (IUMLInteraction)  
sequenceDiagram.LinkedOwner;  
  
IUMLProperty prop = interaction.InsertOwnedAttributeAt(0);  
prop.Type = someclass;
```

```
UModelLib.IUMLLifeline lifeline = interaction.InsertLifelineAt(0);
lifeline.Represents = (IUMLConnectableElement)prop;

// show the lifeline on the diagram

sequenceDiagram.AddUMLGuiNodeLink(lifeline, 200, 0);
```

The resulting lifeline would then look like this:



Setting the Operation of a Message

Messages usually represent the invocation of an operation of an object. Note: based on the type of the Message (normal Message, Creation, Deletion or Reply) and the existence, or absence of underlying UML elements, such as MessageOccurenceSpecifications or CallEvents, it is not always possible for a Message to represent an Operation, and getting the correct UML element to point to the Operation is not that trivial.

This is why the [IUMLMessage](#) interface in the UModel API, offers the method **SetOperation()** which makes it possible to let a Message refer an Operation if it is able to do so:

```
// create a message, an operation in a class and let the message refer
this operation

IUMLMessage msg = addMessage(250, "Message", lifeline1, lifeline2,
wnd);

UModelLib.IUMLOperation someoperation =
someclass.InsertOwnedOperationAt(0);
someoperation.Name = "SomeOperation";

msg.SetOperation(someoperation);
```

Undo / Redo and UMLData transaction handling

When modifying the UML data structure using the [UModel API](#), there is no need to take care of Undo/Redo or transactions.

The following code makes three modifications:

```
public void ChangeClass( IUMLClass iClass )
{
    iClass.SetName("NewName");
    iClass.Visibility =
ENUMUMLVisibilityKind.eVisibility_Public;
    iClass.IsAbstract = true;
}
```

and for every modification, a new undo-step is created, in other words: the user will have to press the "Undo" button three times in UModel to undo these three changes.

This is not always the required behavior so the [UModel API](#) supports "transaction-handling" making it possible to execute multiple modifications in one step.

[IDocument](#) has the functionality to define when a group of modifications starts ("BeginModification") and when it ends ("EndModification"):

```
public void ChangeClass(IUMLClass iClass, IDocument iDoc)
{
    try
    {
        // make all modifications within one UndoStep; start
modification here
        if (!iDoc.BeginModification())
            return;

        iClass.SetName("NewName");
        iClass.Visibility =
ENUMUMLVisibilityKind.eVisibility_Public;
        iClass.IsAbstract = true;

        // do not forget to end modification and finish
UndoStep
        iDoc.EndModification();
    }
    catch (System.Exception)
    {
        // rollback made changes
        iDoc.AbortModification();

        // add error handling
    }
}
```

This kind of transaction handling may only be used for UML data modifications. Other functions, such as e.g. 'synchronize model from code', will create one single Undo step anyway.

How to use predefined UModel elements

UModel defines several important elements as "predefined". This includes several internal elements (Root, Component View and Unknown Externals package) as well as the elements of all profiles installed with UModel (e.g. the C#, VB and Java profile).

Predefined elements can be uniquely identified by using [ENUMUMLPredefinedElement](#), which allows direct and easy access to these elements for several functionalities, e.g.

- Find a predefined element:

```
// get the CSharp profile
IUMLProfile iCSharpProfile = (IUMLProfile)
iDoc
.RootPackage
.FindPredefinedOwnedElement
(ENUMUMLPredefinedElement.ePredefined_CSharp_Profile, false);
```

- Apply a predefined stereotype

```
// set the CSharp 'delegate' stereotype
iClass
.ApplyPredefinedStereotype
( ENUMUMLPredefinedElement
.ePredefined_CSharp_delegateStereotypeOfClass );
```

- Check if a predefined stereotype is applied:

```
// check if package is a CSharp - namespace (if 'namespace' stereotype
is applied)
bool bIsCSharpNamespace =
iPackage
.IsPredefinedStereotypeApplied
( ENUMUMLPredefinedElement
.ePredefined_CSharp_namespaceStereotypeOfPackage );
```

- Set the tagged value of a predefined tag definition:

```
// set attribute-section "STAThread"
// ...
iStereotypeApp.SetPredefinedTaggedValueAt(-1,
ENUMUMLPredefinedElement
.ePredefined_CSharp_attributesStereotypeOfClass_sectionsProperty,
iSTAThread.Name);
```

How to work with Stereotypes and Tagged Values

Stereotypes and tagged values are quite complex as defined in the UML Superstructure Specification.

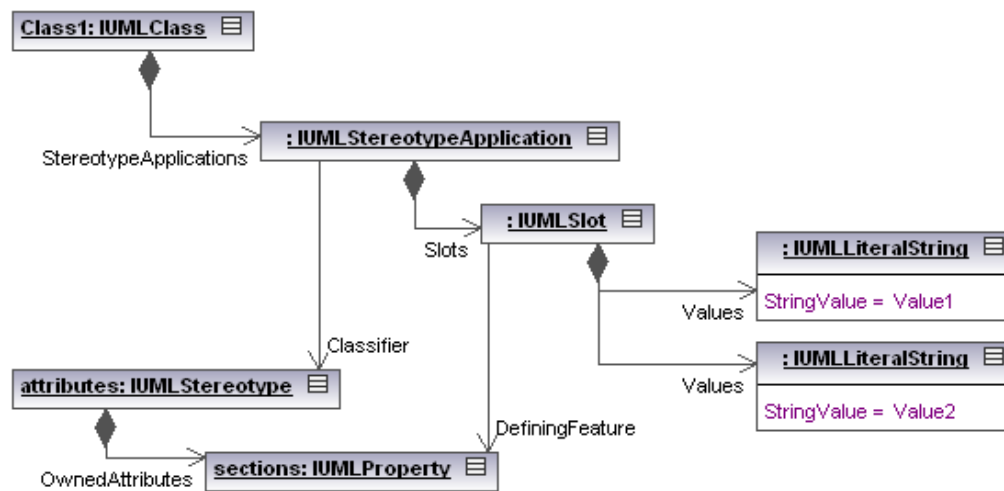
UModel has simplified their handling and treats them similar to [UMLInstanceSpecifications](#) and

[IUMLSlots](#) in UML.

In the following sample the stereotype "attributes" is applied to "Class1", and the tag definition "sections" has the tagged values "Value1" and "Value2":



UModel API introduces [IUMLStereotypeApplications](#) and maps the sample above to the following UMLData structure:



Applying stereotypes and setting tagged values using the UModel API is quite simple:

```

IUMLStereotype iStereotypeAttributes = ...;
IUMLProperty iTagDefSections = ...;
IUMLClass iClass = ...;

IUMLStereotypeApplication iStereotypeApp =
iClass.ApplyStereotype(iStereotypeAttributes);
iStereotypeApp.SetTaggedValueAt(-1, iTagDefSections, "Value1");
iStereotypeApp.SetTaggedValueAt(-1, iTagDefSections, "Value2");

```

Also see section [Predefined UModel elements](#) how to deal with predefined stereotypes, tag definitions and tagged values.

How to use UMLData events and eventfilters

Event receivers must implement the [IUMLDataEvents](#) interface in order to receive one or more of following possible events from [IUMLData](#) objects:

- **OnBeforeErase:** sent immediately before the UML data is erased from the model.
If multiple data are erased, this event is sent for every [IUMLData](#) (not only for the topmost one)
- **OnAfterAddChild:** sent when the UML data is added to the model tree.
If multiple data are added in one step (e.g. a class with multiple attributes is added to a package) only the topmost [IUMLData](#) event is sent
- **OnChanged:** sent when the UML data has been modified (e.g. when a class name is changed)
- **OnMoveData:** sent when the UML data has been moved to a new parent (e.g. when a class is moved to another package in the ModelTree).

This event always occurs twice: once when detaching from the old parent, and once when the UML data is attached to the new parent.

Eventfilter can be set with (combinations of) [ENUMUMLDataEventFilter](#) in order to specify which events should be sent by the [UModel API](#).

To keep performance high and the overhead as low as possible, event receiver should only register for those events, they are really interested in.

E.g. the following code registers "OnAfterAddChild" events when exactly the root-package gets a new child (no event will arrive if a child of the root-package gets a new child):

```
// ensure we get informed when m_RootPackage (and only itself; we do
// not care about its children) gets a new child
m_RootPackage.EventFilter = (int)
ENUMUMLDataEventFilter.eUMLDataEvent_AddChild;
```

UMLData events work hierarchically, so the event filter can be set to receive events from the attached [IUMLData](#) only, or from the attached [IUMLData](#) and any of its children (grandchildren,...).

```
// ensure we get "OnBeforeErase" events also for *any* erased child
// (grandchild,...) of the rootpackage
m_RootPackage.EventFilter |= (int)
ENUMUMLDataEventFilter.eUMLDataEvent_EraseDataOrChild;
```

UMLData events are also sent when UML data is modified by Undo / Redo, but beware that **no** UML data modification may be made during Undo / Redo:

```
public void OnAfterAddChild(IUMLData ipUMLParent, IUMLData ipUMLChild)
{
    // check if child was added by undo/redo
    // (we are not allowed to modify anything during Undo/Redo !!)
    IDocument iDoc = (IDocument) ipUMLChild.Parent;
    if (!iDoc.IsInUndoRedo)
    {
        // ...
    }
}
```

How to create and use Hyperlinks

UModel allows hyperlinks between most modeling elements (except for lines) and:

- any diagram in the current ump project
- any element on a diagram
- any element in the Model Tree
- external documents, e.g. PDF, Excel or Word documents
- web pages

Also see [Hyperlinking modeling elements](#).

Hyperlinks are not part of the UML specification and the UModel API introduces following interfaces for hyperlinks on

[IUMLNamedElements](#):

- [IUMLHyperlink](#) is the common base interface and can be used to open links as well as to retrieve the default- and user-defined link name
- [IUMLHyperlink2File](#) to handle external documents and web pages
- [IUMLHyperlink2GuiElement](#) to handle any diagram in the current ump project or any element on a diagram
- [IUMLHyperlink2Model](#) for hyperlinks to model elements (in the Model Tree)

Examples:

Insert a hyperlink to the Altova homepage:

```
IUMLHyperlink2File iHyperlink = iMyClass.InsertOwnedHyperlink2FileAt(-1, "http://www.altova.com");
```

Insert a hyperlink to a diagram of the current ump project:

```
IUMLGuiDiagram iDiagram = ...;
IUMLHyperlink2GuiElement iHyperlink =
iMyClass.InsertOwnedHyperlink2GuiElementAt(-1, iDiagram, null);
```

Insert a hyperlink to the representation of a class on a diagram:

```
IUMLGuiNodeLink iNodeLink = ...;
IUMLHyperlink2GuiElement iHyperlink =
iMyClass.InsertOwnedHyperlink2GuiElementAt(-1, iNodeLink, null);
```

Insert a hyperlink to an attribute of a class on a diagram:

```
IUMLGuiNodeLink iNodeLink = ...;
IUMLProperty iAttribute = ...;
IUMLHyperlink2GuiElement iHyperlink =
iMyClass.InsertOwnedHyperlink2GuiElementAt(-1, iNodeLink, iAttribute);
```

Insert a hyperlink to the same attribute (from above) in the Model Tree:

```
IUMLHyperlink2Model iHyperlink = iMyClass.InsertOwnedHyperlink2ModelAt(-1,
iAttribute);
```

Open all hyperlinks of an [IUMLNamedElement](#):

```
foreach (IUMLHyperlink iHyperlink in iMyClass.OwnedHyperlinks)
    iHyperlink.OpenLink();
```

UModel also allows hyperlinks in notes ([IUMLGuiNote](#)) and comments ([IUMLComment](#)):



These are handled by [IUMLGuiTextHyperlinks](#) (respectively [IUMLCommentTextHyperlinks](#)) and the start- and end-character position of the hyperlink must be specified, e.g:

```
IUMLGuiDiagram iDiagram = ...;
IUMLGuiNote iNote = iDiagram.AddUMLGuiNote(200, 100);

iNote.NoteText = "This is my Altova link";
int nStart = iNote.NoteText.IndexOf("Altova");
int nEnd = nStart + "Altova".Length;

IUMLGuiTextHyperlink iHyperlink =
iNote.InsertOwnedGuiTextHyperlinkAt(nStart, nEnd, "http://
www.altova.com");
```

Similar for hyperlinks in comments:

```
IUMLComment iComment = ...;
IUMLClass iClass2 = ...;

iComment.Body = "This is my link to Class2";
int nStart = iComment.Body.IndexOf("Class2");
int nEnd = nStart + "Class2".Length;

IUMLCommentTextHyperlink iHyperlink =
iComment.InsertOwnedCommentTextHyperlinkAt(nStart, nEnd, "");
iHyperlink.SetHyperlinkModelElementAddress( iClass2 );
```

Error handling

The UModel API returns errors in two different ways. Every API method returns an `HRESULT`. This return value informs the caller about any errors during the execution of the method. If the call was successful, the return value is equal to `S_OK`. C/C++ programmers generally use `HRESULT` to detect errors.

VisualBasic, scripting languages, and other high-level development environments do not give the programmer access to the returning `HRESULT` of a COM call. They use the second error-raising mechanism supported by the UModel API, the `IErrorInfo` interface. If an error occurs, the API creates a new object that implements the `IErrorInfo` interface. The development environment takes this interface and fills its own error-handling mechanism with the provided information.

The following text describes how to deal with errors raised from the UModel API in different development environments.

VisualBasic

A common way to handle errors in VisualBasic is to define an error handler. This error handler can be set with the `On Error` statement. Usually the handler displays an error message and performs cleanup functions to avoid spare references and any kind of resource leaks.

VisualBasic fills its own `Err` object with the information from the `IErrorInfo` interface.

Example:

```
Sub Validate()  
    'place variable declarations here  
  
    'set error handler  
    On Error GoTo ErrorHandler  
  
    'if DoSomeWork fails, program execution continues at ErrorHandler:  
    objUModel.ActiveDocument.DoSomeWork()  
  
    'additional code comes here  
  
    'exit  
Exit Sub  
  
ErrorHandler:  
    MsgBox("Error: " & (Err.Number - vbObjectError) & Chr(13) &  
        "Description: " & Err.Description)  
End Sub
```

JavaScript

The Microsoft implementation of JavaScript (JScript) provides a try-catch mechanism to deal with errors raised from COM calls. It is very similar to the VisualBasic approach, in that you also declare an error object containing the necessary information.

Example:

```
function Generate()  
{  
    // please insert variable declarations here  
  
    try  
    {  
        objUModel.ActiveDocument.DoSomeWork();  
    }  
    catch(Error)  
    {  
        sError = Error.description;  
        nErrorCode = Error.number & 0xffff;  
        return false;  
    }  
  
    return true;  
}
```

```
}
```

C/C++

C/C++ gives you easy access to the HRESULT of the COM call and to the IErrorInterface.

```
eobpri q= X

// Call DoSomeWork() from the UModel API
if(FAILED(hr = ipDocument->DoSomeWork()))
{
    IErrorInfo *ipErrorInfo = Null;

    if(SUCCEEDED(::GetErrorInfo(0, &ipErrorInfo)))
    {
        BSTRbstrDescr;
        ipErrorInfo->GetDescription(&bstrDescr);

        // handle Error information
        wprintf(L"Error message:\t%s\n",bstrDescr);
        ::SysFreeString(bstrDescr);

        // release Error info
        ipErrorInfo->Release();
    }
}
```

Programming languages

Programming languages differ in the way they support COM access. A few examples for the most frequently used languages (*links below*) will help you get started. The code listings in this section are available in files in the API Examples folder and can be tested straight away. The path to the API Examples folder is given below:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/UModel2016/UModelExamples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/UModel2016/UModelExamples/API/

C#

C# can be used to access the Application API functionality. The code listings show how to access the API for certain basic functionality.

Java

The UModel API can be accessed from Java code. This section explains how some basic UModel functionality can be accessed from Java code. It is organized into the following sub-sections.

JScript

The JScript listings demonstrate the following basic functionality.

C#

The C# programming language can be used to access the Application API functionality. You could use Visual Studio 2008 or Visual Studio 2010 to create the C# code, saving it in a Visual Studio project. Create the project as follows:

1. In Microsoft Visual Studio, add a new project using **File | New | Project**.
2. Add a reference to the UModel Type Library by clicking **Project | Add Reference**. The Add Reference dialog pops up, displaying a list of installed COM components. Select the UModel Type Library component from the list to add it.
3. Enter the code you want.
4. Compile the code and run it.

Example C# project

Your UModel package contains an example C# project, which is located in the C# folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/UModel2016/UModelExamples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/UModel2016/UModelExamples/API/

You can compile and run the project from within Visual Studio 2008 or Visual Studio 2010.

Platform configuration

If you have a 64-bit operating system and are using a 32-bit installation of UModel, you must add the x86 platform in the solution's Configuration Manager and build the sample using this configuration.

A new x86 platform (for the active solution in Visual Studio) can be created in the New Solution Platform dialog (Build | Configuration Manager | Active solution platform | <New...>)

File List:

Readme.txt	This file
AutomateUModel.csproj	Visual Studio 2008 and 2010 project file
AutomateUModel_VS2008.sln	Visual Studio 2008 solution file
AutomateUModel_VS2010.sln	Visual Studio 2010 solution file
Program.cs	C# example source code

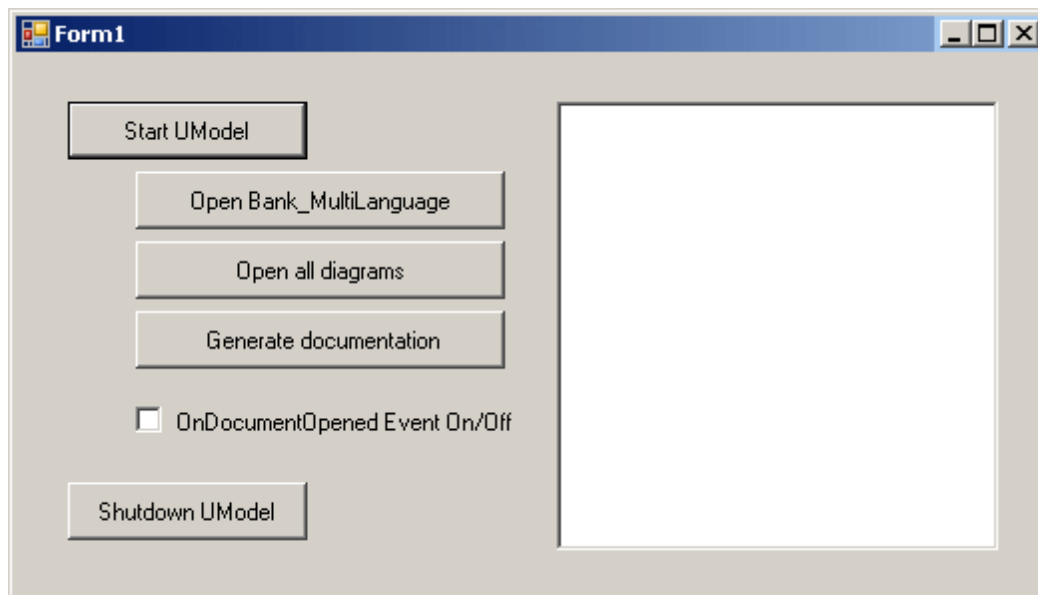
Properties:

Form1.cs
Form1.Designer.cs
Form1.resx

What the example does

The example shows a simple user interface (*screenshot below*) with buttons that invoke basic UModel operations:

- Start UModel
- Open Bank_MultiLanguage.ump
- Open All Diagrams
- Generate documentation for the currently active document
- Shows how to listen to UModel events (OnDocumentOpened Event On/Off)
- Shutdown UModel



Compiling and running the example

Double-click the file `AutomateUModel_VS2008.sln` (for Visual Studio 2008) or the file `AutomateUModel_VS2010.sln` (for Visual Studio 2010). Alternatively the file can be opened from within Visual Studio (with **File | Open | Project/Solution**). To compile and run the example, select **Debug | Start Debugging** or **Debug | Start Without Debugging**.

Code listing of Form1.cs

The listing is commented for ease of understanding. Parts of the code are also presented separately in the sub-sections of this section, according to the Application API functionality they access.

The code essentially consists of a series of handlers for the buttons in the user interface shown in the screenshot above.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
```

```
namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            // An instance of UModel accessed via its automation interface.
            UModelLib.Application UModel;
            UModelLib.Document UModelDocument;
            String docName = "Bank_MultiLanguage.ump";
            // Location of examples installed with UModel
            String strExamplesFolder;

            private void Form1_Load(object sender, EventArgs e)
            {
                // locate examples installed with UModel.
                // REMARK: You might need to adapt this if you have a different
                major version of the product.
                strExamplesFolder =
                Environment.GetEnvironmentVariable("USERPROFILE") + "\\My Documents\\Altova\\
                \\UModel2012\\UModelExamples\\";
            }

            // handler for the "Start UModel" button
            private void StartUModel_Click(object sender, EventArgs e)
            {
                if (UModel == null)
                {
                    Cursor.Current = Cursors.WaitCursor;

                    // if we have no UModel instance, we create one and make it
                    visible.
                    UModel = new UModelLib.Application();
                    UModel.Visible = true;

                    Cursor.Current = Cursors.Default;
                }
                else
                {
                    // if we have already an UModel instance running we toggle its
                    visibility flag.
                    UModel.Visible = !UModel.Visible;
                }
            }

            // handler for the "Open Bank_Multilanguage" button
            private void openBank_MultiLanguage_Click(object sender, EventArgs e)
            {
                if (UModel == null)
                    StartUModel_Click(null, null);

                // Open one of the sample files installed with the product.
                UModelDocument = UModel.OpenDocument(strExamplesFolder + docName);
            }
        }
    }
}
```

```

    }

    // handler for the "Open All Diagrams" button
    private void openAllDiagrams_Click(object sender, EventArgs e)
    {
        if (UModel == null)
            StartUModel_Click(null, null);

        if (UModelDocument != null)
        {
            UModelDocument.OpenAllDiagrams();
            UModelLib.DiagramWindows diagramWindows =
            UModelDocument.DiagramWindows;
            System.Collections.IEnumerator diagramWindowsEnum =
            diagramWindows.GetEnumerator();

            if ( diagramWindows.Count == 0 )
                listBoxMessages.Items.Add("No opened diagrams.");
            else
                listBoxMessages.Items.Add("Opened diagrams:");

            int count = 0;

            while ( diagramWindowsEnum.MoveNext() )
            {
                String sName = "";
                Object obj = diagramWindowsEnum.Current;

                if (obj is UModelLib.DiagramWindow)
                {
                    sName = ((UModelLib.DiagramWindow)obj).Name;
                    listBoxMessages.Items.Add("\t" + ++count + " " + sName);
                }
            }
        }
    }

    // handler for the "Shutdown UModel" button
    // shut-down application instance by explicitly releasing the COM
    object.
    private void shutdownUModel_Click(object sender, EventArgs e)
    {
        if (UModel != null)
        {
            // allow shut-down of UModel by releasing UI
            UModel.Visible = false;

            // explicitly release COM object
            try
            {
                while
                (System.Runtime.InteropServices.Marshal.ReleaseComObject(UModel) > 0) ;
            }
            finally
            {
                // avoid later access to this object.
                UModel = null;
            }
        }
    }

```

```

    }
}

// handler for button "Generate Documentation"
private void generateDocumentation_Click(object sender, EventArgs e)
{
    if (UModel == null)
        listBoxMessages.Items.Add("start UModel first.");
    // COM errors get returned to C as exceptions. We use a try/catch
    block to handle them.
    try
    {
        // generate documentation
        UModelLib.Dialogs dlgs = UModel.Dialogs;
        UModelLib.ENUMDocumentationOutputFormat docFormat =
        UModelLib.ENUMDocumentationOutputFormat.eDocumentationOutputFormat_HTML;
        UModelLib.GenerateDocumentationDlg docDlg =
        dlgs.GenerateDocumentationDlg;
        docDlg.OutputFormat = docFormat;
        String strDocOutputFolder = strExamplesFolder +
        "GeneratedDocFromCExample\\";
        if (!Directory.Exists(strDocOutputFolder))
        {
            Directory.CreateDirectory(strDocOutputFolder);
            listBoxMessages.Items.Add("Created folder \"" +
            strDocOutputFolder + "\"");
        }
        String strResultFile = strDocOutputFolder +
        docName.Replace(".ump", ".html"); ;
        listBoxMessages.Items.Add("Generating documentation...");
        UModelDocument.GenerateDocumentation(docDlg, strResultFile);
        listBoxMessages.Items.Add("Generated document: " +
        strResultFile);
    }
    catch (Exception ex)
    {
        // The COM call was not successful.
        // Probably no application instance has been started or no
        document is open.
        MessageBox.Show("COM error: " + ex.Message);
    }
}

delegate void addListBoxItem_delegate(string sText);
// called from the UI thread
private void addListBoxItem(string sText)
{
    listBoxMessages.Items.Add(sText);
}
// wrapper method to allow to call UI controls methods from a worker
thread
void syncWithUiThread(Control ctrl, addListBoxItem_delegate
methodToInvoke, String sText)
{
    // Control.Invoke: Executes on the UI thread, but calling thread
    waits for completion before continuing.

```

```

        // Control.BeginInvoke: Executes on the UI thread, and calling
        thread doesn't wait for completion.
        if (ctrl.InvokeRequired)
            ctrl.BeginInvoke(methodToInvoke, new Object[] { sText });
    }

    // event handler for OnDocumentOpened event
    private void handleOnDocumentOpened(UModelLib.Document i_ipDocument)
    {
        String sText = "";

        if (i_ipDocument.Name.Length > 0)
            sText = "Document " + i_ipDocument.Name + " was opened!";
        else
            sText = "An empty document was created.";

        // we need to synchronize the calling thread with the UI thread
        because
        // the COM events are triggered from a working thread
        addListBoxItem_delegate methodToInvoke = new
addListBoxItem_delegate(addListBoxItem);
        // call syncWithUIThread with the following arguments:
        // 1 - listBoxMessages - list box control to display messages from
        COM events
        // 2 - methodToInvoke - a C delegate which points to the method
        which will be called from the UI thread
        // 3 - sText - the text to be displayed in the list box
        syncWithUIThread(listBoxMessages, methodToInvoke, sText);
    }

    private void checkBoxEventOnOff_CheckedChanged(object sender, EventArgs
e)
    {
        if (UModel != null)
        {
            if (checkBoxEventOnOff.Checked)
                UModel.OnDocumentOpened += new
UModelLib._IApplicationEvents_OnDocumentOpenedEventHandler(handleOnDocumentOpene
d);
            else
                UModel.OnDocumentOpened -= new
UModelLib._IApplicationEvents_OnDocumentOpenedEventHandler(handleOnDocumentOpene
d);
        }
    }
}

```

Java

The Application API can be accessed from Java code. To allow accessing the UModel automation server directly from Java code, the libraries listed below must reside in the `classpath`. They are installed in the folder: `JavaAPI` in the UModel application folder.

- `AltovaAutomation.dll`: a JNI wrapper for Altova automation servers
- `AltovaAutomation.jar`: Java classes to access Altova automation servers

- `UModelAPI.jar`: Java classes that wrap the UModel automation interface
- `UModelAPI_JavaDoc.zip`: a Javadoc file containing help documentation for the Java API

Note: In order to use the Java API, the DLL and Jar files must be on the Java Classpath.

Example Java project

An example Java project is supplied with your product installation. You can test the Java project and modify and use it as you like. For more details of the example Java project, see the section, [Example Java Project](#).

Rules for mapping the Application API names to Java

The rules for mapping between the Application API and the Java wrapper are as follows:

- **Classes and class names**
For every interface of the UModel automation interface a Java class exists with the name of the interface.
- **Method names**
Method names on the Java interface are the same as used on the COM interfaces but start with a small letter to conform to Java naming conventions. To access COM properties, Java methods that prefix the property name with `get` and `set` can be used. If a property does not support write-access, no setter method is available. Example: For the `Name` property of the `Document` interface, the Java methods `getName` and `setName` are available.
- **Enumerations**
For every enumeration defined in the automation interface, a Java enumeration is defined with the same name and values.
- **Events and event handlers**
For every interface in the automation interface that supports events, a Java interface with the same name plus 'Event' is available. To simplify the overloading of single events, a Java class with default implementations for all events is provided. The name of this Java class is the name of the event interface plus 'DefaultHandler'. For example:
`Application`: Java class to access the application
`ApplicationEvents`: Events interface for the Application
`ApplicationEventsDefaultHandler`: Default handler for `ApplicationEvents`

The following section explains how some basic UModel functionality can be accessed from Java code.

- [Example Java Project](#)

Example Java Project

The UModel installation package contains an example Java project, located in the Java folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/UModel2016/UModelExamples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/UModel2016/UModelExamples/API/

This folder contains Java examples for the UModel API. You can test it directly from the command line using the batch file `BuildAndRun.bat`, or you can compile and run the example project from within Eclipse. See below for instructions on how to use these procedures.

File list

The Java examples folder contains all the files required to run the example project. These files are listed below:

<code>AltovaAutomation.dll</code>	Java-COM bridge: DLL part
<code>AltovaAutomation.jar</code>	Java-COM bridge: Java library part
<code>UModelAPI.jar</code>	Java classes of the UModel API
<code>RunUModel.java</code>	Java example source code
<code>BuildAndRun.bat</code>	Batch file to compile and run example code from the command line prompt. Expects folder where Java Virtual Machine resides as parameter.
<code>.classpath</code>	Eclipse project helper file
<code>.project</code>	Eclipse project file
<code>UModelAPI_JavaDoc.zip</code>	Javadoc file containing help documentation for the Java API
<code>Readme.txt</code>	This file

What the example does

The example starts up UModel and performs a few operations, including opening and closing documents. When done, UModel stays open. You must close it manually.

Running the example from the command line

To run the example from the command line, open a command prompt window, go to the Java folder of the API Examples folder (*see above for location*), and then type:

```
buildAndRun.bat "<Path-to-the-Java-bin-folder>"
```

The Java binary folder must be that of a JDK 1.5 or later installation on your computer.

Press the **Return** key. The Java source in `RunUModel.java` will be compiled and then executed.

Loading the example in Eclipse

Open Eclipse and use the **File | Import... | General | Existing Projects into Workspace** command to add the Eclipse project file (`.project`) located in the Java folder of the API Examples folder (*see above for location*). The project `RunUModel` will then appear in your Package Explorer or Navigator.

Select the project and then the command **Run as | Java Application** to execute the example.

Note: You can select a class name or method of the Java API and press F1 to get help for that class or method.

Java source code listing

The Java source code in the example file `RunUModel.java` is listed below with comments.

```
// access general JAVA-COM bridge classes
import java.io.File;
import java.util.Iterator;

import com.altova.automation.libs.*;

// access XMLSpy Java-COM bridge
import com.altova.automation.UModel.*;
import com.altova.automation.UModel.Enums.ENUMDocumentationOutputFormat;

/**
 * A simple example that starts UModel COM server and performs a few operations
 * on it.
 * Feel free to extend.
 */
public class RunUModel
{
    public static void main(String[] args)
    {
        // an instance of the application.
        Application umodel = null;

        // instead of COM error handling use Java exception mechanism.
        try
        {
            // Start UModel as COM server.
            umodel = new Application();
            // COM servers start up invisible so we make it visible
            umodel.setVisible(true);

            String docName = "Bank_MultiLanguage.ump";

            // The following lines attach to the application events using a default
            // implementation
            // for the events and override one of its methods.
            // If you want to override all application events it is better to
            // derive your listener class
            // from ApplicationEvents and implement all methods of this interface.
            umodel.addListener(new ApplicationEventsDefaultHandler()
            {
                @Override
                public void onDocumentOpened(Document i_ipDoc) throws
AutomationException
                {
                    String name = i_ipDoc.getName();
                }
            });
        }
    }
}
```

```

        System.out.println("Document " + name + " was opened.");
    }
});

// Locate samples installed with the product.
String strExamplesFolder = System.getenv("USERPROFILE") + "\\My
Documents\\Altova\\UModel2012\\UModelExamples\\";
Document doc = umodel.openDocument(strExamplesFolder + docName);

// The following lines attach to the document events using a default
implementation
// for the events and override one of its methods.
// If you want to override all document events it is better to derive
your listener class
// from DocumentEvents and implement all methods of this interface.
doc.addListener(new DocumentEventsDefaultHandler()
{
    @Override
    public void onDiagramWindowOpened(DiagramWindow ipDiagram) throws
AutomationException
    {
        String name = ipDiagram.getName();

        System.out.println("Diagram " + name + " was opened.");
    }
});
// open all diagrams
doc.openAllDiagrams();

DiagramWindows diagramWindows = doc.getDiagramWindows();

Iterator<DiagramWindow> itr = diagramWindows.iterator();

if (diagramWindows.getCount() == 0)
    System.out.print("No opened diagrams.\n");
else
    System.out.print("Opened diagrams:\n");

int count = 0;
while (itr.hasNext())
{
    // iterate through the opened diagram windows
    String sName = "";
    Object element = itr.next();
    if (element instanceof DiagramWindow)
    {
        sName = ((DiagramWindow)element).getName();
        System.out.print("\t" + ++count + " " + sName + "\n");
    }
}

// generate documentation
Dialogs dlgs = umodel.getDialogs();
ENUMDocumentationOutputFormat docFormat =
ENUMDocumentationOutputFormat.eDocumentationOutputFormat_HTML;
GenerateDocumentationDlg docDlg = dlgs.getGenerateDocumentationDlg();
docDlg.setOutputFormat(docFormat);

```

```

        String strDocOutputFolder = strExamplesFolder + "Generated
documentation\\Java\\";
        if ((new File(strDocOutputFolder)).mkdirs())
            System.out.println("Created folder \"" + strDocOutputFolder + "\"");
        String strResultFile = strDocOutputFolder + docName.replace(".ump",
".html");
        System.out.print("Generating documentation...\n");
        doc.generateDocumentation(docDlg, strResultFile);
        System.out.print("Generated document: " + strResultFile + "\n");

        System.out.println("Watch UModel!");
    }
    catch (AutomationException e)
    {
        // e.printStackTrace();
    }
    finally
    {
        // Make sure that UModel can shut down properly.
        if (umodel != null)
            umodel.dispose();

        // Since the COM server was made visible and still is visible, it will
        keep running
        // and needs to be closed manually.
        System.out.println("Now close UModel!");
    }
}
}

```

JScript

This section contains listings of JScript code that demonstrate the following basic functionality:

Example files

The code is available in example files that you can test as is or modify to suit your needs. The JScript example files are located in the JScript folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/UModel2016/UModelExamples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/UModel2016/UModelExamples/API/

The example files can be run in one of two ways:

- *From the command line:*
Open a command prompt window and type the name of one of the example scripts (for example, `Start.js`). The Windows Scripting Host that is packaged with newer versions of Windows (XP, Vista, 7) will execute the script.
- *From Windows Explorer:*
In Windows Explorer, browse for the JScript file and double-click it. The Windows

Scripting Host that is packaged with newer versions of Windows (XP, Vista, 7) will execute the script. After the script is executed, the command console gets closed automatically.

DocumentAccess.js	Shows how to open , iterate and close documents
GenerateDoc.js	Shows how to generate documentation using JScript
Start.js	Start UModel as a registered automation server, or connects to a running instance.
UModelCreateCode.js	Shows forward engineering to create code elements in a new UModel project and then generate code.
UModelUpdateDocumentation.js	reverse engineers all C# UModelAPI samples, and creates HTML and RTF documentation.
Readme.txt	This file.

Start application

The JScript below starts the application and shuts it down. If an instance of the application is already running, the running instance will be returned.

Script listing - Start.js

The JScript listing below is explained with comments in the code.

```
// Initialize application's COM object. This will start a new instance of the
// application and
// return its main COM object. Depending on COM settings, a the main COM object
// of an already
// running application might be returned.
try
{
    objUModel = WScript.GetObject("", "UModel.Application");
}
catch(err)
{
    Exit("Can't access or create UModel.Application");
}

// if newly started, the application will start without its UI visible. Set it
// to visible.
objUModel.Visible = true;

WScript.Echo(objUModel.Edition + " has successfully started. ");

objUModel.Visible = false; // will shutdown application if it has no more COM
connections
//objUModel.Visible = true; // will keep application running with UI visible
```

Running the script

The JScript code listed above is available in the file `Start.js` located in the `JScript` folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/UModel2016/UModelExamples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/UModel2016/UModelExamples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

Document Access

The JScript listing below shows how to open documents, set a document as the active document, iterate through the open documents, and close documents.

Script listing - DocumentAccess.js

The JScript listing below is explained with comments in the code.

```
// Initialize application's COM object. This will start a new instance of the
// application and
// return its main COM object. Depending on COM settings, a the main COM object
// of an already
// running application might be returned.
try
{
    objUModel = WScript.GetObject("", "UModel.Application");
}
catch(err)
{
    Exit("Can't access or create UModel.Application");
}

// if newly started, the application will start without its UI visible. Set it
// to visible.
objUModel.Visible = true;

// ***** code snippet for "Simple Document Access"
// *****

// Locate examples via USERPROFILE shell variable. The path needs to be adapted
// to major release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
strExampleFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\\My
Documents\\Altova\\UModel2012\\UModelExamples\\";

objDoc = objUModel.OpenDocument(strExampleFolder + "Bank_MultiLanguage.ump");
// open all diagrams
objDoc.OpenAllDiagrams();

// ***** code snippet for "Simple Document Access"
// *****
```

```
// ***** code snippet for "Iteration"
// *****

objName = "";
count = 0;
// go through all open diagrams using a JScript Enumerator
for (var iterDiagrams = new
Enumerator(objDoc.DiagramWindows); !iterDiagrams.atEnd();
iterDiagrams.moveNext())
{
    objName += "\t" + ++count + " " + iterDiagrams.item().Name + "\n";
}

WScript.Echo("Opened diagrams: \n" + objName);

// go through all open diagrams using index-based access to the document
collection
for (i = objDoc.DiagramWindows.Count; i > 0; i--)
    objDoc.DiagramWindows.Item(i).Close();

// ***** code snippet for "Iteration"
// *****

//objUModel.Visible = false;      // will shutdown application if it has no more
COM connections
objUModel.Visible = true;    // will keep application running with UI visible
```

Generate documentation

The JScript listing below shows how to generate documentation for the Bank_MultiLanguage.ump file in the UModelExamples folder.

Script listing - GenerateDoc.js

The JScript listing below is explained with comments in the code.

```
// Initialize application's COM object. This will start a new instance of the
application and
// return its main COM object. Depending on COM settings, a the main COM object
of an already
// running application might be returned.
try
{
    objUModel = WScript.GetObject("", "UModel.Application");
}
catch(err)
{
    Exit("Can't access or create UModel.Application");
}

// if newly started, the application will start without its UI visible. Set it
to visible.
objUModel.Visible = true;

// Locate examples via USERPROFILE shell variable. The path needs to be adapted
```

```

to major release versions.
objWshShell = WScript.CreateObject("WScript.Shell");
strExamplesFolder = objWshShell.ExpandEnvironmentStrings("%USERPROFILE%") + "\
\My Documents\Altova\UModel2012\UModelExamples\";

objDoc = objUModel.OpenDocument(strExamplesFolder + "Bank_MultiLanguage.ump");

// generate documentation
dlg = objUModel.Dialogs;
docDlg = dlg.GenerateDocumentationDlg;
docDlg.OutputFormat = 0; //
ENUMDocumentationOutputFormat.eDocumentationOutputFormat_HTML

var myObject = new ActiveXObject("Scripting.FileSystemObject");
strDocOutputFolder = strExamplesFolder + "GeneratedDocWithJScriptExample\\";

if (!myObject.FolderExists(strDocOutputFolder))
    myObject.CreateFolder(strDocOutputFolder);

strResultFile = strDocOutputFolder + "Bank_MultiLanguage.html";
objDoc.generateDocumentation(docDlg, strResultFile);

//objUModel.Visible = false; // will shutdown application if it has no more
COM connections
objUModel.Visible = true; // will keep application running with UI visible

```

Running the script

The JScript code listed below is available in the file `DocumentAccess.js` located in the JScript folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/UModel2016/UModelExamples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/UModel2016/UModelExamples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

Generate code

The JScript listing below shows how to open documents, set a document as the active document, iterate through the open documents, and generate code.

Script listing - UModelCreateCode.js

The JScript listing below is explained with comments in the code.

```

// #####
// access running UModel.Application or
// launch new one and access it
// #####

```

```
// #####
// CreateCode sample
// shows forward engineering from scratch
// it creates some coding elements in a new UModel project and generates code
// (saving the project afterwards)
// #####

// /////////// global variables ///////////
var objUModel = null;
var objWshShell = null;
var objFSO = null;

// /////////// Helpers ///////////

function Exit(strErrorText)
{
    WScript.Echo(strErrorText);

    if (objUModel != null)
        objUModel.Quit();

    WScript.Quit(-1);
}

function CreateGlobalObjects ()
{
    // the Shell and FileSystemObject of the windows scripting host often always
    // useful
    try
    {
        objWshShell = WScript.CreateObject("WScript.Shell");
        objFSO = WScript.CreateObject("Scripting.FileSystemObject");
    }
    catch(err)
    {
        Exit("Can't create WScript.Shell object");
    }

    // create the UModel connection
    // if there is a running instance of UModel (that never had a connection) -
    // use it
    // otherwise, we automatically create a new instance
    try
    {
        objUModel = WScript.GetObject("", "UModel.Application");
    }
    catch(err)
    {
        Exit("Can't access or create UModel.Application");
    }
}

function GetSourceCodeDirectory()
```



```

{
    // get directory for source code
    var path = objUModel.PersonalDataDirectory + "\\UModelExamples\\API\\JScript\\CreateCode";
    var codeDirectory = objFSO.BuildPath( path, "SampleCode" );
    return codeDirectory;
}

function GetUMPFilePath()
{
    // get file path to save UModel projectfile
    return objUModel.PersonalDataDirectory + "\\UModelExamples\\API\\JScript\\CreateCode\\CreateCode.ump";
}

function IncludeCSharpProfile( objDocument )
{
    try
    {
        // get dialog for including subprojects:
        var objIncludeSubProjectDialog = objUModel.Dialogs.IncludeSubprojectDlg;

        objIncludeSubProjectDialog.ProjectFile = objUModel.InstallationDirectory + "\\UModelInclude\\c Profile.ump";

        return objDocument.IncludeSubproject( objIncludeSubProjectDialog );
    }
    catch(err)
    {
        Exit("Can't include CSharp profile");
    }
}

// ////////////////////////////////// MAIN //////////////////////////////////

CreateGlobalObjects();

objUModel.Visible = true;

// open an new, empty document
var objDocument = objUModel.NewDocument();
// get the root-package
var objRootPackage = objDocument.RootPackage;

if ( objDocument != null &&
    objRootPackage != null &&
    IncludeCSharpProfile( objDocument ) )
{
    // create coding elements
    try
    {
        {
            // make all modifications within one UndoStep; start modification here
            if ( !objDocument.BeginModification() )
                Exit("No modifications allowed");

            // create a namespace root package
            var objCSharpRootNamespace = objRootPackage.InsertPackagedElementAt( -1,

```

```

"Package" );
    objCSharpRootNamespace.SetName( "CSharp" );

    // find C Profile...
    var objCSharpProfile = objRootPackage.FindPredefinedOwnedElement( 159,
false );// ePredefined_CSharp_Profile = 159,
    // ...and apply it to the package, which is now a CSharp namespace root
    objCSharpRootNamespace.InsertProfileApplicationAt( -1, objCSharpProfile );

    // create a C namespace package...
    var objCSharpNamespace = objCSharpRootNamespace.InsertPackagedElementAt( -
1, "Package" );
    objCSharpNamespace.SetName( "Namespace1" );
    // ... and apply the predefined C namespace stereotype
    objCSharpNamespace.ApplyPredefinedStereotype( 223 ); //
ePredefined_CSharp_namespaceStereotypeOfPackage = 223,

    // create new class within the C namespace
    var objClass      = objCSharpNamespace.InsertPackagedElementAt( -1,
"Class" );
    var objClass2     = objCSharpNamespace.InsertPackagedElementAt( -1,
"Class" );
    var objBaseClass = objCSharpNamespace.InsertPackagedElementAt( -1,
"Class" );
    objClass      .SetName( "MyClass"      );
    objClass2     .SetName( "MyClass2"     );
    objBaseClass. SetName( "MyBaseClass" );

    // set attribute-section "STAThread"
    var objAttributesStereotypeApplication =
objClass.ApplyPredefinedStereotype( 191 );//
ePredefined_CSharp_attributesStereotypeOfClass = 191
    var objSTAThread = objCSharpProfile.FindPredefinedOwnedElement( 185,
true ); //
ePredefined_CSharp_AttributePresetsEnumeration_STAThreadEnumerationLiteral = 185
    objAttributesStereotypeApplication.SetPredefinedTaggedValueAt( -1, 192,
objSTAThread.Name); //
ePredefined_CSharp_attributesStereotypeOfClass_sectionsProperty = 192

    // insert new attribute
    var objProperty = objClass.InsertOwnedAttributeAt( -1 );
    objProperty.SetName( "m_Att" );
    objProperty.Visibility = 2;      // eVisibility_Private = 2
    objProperty.Type = objClass2;

    // insert new operation
    var objOperation = objClass.InsertOwnedOperationAt( -1 );
    objOperation.SetName( "GetAtt" );
    objOperation.Type = objClass2;

    // derive MyClass from MyBaseClass
    objClass.InsertGeneralizationAt( -1, objBaseClass );

    // find the component view package
    var objComponentView = objRootPackage.FindPredefinedOwnedElement( 1,

```

```
false );// ePredefined_ComponentViewPackage = 1

    // create a new component for C 3.0 and set the source code directory,
    where we want to generate the source code
    var objComponent = objComponentView.InsertPackagedElementAt( -1,
"Component" );
    objComponent.CodeLangVersion= 5; //    eCodeLang_CSharp_3_0    = 5,
    objComponent.CodeProjectFileOrDirectory = GetSourceCodeDirectory();
    objComponent.IsCodeProjectFile = false;

    // this component should realize our classes:
    objComponent.InsertRealizationAt( -1, objClass    );
    objComponent.InsertRealizationAt( -1, objClass2    );
    objComponent.InsertRealizationAt( -1, objBaseClass );

    // do not forget to end modification and finish UndoStep
    objDocument.EndModification();
}
catch( err )
{
    // rollback made changes
    objDocument.AbortModification();
    Exit("Error when creating UML model elements");
}

// update code from model
try
{
    // explicitly run a syntax check
    if ( objDocument.CheckProjectSyntax() )
    {
        // get dialog for code <=> model synchronizations and set the wanted
options:
        var objSynchronizationSettingsDlg =
objUModel.Dialogs.SynchronizationSettingsDlg;

        objSynchronizationSettingsDlg.CodeFromModel_Synchronization = 0; //
eSynchronization_Merge = 0
        objSynchronizationSettingsDlg.CodeFromModel_UserDefinedSPLTemplatesOver
rideDefault = true;

        // update code from model
        if
( !objDocument.SynchronizeCodeFromModel( objSynchronizationSettingsDlg ) )
            Exit("Update code from model failed");
        }
        else
            Exit("Syntax check failed");
    }
catch( err )
{
    Exit("Error when updating code from model");
}

// save project
objDocument.SaveAs( GetUMPFilePath() );
```

```

    WScript.Echo("Finished successfully");
}

// if something went wrong (and we did not save the project),
// we also do not want get asked for saving => set ModifiedFlag to false
if ( objDocument != null )
    objDocument.ModifiedFlag = false;

objUModel.Visible = false;    // will shutdown application if it was started by
this script

```

Running the script

The JScript code listed below is available in the file `DocumentAccess.js` located in the JScript folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My Documents/ Altova/UModel2016/UModelExamples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/UModel2016/UModelExamples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

Update Documentation

The JScript listing below shows how to:

- reverse engineer all C# UModelAPI samples If a UMP file does not exist
- create HTML and RTF documentation
- Export as XML and save the *.UMP file

or,

- open the UMP file and synchronize model from code If a UMP file exists
- create HTML and RTF documentation only if some data has been changed

Script listing - UModelUpdateDocumentation.js

The JScript listing below is explained with comments in the code.

```

// #####
// access runing UModel.Application or
// launch new one and access it
// #####

// #####

```

```
// UpdateDocumentation sample
// *) When running the first time (= when no UMP file exists), reverse engineer
all C UModelAPI samples
//   and create HTML and RTF documentation, make XMI export and save UMP file
// *) when UMP file already exists, open it and synchronize model from code
//   create HTML and RTF documentation and XMI export only if something has
been changed (listen to all different UML data events)
// #####

var bRunVisible = true;
var bShowDialogs = bRunVisible && false;

// //////////// global variables ////////////
var objUModel    = null;
var objWshShell  = null;
var objFSO       = null;

var bChangedAnything = false;
var nAddedClasses    = 0;
var nAddedInterfaces = 0;
var nAddedProperties = 0;
var nAddedOperations = 0;

// //////////// Helpers ////////////

function Exit(strErrorText)
{
    WScript.Echo(strErrorText);

    if (objUModel != null)
        objUModel.Quit();

    WScript.Quit(-1);
}

function CreateGlobalObjects ()
{
    // the Shell and FileSystemObject of the windows scripting host often always
    useful
    try
    {
        objWshShell = WScript.CreateObject("WScript.Shell");
        objFSO = WScript.CreateObject("Scripting.FileSystemObject");
    }
    catch(err)
    {
        Exit("Can't create WScript.Shell object");
    }

    // create the UModel connection
    // if there is a running instance of UModel (that never had a connection) -
    use it
    // otherwise, we automatically create a new instance
    try
    {
        objUModel = WScript.GetObject("", "UModel.Application");
    }
}
```

```

        catch(err)
        {
            Exit("Can't access or create UModel.Application");
        }
    }

    // ////////////////////////////////// get different filepaths / ensure folders are created
    // //////////////////////////////////
    function GetScriptPath()
    {
        var path = objUModel.PersonalDataDirectory + "\\UModelExamples\\API\\JScript\\
        \\UpdateDocumentation";

        if ( !objFSO.FolderExists( path ) )
            objFSO.CreateFolder( path );

        return path;
    }

    function GetFilePath( subdir, filename )
    {
        var path = objFSO.BuildPath( GetScriptPath(), subdir );

        if ( !objFSO.FolderExists( path ) )
            objFSO.CreateFolder( path );

        return path + "\\\" + filename;
    }

    function GetUMPFilePath ()    { return GetFilePath( "UMP",
    "UModelAPI.ump" ); }
    function GetXMIFilePath ()    { return GetFilePath( "Output_XMI",
    "UModelAPI.xmi" ); }
    function GetHTMLFilePath()    { return
    GetFilePath( "Output_HTML", "UModelAPI.html"); }
    function GetRTFFilePath ()    { return GetFilePath( "Output_RTF",
    "UModelAPI.rtf" ); }

    // ////////////////////////////////// UML data event
    handlers //////////////////////////////////
    function objRootPackage_OnChanged( objData, strHint )
    {
        bChangedAnything = true;
    }

    // recursively count newly added classes, interfaces, properties and operations
    function CountAddedElements( objNewChild )
    {
        if ( objNewChild != null )
        {
            if ( objNewChild.KindName == "Class"      ) ++nAddedClasses;
            if ( objNewChild.KindName == "Interface" ) ++nAddedInterfaces;
            if ( objNewChild.KindName == "Property"   ) ++nAddedProperties;
            if ( objNewChild.KindName == "Operation"  ) ++nAddedOperations;

            var ownedElements = objNewChild.OwnedElements;
            var itr = new Enumerator( ownedElements );

```

```

        for ( ; !itr.atEnd(); itr.moveToNext() )
            CountAddedElements( itr.item() );
    }
}

function objRootPackage_OnAfterAddChild( objParent, objNewChild )
{
    bChangedAnything = true;

    // recursively count newly added classes, interfaces, properties and
    // operations
    CountAddedElements( objNewChild );
}

function objRootPackage_OnBeforeErase( objData )
{
    bChangedAnything = true;
}

function objRootPackage_OnMoveData( objParent, objChild, bAttach )
{
    bChangedAnything = true;
}

// ////////////////////////////////// MAIN //////////////////////////////////

CreateGlobalObjects();

if ( bRunVisible )
    objUModel.Visible = true;

var objDocument = null;

try
{
    // open document if it exists; create new one otherwise
    var bDocumentExisted = false;

    if ( objFSO.FileExists( GetUMPFilePath() ) )
    {
        objDocument = objUModel.OpenDocument( GetUMPFilePath() );
        bDocumentExisted = true;
    }
    else
    {
        objDocument = objUModel.NewDocument();
        objDocument.SaveAs( GetUMPFilePath() );
    }

    if ( objDocument == null )
        Exit( "Cannot create or open UModel projectfile" );

    // connect to receive _IUMMLDataEvents from the root-package and all its
    // children:
    var objRootPackage = objDocument.RootPackage;
    WScript.ConnectObject (objRootPackage, "objRootPackage_");
}

```

```

// ensure we get *all* events from root-package and *all* children:
objRootPackage.EventFilter = 2 + // eUMLDataEvent_EraseDataOrChild      = 2,
                                8 + // eUMLDataEvent_AddChildOrGrandChild = 8,
                                32 + // eUMLDataEvent_ChangeDataOrChild   = 32,
                                128; // eUMLDataEvent_MoveChildOrGrandChild = 128
if ( bDocumentExisted )
{
    // UModel projectfile already exists => update model from code

    // get dialog for code <=> model synchronizations and set the wanted
options:
    var objSynchronizationSettingsDlg =
objUModel.Dialogs.SynchronizationSettingsDlg;
    objSynchronizationSettingsDlg.ShowDialog = bShowDialogs;

    objSynchronizationSettingsDlg.ModelFromCode_Synchronization = 0; //
eSynchronization_Merge = 0

    // update model from code
    if
( !objDocument.SynchronizeModelFromCode( objSynchronizationSettingsDlg ) )
        Exit("Update model from code failed");
}
else
{
    // UModel projectfile did not exist => newly import code into model

    var objImportSourceDirectoryDlg =
objUModel.Dialogs.ImportSourceDirectoryDlg;
    objImportSourceDirectoryDlg.ShowDialog = bShowDialogs;

    // set source code directory to import
    objImportSourceDirectoryDlg.Directory = objUModel.PersonalDataDirectory +
"\\UModelExamples\\IDEPlugIn";
    objImportSourceDirectoryDlg.ProcessSubdirectories = true;
    // set source code language to import (C 3.0)
    objImportSourceDirectoryDlg.Language      = 5; // eCodeLang_CSharp_3_0
= 5
    objImportSourceDirectoryDlg.Synchronization = 0; // eSynchronization_Merge
= 0
    // import in a new package
    objImportSourceDirectoryDlg.ImportInNewPackage = true;

    objImportSourceDirectoryDlg.DiagramGeneration = true;

    // content diagram generation settings
    objImportSourceDirectoryDlg.Content_GenerateSingleDiagram      =
true;
    objImportSourceDirectoryDlg.Content_GenerateDiagramPerPackage  =
true;
    objImportSourceDirectoryDlg.Content_ShowNestedClassifiersSeparately
= false;
    objImportSourceDirectoryDlg.Content_ShowAnonymousBoundElements  =
false;
    objImportSourceDirectoryDlg.Content_HyperlinkPackagesToDiagrams  =
true;
    objImportSourceDirectoryDlg.Content_ShowAttributesCompartment    =

```



```

true;
    objImportSourceDirectoryDlg.Content_ShowOperationsCompartment      =
true;
    objImportSourceDirectoryDlg.Content_ShowNestedClassifiersCompartment  =
false;
    objImportSourceDirectoryDlg.Content_ShowEnumerationLiteralsCompartment
= true;
    objImportSourceDirectoryDlg.Content_ShowTaggedValues                = true;
    objImportSourceDirectoryDlg.Content_Autolayout                      = 1; //
eDiagramLayout_Hierarchic = 1
    // open diagrams that autolayout is done:
    objImportSourceDirectoryDlg.Content_OpenDiagrams                    = true;

    // package dependency diagram generation settings (disabled)
    objImportSourceDirectoryDlg.PackageDependency_GenerateDiagram = false;

    // import source directory
    if ( !objDocument.ImportSourceDirectory( objImportSourceDirectoryDlg ) )
    {
        // also delete newly created (empty) UMP file that source code
        // directory import is retried the next time
        objFSO.DeleteFile( GetUMPFilePath() );
        Exit( "Error on importing source directory" );
    }
}

    // disconnect from getting root-package events
    WScript.DisconnectObject( objRootPackage );
}
catch( err )
{
    // also delete newly created (empty) UMP file that source code directory
    // import is retried the next time
    objFSO.DeleteFile( GetUMPFilePath() );
    Exit( "Error on importing source directory" );
}

//if something has changed, update the outputs:
if ( bChangedAnything )
{
    try
    {
        // make XMI export for UML2.1.2
        var objIExportXMIFileDlg = objUModel.Dialogs.ExportXMIFileDlg;
        objIExportXMIFileDlg.ShowDialog = bShowDialogs;
        objIExportXMIFileDlg.XMIFile      = GetXMIFilePath();
        objIExportXMIFileDlg.PrettyPrintXMIOutput = true;
        objIExportXMIFileDlg.ExportUUIDs   = true;
        objIExportXMIFileDlg.ExportExtensions = true;
        objIExportXMIFileDlg.ExportDiagrams = true;
        objIExportXMIFileDlg.XMIType       = 1; // eXMI21ForUML212 = 1

        // export to XMI file:
        if ( !objDocument.ExportToXMIFile( objIExportXMIFileDlg ) )
        {
            // error on XMI generation
        }
    }
}

```

```

    }
    catch( err )
    {
        // error on XMI generation
    }

    try
    {
        var objIDocumentationGenerationDlg =
objUModel.Dialogs.GenerateDocumentationDlg;
        objIDocumentationGenerationDlg.ShowDialog = bShowDialogs;

        objIDocumentationGenerationDlg.GenerateLinksToLocalFiles = 1; //
eDocumentationFilePath_RelativeToResultFile    = 1
        objIDocumentationGenerationDlg.SplitOutputToMultipleFiles = true;
        objIDocumentationGenerationDlg.ShowResultFileAfterGeneration = true;

        objIDocumentationGenerationDlg.Details_SelectAll();
        // show up to 10 base class/interface hierarchies
        objIDocumentationGenerationDlg.Details_HierarchyDiagramNestingDepthUp =
10;
        // only show directly derived classes/interfaces
        objIDocumentationGenerationDlg.Details_HierarchyDiagramNestingDepthDown =
1;
        // keep hierarchy diagram as small as possible => expand each element only
once
        objIDocumentationGenerationDlg.Details_HierarchyDiagramExpandItemsOnlyOnce
= true;

        objIDocumentationGenerationDlg.Include_SelectAllDiagrams();
        objIDocumentationGenerationDlg.Include_SelectNoElements();
        objIDocumentationGenerationDlg.Include_Index = true;
        objIDocumentationGenerationDlg.Include_IncludedSubprojects = false;
        objIDocumentationGenerationDlg.Include_NamedElementsOnly = true;
        objIDocumentationGenerationDlg.Include_UnknownExternals = false;

        var objIncludeElements = objIDocumentationGenerationDlg.Include_Elements;
        var itrIncludeElements = new Enumerator( objIncludeElements );
        for ( ; !itrIncludeElements.atEnd(); itrIncludeElements.moveNext() )
        {
            var objElemSel = itrIncludeElements.item();

            if ( objElemSel.KindName == "Class"           ||
                objElemSel.KindName == "Interface"       ||
                objElemSel.KindName == "Enumeration"     ||
                objElemSel.KindName == "Operation"       ||
                objElemSel.KindName == "Package"         )
            {
                objElemSel.Selection = true;
            }
        }

        // generate HTML documentation (with PNG pictures)
        objIDocumentationGenerationDlg.OutputFormat = 0; //
eDocumentationOutputFormat_HTML    = 0
        objIDocumentationGenerationDlg.DiagramImageFormat = 0; //
eOutputImageFormat_PNG    = 0

```

```

objIDocumentationGenerationDlg.EmbedDiagrams = false;
if ( !objDocument.GenerateDocumentation( objIDocumentationGenerationDlg,
GetHTMLFilePath() ) )
{
    // error on HTML documentation generation
}

// generate RTF documentation (with embeded EMF pictures)
objIDocumentationGenerationDlg.ShowDialog = false; // don't show dialog
again
objIDocumentationGenerationDlg.OutputFormat = 2; //
eDocumentationOutputFormat_RTF = 2
objIDocumentationGenerationDlg.DiagramImageFormat = 1; //
eOutputImageFormat_EMF = 1
objIDocumentationGenerationDlg.EmbedDiagrams = true;
if ( !objDocument.GenerateDocumentation( objIDocumentationGenerationDlg,
GetRTFFilePath() ) )
{
    // error on RTF documentation generation
}
}
catch( err )
{
    // error on documentation generation
}

// show the number of newly added classes, interfaces, properties and
operations
if ( bRunVisible )
{
    WScript.Echo( "Added classes: " + nAddedClasses +
        "\nAdded interfaces: " + nAddedInterfaces +
        "\nAdded properties: " + nAddedProperties +
        "\nAdded operations: " + nAddedOperations );
}
}
else
{
    if ( bRunVisible )
        WScript.Echo( "Nothing has changed" );
}

// always save document (although it's not really necessary when nothing has
been changed)
objDocument.Save();

if ( bRunVisible )
    objUModel.Visible = false; // will shutdown application if it was started
by this script

```

Running the script

The JScript code listed below is available in the file `DocumentAccess.js` located in the JScript folder of the API Examples folder:

Windows XP	C:/Documents and Settings/<username>/My

	Documents/ Altova/UModel2016/UModelExamples/API/
Windows Vista, Windows 7	C:/Users/<username>/Documents/ Altova/UModel2016/UModelExamples/API/

To run the script, start it from a command prompt window or from Windows Explorer.

20.4.2 Usage Examples

The automation interface of UModel can be used in many different ways and accessed from different programming languages. In this section you will find sample code that show the capabilities of the new UModel API interfaces.

These samples should make it easier for you to get started with your own project. Additional code snippets can be found in the [How to...](#) section of the [Overview](#).

The listed code samples are:

UModel IDE plug-in samples (written in C#):

- [Styles](#) sets multiple styles for selected diagram elements
- [CSharpDelegate](#) inserts a new C# delegate at the current scroll position on the active diagram window
- [DefaultPrefix](#) automatically sets a prefix when new attributes or enumeration literals are added to the UModel project

UModel IDE plug-in ActiveX sample (written in C#):

- [StatisticsActiveX](#) counts the elements of the different element kinds after every data modification (transaction) and shows the result in an ActiveX control

Scripting samples (written in JScript):

- [CreateCode](#) creates a new UModel project, creates some classes and generates code
- [UpdateDocumentation](#) reverse engineers all UModel API samples and creates HTML and RTF documentation as well as an XML export of the project

Deployment sample:

- [SetupStyles](#) creates an installer for deploying the [Styles](#) IDE plug-in sample

Styles (C#)

The following C# IDE plug-in sample sets multiple styles for selected diagram elements (if style is available and not already set).

Keywords: UModel IDE plug-in, active document, active diagram, diagram selection, elements on diagrams, undo-transaction handling, styles, styles enumeration, verification of API version

File: UModelExamples\IDEPlugIn\Styles\Styles.cs:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows.Forms;
using UModelLib;
using UModelPlugInLib;

/*
 * Styles sample
 * set following styles for selected diagram elements
 *     Fill Color
 *     Header Gradient Begin Color
```

```

*      Header Gradient End Color
* if style is available and not already set
*/

namespace Styles
{
    public class UModelStyles : UModelPlugInLib.IUModelPlugIn
    {
        bool m_bPlugInVersionOK = true; // verify if UModel-API has
        been changed in a way that a recompile of this plug-in is recommended

        #region helpers

        protected string GetPlugInPath()
        {
            string sDLLPath =
System.Reflection.Assembly.GetExecutingAssembly().Location;
            return System.IO.Path.GetDirectoryName(sDLLPath);
        }

        #endregion

        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "Styles sample Plug-in for UModel;This Plug-in
demonstrates how to change several styles of the selected diagram
elements.";
        }

        public string GetUIModifications()
        {
            try
            {
                string sPath = GetPlugInPath();

                System.IO.StreamReader myFile = new
System.IO.StreamReader(sPath + "\\config.xml");
                string sRet = myFile.ReadToEnd();
                myFile.Close();

                // this replaces the token "***path**" from the XML
file with
                // the actual installation path of the plug-in to get
the image file
                return sRet.Replace("***path**", sPath);
            }
            catch (System.Exception ex)
            {
                MessageBox.Show("Error in GetUIModifications:" +
ex.Message);
                throw ex;
            }
        }
    }
}

```

```
    }
}

public void OnInitialize(object pUModel)
{
    // before processing DDE or batch commands
}

public void OnRunning(object pUModel)
{
    // DDE or batch commands are processed; application is
    fully initialized

    // verify if UModel-API has been changed in a way that a
    recompile of this plug-in is recommended:
    IApplication iApp = (IApplication)pUModel;
    if (iApp == null || iApp.APIMajorVersion != 2) // this
    plug-in was compiled for API major version '2'!
    {
        MessageBox.Show("'Styles': This Plug-in has been made
        with a previous version of the UModel-API and should be recompiled.
        \nDisabled Plug-in commands in the meantime.");
        m_bPlugInVersionOK = false;
    }
}

public void OnShutdown(object pUModel)
{
    // application will shutdown; release all unused objects
    GC.Collect();
}

public UModelUpdateAction OnUpdateCommand(int nID, object
pUModel)
{
    UModelUpdateAction action =
    UModelUpdateAction.UModelUpdateAction_Disable;
    if (!m_bPlugInVersionOK)
        return action;

    // check for "fill red"
    if (nID == 3 || nID == 6)
        action = OnUpdateSetStyles((IApplication)pUModel);

    // check for "fill green"
    if (nID == 4 || nID == 7)
        action = OnUpdateSetStyles((IApplication)pUModel);

    // release unused objects
    GC.Collect();

    return action;
}
```

```

public void OnCommand(int nID, object pUModel)
{
    if (!m_bPlugInVersionOK)
        return;

    // fill red
    if (nID == 3 || nID == 6)
        OnSetStyles((IApplication)pUModel, "red");

    // fill green
    if (nID == 4 || nID == 7)
        OnSetStyles((IApplication)pUModel, "green");

    // release unused objects
    GC.Collect();
}

#endregion

#region SetStyles // set styles of selected diagram elements

UModelUpdateAction OnUpdateSetStyles(IApplication pUModel)
{
    if (pUModel == null)
        return UModelUpdateAction.UModelUpdateAction_Disable;

    // get the active document of the application
    IDocument iDoc = pUModel.ActiveDocument;
    if (iDoc == null)
        return UModelUpdateAction.UModelUpdateAction_Disable;

    // get the active diagram window
    IDiagramWindow iActiveDiagram = iDoc.ActiveDiagramWindow;
    if ( iActiveDiagram == null )
        return UModelUpdateAction.UModelUpdateAction_Disable;

    // get the selected elements on the active diagram
    IUMLDataList iSelection =
iActiveDiagram.SelectedGuiElements;
    if ( iSelection == null )
        return UModelUpdateAction.UModelUpdateAction_Disable;

    // search all selected elements, if at least one has one
of the styles to change
    foreach ( IUMLGuiElement iSelGuiElement in iSelection )
    {
        // verify if it is a GuiVisibleElement (with Styles)
and if it may be modified
        if ( iSelGuiElement is IUMLGuiVisibleElement &&
iSelGuiElement.IsEditable )
        {
            IUMLGuiVisibleElement iVisGuiElement =

```



```

(IUMLGuiVisibleElement) iSelGuiElement;

        if
        ( iVisGuiElement
        .Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_FillColor) != null
        ||
                iVisGuiElement
        .Styles
        .GetStyle
        (ENUMUMLGuiStyleKind.eUMLGuiStyle_HeaderGradientBeginColor) != null ||
                iVisGuiElement
        .Styles
        .GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_HeaderGradientEndColor) !=
        null )
        {
                return
        UModelUpdateAction.UModelUpdateAction_Enable;
        }
    }

    // nothing found => disable command
    return UModelUpdateAction.UModelUpdateAction_Disable;
}

public void OnSetStyles(IApplication pUModel, string sColor)
{
    if (pUModel == null)
        return;

    // get the active document of the application
    IDocument iDoc = pUModel.ActiveDocument;
    if (iDoc == null)
        return;

    // get the active diagram window
    IDiagramWindow iActiveDiagram = iDoc.ActiveDiagramWindow;
    if (iActiveDiagram == null)
        return;

    // get the selected elements on the active diagram
    IUMLDataList iSelection =
iActiveDiagram.SelectedGuiElements;
    if (iSelection == null)
        return;

    try
    {
        // make all modifications within one UndoStep; start
        modification here
        if (!iDoc.BeginModification())
            return;
    }
}

```

```

        // search all selected elements, and change the style
        if the wanted value is not already used (directly applied or through
        style-chain)
        foreach (IUMLGuiElement iSelGuiElement in iSelection)
        {
            // verify if it is a GuiVisibleElement (with
            Styles) and if it may be modified
            if (iSelGuiElement is IUMLGuiVisibleElement &&
            iSelGuiElement.IsEditable )
            {
                IUMLGuiVisibleElement iVisGuiElement =
                (IUMLGuiVisibleElement)iSelGuiElement;

                // set Fill Color if possible and not already
                set
                IUMLGuiStyle iStyle =
                iVisGuiElement
                .Styles.GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_FillColor);
                if (iStyle != null && iStyle.UsedValue !=
                sColor)
                    iStyle.Value = sColor;

                // set Header Gradient Begin Color if possible
                and not already set
                iStyle =
                iVisGuiElement
                .Styles
                .GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_HeaderGradientBeginColor);
                if (iStyle != null && iStyle.UsedValue !=
                sColor)
                    iStyle.Value = sColor;

                // set Header Gradient End Color if possible
                and not already set
                iStyle =
                iVisGuiElement
                .Styles
                .GetStyle(ENUMUMLGuiStyleKind.eUMLGuiStyle_HeaderGradientEndColor);
                if (iStyle != null && iStyle.UsedValue !=
                sColor)
                    iStyle.Value = sColor;
            }
        }

        // do not forget to end modification and finish
        UndoStep
        iDoc.EndModification();
    }
    catch( System.Exception )
    {
        // rollback made changes
        iDoc.AbortModification();
    }

```

```

        // add error handling
    }
}

#endregion
}
}

```

SetupStyles (Deployment)

The SetupStyles sample creates an installer for deploying the [Styles](#) IDE plug-in sample.

The setup installs all necessary files, registers the IDE plug-in for COM and UModel on your target system, that the [Styles](#) plug-in is automatically loaded when UModel is started the next time.

File: UModelExamples\IDEPlugIn\Styles\SetupStyles\SetupStyles.sln

CSharpDelegate (C#)

The following C# IDE plug-in sample inserts a new C# delegate at the top/left corner of the active diagram window (if this diagram is inside a C# namespace root)

Keywords: UModel IDE plug-in, active diagram; diagram scroll positions, setting diagram selection, check for C# namespace root; add new data on diagram; applying stereotypes and setting tagged values; predefined UModel elements, undo-transaction handling

File: UModelExamples\IDEPlugIn\CSharpDelegate\UModelCSharpDelegate.cs:

```

using System;
using System.Collections.Generic;
using System.Text;
using UModelLib;
using UModelPlugInLib;

/*
 * CSharp delegate sample
 * add a new CSharp delegate on the top/left corner of the active
class diagram if possible
 * (i.e. when diagram is inside a C# root namespace)
 */

namespace CSharpDelegate
{
    public class UModelCSharpDelegate : UModelPlugInLib.IUModelPlugIn
    {
        #region helpers

        protected string GetPlugInPath()
        {
            string sDLLPath =
System.Reflection.Assembly.GetExecutingAssembly().Location;
            return System.IO.Path.GetDirectoryName(sDLLPath);
        }
    }
}

```

```

        #endregion

        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "CSharpDelegate sample Plug-in for UModel;This
Plug-in demonstrates how to create a new CSharp delegate on a class
diagram.";
        }

        public string GetUIModifications()
        {
            try
            {
                string sPath = GetPlugInPath();

                System.IO.StreamReader myFile = new
System.IO.StreamReader(sPath + "\\config.xml");
                string sRet = myFile.ReadToEnd();
                myFile.Close();

                // this replaces the token "***path**" from the XML
file with
                // the actual installation path of the plug-in to get
the image file
                return sRet.Replace("***path**", sPath);
            }
            catch (System.Exception ex)
            {
                System.Windows.Forms.MessageBox.Show("Error in
GetUIModifications:" + ex.Message);
                throw ex;
            }
        }

        public void OnInitialize(object pUModel)
        {
            // before processing DDE or batch commands
        }

        public void OnRunning(object pUModel)
        {
            // DDE or batch commands are processed; application is
fully initialized
        }

        public void OnShutdown(object pUModel)
        {
            // application will shutdown; release all unused objects
GC.Collect();
        }
    }

```

```
        public UModelUpdateAction OnUpdateCommand(int nID, object
pUModel)
        {
            UModelUpdateAction action =
UModelUpdateAction.UModelUpdateAction_Disable;

            // check if we can add a new CSharpDelegate on the active
diagram
            if (nID == 3 || nID == 4)
                action = OnUpdateAddNewCSharpDelegate((IApplication)
pUModel);

            // release unused objects
GC.Collect();

            return action;
        }

        public void OnCommand(int nID, object pUModel)
        {
            // add a new CSharpDelegate on the active diagram
            if (nID == 3 || nID == 4)
                OnAddNewCSharpDelegate((IApplication)pUModel);

            // release unused objects
GC.Collect();
        }

        #endregion

        #region AddNewCSharpDelegate // add new CSharp delegate on
active diagram

        UModelUpdateAction OnUpdateAddNewCSharpDelegate(IApplication
pUModel)
        {
            if (pUModel == null)
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the active document of the application
IDocument iDoc = pUModel.ActiveDocument;
            if (iDoc == null)
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the active diagram window
IDiagramWindow iActiveDiagram = iDoc.ActiveDiagramWindow;
            if ( iActiveDiagram == null )
                return UModelUpdateAction.UModelUpdateAction_Disable;

            // get the UML diagram of the diagram window
IUMLGuiDiagram iUMLDiagram = iActiveDiagram.Diagram;

            // check if it is a class diagram
```

```

        if ( !( iUMLDiagram is IUMLGuiClassDiagram) )
            return UModelUpdateAction.UModelUpdateAction_Disable;

        // verify if the diagram may be modified
        if ( !iUMLDiagram.IsEditable )
            return UModelUpdateAction.UModelUpdateAction_Disable;

        // get the UML element, which "owns" the class diagram
        IUMLElement iDiagramOwner = iUMLDiagram.LinkedOwner;
        if (iDiagramOwner == null)
            return UModelUpdateAction.UModelUpdateAction_Disable;

        // verify if we are inside a CSharp namespace root
        (otherwise adding a CSharp delegate makes no sense)
        IUMLElement iFindNamespaceRoot = iDiagramOwner;
        while( iFindNamespaceRoot != null)
        {
            if ( iFindNamespaceRoot is IUMLPackage)
            {
                IUMLPackage iPackage = (IUMLPackage)
iFindNamespaceRoot;
                if
                ( iPackage.IsCodeLangNamespaceRoot( ENUMCodeLang.eCodeLang_CSharp ) )
                    return
UModelUpdateAction.UModelUpdateAction_Enable;
            }

            iFindNamespaceRoot = iFindNamespaceRoot.Owner;
        }

        // nothing found => disable command
        return UModelUpdateAction.UModelUpdateAction_Disable;
    }

    public void OnAddNewCSharpDelegate(IApplication pUModel)
    {
        if (pUModel == null)
            return;

        // get the active document of the application
        IDocument iDoc = pUModel.ActiveDocument;
        if (iDoc == null)
            return;

        // get the active diagram window
        IDiagramWindow iActiveDiagram = iDoc.ActiveDiagramWindow;
        if (iActiveDiagram == null)
            return;

        // get the UML diagram of the diagram window
        IUMLGuiDiagram iUMLDiagram = iActiveDiagram.Diagram;

        // get the CSharp profile

```

```

        IUMLProfile iCSharpProfile = (IUMLProfile)
iDoc
.RootPackage
.FindPredefinedOwnedElement
(ENUMUMLPredefinedElement.ePredefined_CSharp_Profile, false);
        if ( iCSharpProfile == null)
            return;

        try
        {
            // make all modifications within one UndoStep; start
modification here
            if (!iDoc.BeginModification())
                return;

            // get top left corner of the visible diagram area
            int nInsertPosX = iActiveDiagram.ScrollPosX;
            int nInsertPosY = iActiveDiagram.ScrollPosY;

            // add new class on diagram
            IUMLGuiNodeLink iClassNode =
iUMLDiagram.AddUMLElement("Class", nInsertPosX + 100, nInsertPosY +
100);

            IUMLClass iClass = (IUMLClass) iClassNode.Element;
            // use SetName (instead of Name) that UModel
automatically generates a valid, unique name starting with
"NewDelegate"
            iClass.SetName("NewDelegate");

            // set the CSharp 'delegate' stereotype
            iClass
.ApplyPredefinedStereotype
( ENUMUMLPredefinedElement
.ePredefined_CSharp_delegateStereotypeOfClass );

            // set attribute-section "STAThread"
            IUMLStereotypeApplication iStereotypeApp =
iClass
.ApplyPredefinedStereotype
(ENUMUMLPredefinedElement
.ePredefined_CSharp_attributesStereotypeOfClass);
            IUMLEnumerationLiteral iSTAThread =
(IUMLEnumerationLiteral)
iCSharpProfile
.FindPredefinedOwnedElement
(ENUMUMLPredefinedElement
.ePredefined_CSharp_AttributePresetsEnumeration_STAThreadEnumerationLi
teral, true);
            iStereotypeApp.SetPredefinedTaggedValueAt(-1,
ENUMUMLPredefinedElement
.ePredefined_CSharp_attributesStereotypeOfClass_sectionsProperty,
iSTAThread.Name);

```

```

        // add delegate operation:
        IUMLOperation iOperation =
iClass.InsertOwnedOperationAt(-1);
        iOperation.SetName( "delegate");

        // per default set operation-return type "void"
        IUMLPrimitiveType iTypeVoid = (IUMLPrimitiveType)
iCSharpProfile
.FindPredefinedOwnedElement
(ENUMUMLPredefinedElement.ePredefined_CSharp_voidPrimitiveType, true);
        iOperation.Type = iTypeVoid;

        // do not forget to end modification and finish
UndoStep
        iDoc.EndModification();

        // at last focus newly inserted delegate on the
diagram:
        iActiveDiagram.SelectGuiElement(iClassNode, true);
    }
    catch( System.Exception )
    {
        // rollback made changes
        iDoc.AbortModification();

        // add error handling
    }
}

#endregion
}
}

```

DefaultPrefix (C#)

The following C# IDE plug-in sample automatically sets a prefix when new attributes or enumeration literals are added to your UModel project.

Keywords: UModel IDE plug-in, application events, UMLData events, UMLData event filter, connection points and delegates

File: UModelExamples\IDEPlugIn\DefaultPrefix\DefaultPrefix.cs:

```

using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using System.Runtime.InteropServices.ComTypes;
using UModelLib;
using UModelPlugInLib;

/*
 * DefaultPrefix sample

```



```
* listen for newly added UML data and
* set the prefix of properties ('m_') and EnumerationLiterals ('k_')
* if the corresponding option is turned on
*/

namespace DefaultPrefix
{
    /* UModelDefaultPrefix is the main class of this plugin and
    implements UModelPlugInLib.IUModelPlugIn
    * it is also responsible for attaching/detaching
    UModelApplicationEvents to/from UModels IApplication interface
    * and implements the handling of turning on/off the whole
    "SetPrefix" functionality
    */
    public class UModelDefaultPrefix : UModelPlugInLib.IUModelPlugIn
    {
        // variable which defines whether "SetPrefix" functionality is
        turned on or off
        bool m_bSetPrefix = true;

        // reference to UModelApplicationEvents; is only used when
        "SetPrefix" functionality is turned on (to reduce overhead in the
        other case)
        UModelApplicationEvents m_AppEvents = null;

        #region helpers

        protected string GetPlugInPath()
        {
            string sDLLPath =
System.Reflection.Assembly.GetExecutingAssembly().Location;
            return System.IO.Path.GetDirectoryName(sDLLPath);
        }

        #endregion

        // create UModelApplicationEvents and attach it to
        IApplication
        protected void AttachAppEvents( IApplication iUModelApp )
        {
            if (m_AppEvents == null && iUModelApp != null)
            {
                m_AppEvents = new UModelApplicationEvents();
                m_AppEvents.Attach(iUModelApp);
            }
        }

        // detach UModelApplicationEvents;
        protected void DetachAppEvents()
        {
            if (m_AppEvents != null)
            {
                m_AppEvents.Detach();
            }
        }
    }
}
```

```

        m_AppEvents = null;
    }
}

#region IUModelPlugIn Members

public string GetDescription()
{
    return "DefaultPrefix sample Plug-in for UModel;This Plug-
in demonstrates how to attach to several callback interfaces and how
to add a prefix to newly inserted elements.";
}

public string GetUIModifications()
{
    try
    {
        string sPath = GetPlugInPath();

        System.IO.StreamReader myFile = new
System.IO.StreamReader(sPath + "\\config.xml");
        string sRet = myFile.ReadToEnd();
        myFile.Close();

        // this replaces the token "***path**" from the XML
file with
        // the actual installation path of the plug-in to get
the image file
        return sRet.Replace("***path**", sPath);
    }
    catch (System.Exception ex)
    {
        System.Windows.Forms.MessageBox.Show("Error in
GetUIModifications:" + ex.Message);
        throw ex;
    }
}

public void OnInitialize(object pUModel)
{
    // before processing DDE or batch commands
}

public void OnRunning(object pUModel)
{
    // DDE or batch commands are processed; application is
fully initialized
    // and we can attach UModelApplicationEvents
    AttachAppEvents( (IApplication)pUModel );
}

public void OnShutdown(object pUModel)
{

```

```

        // detach UModelApplicationEvents; stop receiving events
        DetachAppEvents();

        // application will shutdown; release all unused objects
        GC.Collect();
    }

    public UModelUpdateAction OnUpdateCommand(int nID, object
pUModel)
    {
        UModelUpdateAction action =
UModelUpdateAction.UModelUpdateAction_Disable;

        // check if automatically setting the prefix is turned on:
        if (nID == 3 || nID == 4)
        {
            action = UModelUpdateAction.UModelUpdateAction_Enable;

            if (m_bSetPrefix)
                action |=
UModelUpdateAction.UModelUpdateAction_Check;
        }

        // release unused objects
        //GC.Collect(); not necessary since we do not access
objects here

        return action;
    }

    public void OnCommand(int nID, object pUModel)
    {
        // toggle automatically setting the prefix:
        if (nID == 3 || nID == 4)
            m_bSetPrefix = !m_bSetPrefix;

        // attach UModelApplicationEvents when "SetPrefix"
functionality is turned on; detach otherwise
        if (m_bSetPrefix)
            AttachAppEvents((IApplication)pUModel);
        else
            DetachAppEvents();

        // release unused objects
        GC.Collect();
    }

    #endregion
}

/* UModelApplicationEvents is an eventhandler to receive
_IApplicationEvents

```

```

    * that we know when UModel documents are opened or closed
    * and that we can Attach/Detach UModelDataEvents
    * We are interested in all _IApplicationEvents and use a
    connectionpoint to connect to all these events
    */
    public class UModelApplicationEvents :
    UModelLib._IApplicationEvents
    {
        // connection point to _IApplicationEvents
        System.Runtime.InteropServices.ComTypes.IConnectionPoint
    m_cpApplicationEvents = null;
        // connection cookie
        int m_nApplicationEventsCookie = 0;
        // we always hold a reference to UModelDataEvents
        UModelDataEvents m_UMLDataEvents = new UModelDataEvents();

        public void Attach(IApplication iApp)
        {
            if (m_cpApplicationEvents == null && iApp != null)
            {
                // find connection point of _IApplicationEvents
                IConnectionPointContainer icpc =
                (IConnectionPointContainer)iApp;
                Guid IID = typeof(UModelLib._IApplicationEvents).GUID;
                icpc.FindConnectionPoint(ref IID, out
                m_cpApplicationEvents);

                // advise UModelApplicationEvents as sink for
                _IApplicationEvents
                m_cpApplicationEvents.Advise(this, out
                m_nApplicationEventsCookie);

                // also attach UModelDataEvents to the current
                document and start receiving events there
                m_UMLDataEvents.Attach(iApp.ActiveDocument);
            }
        }
        public void Detach()
        {
            if (m_cpApplicationEvents != null)
            {
                // also detach UModelDataEvents and stop receiving
                events there
                m_UMLDataEvents.Detach();

                // terminate established connection to
                _IApplicationEvents
                m_cpApplicationEvents
                .Unadvise(m_nApplicationEventsCookie);
                m_cpApplicationEvents = null;
            }
        }
    }

```

```

        #region _IApplicationEvents Members
        public void OnNewDocument(Document ipDocument)
        {
            Debug.WriteLine("UModelApplicationEvents.OnNewDocument " +
ipDocument.Name);
            // a new document has been created in UModel => (re-)
connect UModelDataEvents
            m_UMLDataEvents.Attach(ipDocument);
        }

        public void OnDocumentOpened(Document ipDocument)
        {
            Debug.WriteLine("UModelApplicationEvents.OnDocumentOpened
" + ipDocument.Name);
            // a document has been opened in UModel => (re-)connect
UModelDataEvents
            m_UMLDataEvents.Attach(ipDocument);
        }

        public void OnDocumentClosed(Document ipDocument)
        {
            Debug.WriteLine("UModelApplicationEvents.OnDocumentClosed
" + ipDocument.Name);
            // document has been closed in UModel => disconnect
UModelDataEvents
            m_UMLDataEvents.Detach();
        }

        public void OnShutdown()
        {
            Debug.WriteLine("UModelApplicationEvents.OnShutdown");
        }

        #endregion
    }

    /* UModelDataEvents is an eventhandler to receive _IUMLDataEvents
    * from the root-package and all its children.
    * We are only interested in 'OnAfterAddChild' events, so we use a
    delegate to connect to this event.
    */
    public class UModelDataEvents : UModelLib._IUMLDataEvents
    {
        // hold a reference to the current UML Root package; this is
        safe as long as we listen to when it is deleted
        protected UMLData m_RootPackage = null;

        // attach this eventhandler to the root-package of the
        (current) document
        public void Attach(IDocument iDoc)
        {
            if (m_RootPackage == null && iDoc != null &&

```

```

iDoc.RootPackage != null)
{
    // hold a reference to the current UML Root package
    m_RootPackage = (UMLData)iDoc.RootPackage;

    // ensure we get 'OnAfterAddChild' events for *any*
    added child of the rootpackage
    // (added to the root-package or one of its children)
    m_RootPackage.EventFilter = (int)
ENUMUMLDataEventFilter.eUMLDataEvent_AddChildOrGrandChild;
    // ensure we get informed when m_RootPackage (and only
    itself; we do not care about its children) is deleted
    m_RootPackage.EventFilter |= (int)
ENUMUMLDataEventFilter.eUMLDataEvent_EraseData;

    // we are only interested in 'OnAfterAddChild' and
    'OnBeforeErase' events so use and connect the delegates
    m_RootPackage.OnAfterAddChild += new
_IUMLDataEvents_OnAfterAddChildEventHandler(OnAfterAddChild);
    m_RootPackage.OnBeforeErase += new
_IUMLDataEvents_OnBeforeEraseEventHandler(OnBeforeErase);
}
}

// detach eventhandler from the current UML Root package
public void Detach()
{
    if (m_RootPackage != null)
    {
        m_RootPackage.OnAfterAddChild -= OnAfterAddChild;
        m_RootPackage.OnBeforeErase -= OnBeforeErase;
        m_RootPackage = null;

        // release unused objects
        GC.Collect();
    }
}

#region _IUMLDataEvents Members

public void OnAfterAddChild(UMLData ipUMLParent, UMLData
ipUMLChild)
{
    if (ipUMLParent == null || ipUMLChild == null)
        return;

    Debug.WriteLine("UModelDataEvents.OnAfterAddChild " +
GetName(ipUMLChild) + " to " + GetName(ipUMLParent));

    // verify if newly added child is of interesting kind:
    bool bIsEnumerationLiteral = (ipUMLChild is
IUMLEnumerationLiteral);
    bool bIsProperty = (ipUMLChild is IUMLProperty);

```

```
        if (bIsProperty || bIsEnumerationLiteral)
        {
            try
            {
                // check if child was added by undo/redo
                // (we are not allowed to modify anything during
Undo/Redo !!)
                IDocument iDoc = (IDocument)ipUMLChild.Parent;
                if (!iDoc.IsInUndoRedo)
                {
                    // we only make one single modification here
                    // no need to use iDoc.BeginModification /
iDoc.EndModification in this case

                    // get the wanted prefix for the element kind
                    string sPrefix = null;

                    if (bIsProperty)
                        sPrefix = "m_";
                    if (bIsEnumerationLiteral)
                        sPrefix = "k_";

                    IUMLNamedElement iNamedChild =
(IUMLNamedElement)ipUMLChild;

                    // set prefix only if not already set:
                    if (sPrefix != null &&
!iNamedChild.Name.StartsWith(sPrefix))
                    {
                        // use SetName (instead of Name) that
UModel automatically generates a valid, unique name starting with
'sPrefix + iNamedChild.Name'
                        iNamedChild.SetName(sPrefix +
iNamedChild.Name);
                    }
                }
            }
            catch (System.Exception e)
            {
                Debug.WriteLine("EXCEPTION: " + e.Message);
            }
        }

        // release unused objects
        GC.Collect();
    }

    public void OnBeforeErase(IUMLData ipUMLData)
    {
        if (ipUMLData != null && m_RootPackage != null &&
ipUMLData.IsSameUMLData((IUMLData)m_RootPackage)) // should always be
        {

```

```

        // Detach ourself, since the UML data of m_RootPackage
        has been deleted in UModel and we may not access it anymore
        Detach();
    }
}

public void OnChanged(IUMLData ipUMLData, string strHint)
{
    // unused
}

public void OnMoveData(IUMLData ipUMLParent, IUMLData
ipUMLChild, bool bAttach)
{
    // unused
}

#endregion

protected string GetName(IUMLData iUMLData)
{
    if (iUMLData is IUMLNamedElement)
        return ((IUMLNamedElement) iUMLData).Name;

    return "";
}
}
}

```

CreateCode (JScript)

The following JScript sample creates a new UModel project, creates some classes and generates code.

Topics: loading include files, saving UModel projectfiles, namespace root handling, using predefined elements and stereotypes, applying profiles and stereotypes, setting tagged values, creating new UML elements, code from model synchronization (forward engineering)

File: UModelExamples\API\CreateCode\UModelCreateCode.js:

```

// #####
// access runing UModel.Application or
// launch new one and access it
// #####

// #####
// CreateCode sample
// shows forward engineering from scratch
// it creates some coding elements in a new UModel project and
// generates code (saving the project afterwards)
// #####

```



```
// //////////// global variables ////////////
var objUModel = null;
var objWshShell = null;
var objFSO = null;

// //////////// Helpers ////////////

function Exit(strErrorText)
{
    WScript.Echo(strErrorText);

    if (objUModel != null)
        objUModel.Quit();

    WScript.Quit(-1);
}

function CreateGlobalObjects ()
{
    // the Shell and FileSystemObject of the windows scripting host
    // often always useful
    try
    {
        objWshShell = WScript.CreateObject("WScript.Shell");
        objFSO =
WScript.CreateObject("Scripting.FileSystemObject");
    }
    catch(err)
    {
        Exit("Can't create WScript.Shell object");
    }

    // create the UModel connection
    // if there is a running instance of UModel (that never had a
    // connection) - use it
    // otherwise, we automatically create a new instance
    try
    {
        objUModel = WScript.GetObject("", "UModel.Application");
    }
    catch(err)
    {
        Exit("Can't access or create UModel.Application");
    }
}

function GetSourceCodeDirectory()
{
    // get directory for source code
    var path = objUModel.PersonalDataDirectory + "\\UModelExamples\\"
}
```

```

\API\CreateCode";
    var codeDirectory = objFSO.BuildPath( path, "SampleCode" );
    return codeDirectory;
}

function GetUMPFilePath()
{
    // get file path to save UModel projectfile
    return objUModel.PersonalDataDirectory + "\\UModelExamples\\API\
\CreateCode\\CreateCode.ump";
}

function IncludeCSharpProfile( objDocument )
{
    try
    {
        // get dialog for including subprojects:
        var objIncludeSubProjectDialog =
objUModel.Dialogs.IncludeSubprojectDlg;

        objIncludeSubProjectDialog.ProjectFile =
objUModel.InstallationDirectory + "\\UModelInclude\\c# Profile.ump";

        return
objDocument.IncludeSubproject( objIncludeSubProjectDialog );
    }
    catch(err)
    {
        Exit("Can't include CSharp profile");
    }
}

// ////////////////////////////////// MAIN //////////////////////////////////

CreateGlobalObjects();

objUModel.Visible = true;

// open an new, empty document
var objDocument = objUModel.NewDocument();
// get the root-package
var objRootPackage = objDocument.RootPackage;

if ( objDocument    != null &&
    objRootPackage != null &&
        IncludeCSharpProfile( objDocument ) )
{
    // create coding elements
    try
    {
        // make all modifications within one UndoStep; start
modification here

```

```
if ( !objDocument.BeginModification() )
    Exit("No modifications allowed");

// create a namespace root package
var objCSharpRootNamespace =
objRootPackage.InsertPackagedElementAt( -1, "Package" );
objCSharpRootNamespace.SetName( "CSharp" );

// find C# Profile...
var objCSharpProfile =
objRootPackage.FindPredefinedOwnedElement( 159, false );//
ePredefined_CSharp_Profile = 159,
// ...and apply it to the package, which is now a CSharp
namespace root
    objCSharpRootNamespace.InsertProfileApplicationAt( -1,
objCSharpProfile );

// create a C# namespace package...
var objCSharpNamespace =
objCSharpRootNamespace.InsertPackagedElementAt( -1, "Package" );
objCSharpNamespace.SetName( "Namespace1" );
// ... and apply the predefined C# namespace stereotype
objCSharpNamespace.ApplyPredefinedStereotype( 223 ); //
ePredefined_CSharp_namespaceStereotypeOfPackage = 223,

// create new class within the C# namespace
var objClass =
objCSharpNamespace.InsertPackagedElementAt( -1, "Class" );
var objClass2 =
objCSharpNamespace.InsertPackagedElementAt( -1, "Class" );
var objBaseClass =
objCSharpNamespace.InsertPackagedElementAt( -1, "Class" );
objClass.SetName( "MyClass" );
objClass2.SetName( "MyClass2" );
objBaseClass.SetName( "MyBaseClass" );

// set attribute-section "STAThread"
var objAttributesStereotypeApplication =
objClass.ApplyPredefinedStereotype( 191 );//
ePredefined_CSharp_attributesStereotypeOfClass = 191
var objSTAThread =
objCSharpProfile.FindPredefinedOwnedElement( 185, true ); //
ePredefined_CSharp_AttributePresetsEnumeration_STAThreadEnumerationLit
eral = 185

objAttributesStereotypeApplication.SetPredefinedTaggedValueAt(-1,
192, objSTAThread.Name); //
ePredefined_CSharp_attributesStereotypeOfClass_sectionsProperty = 192

// insert new attribute
```

```

var objProperty = objClass.InsertOwnedAttributeAt( -1 );
objProperty.SetName( "m_Att" );
objProperty.Visibility = 2;           // eVisibility_Private

= 2

objProperty.Type = objClass2;

// insert new operation
var objOperation = objClass.InsertOwnedOperationAt( -1 );
objOperation.SetName( "GetAtt" );
objOperation.Type = objClass2;

// derive MyClass from MyBaseClass
objClass.InsertGeneralizationAt( -1, objBaseClass );

// find the component view package
var objComponentView =
objRootPackage.FindPredefinedOwnedElement( 1, false );//
ePredefined_ComponentViewPackage = 1

// create a new component for C# 3.0 and set the source
code directory, where we want to generate the source code
var objComponent =
objComponentView.InsertPackagedElementAt( -1, "Component" );
objComponent.CodeLangVersion= 5; //
eCodeLang_CSharp_3_0      = 5,
objComponent.CodeProjectFileOrDirectory =
GetSourceCodeDirectory();
objComponent.IsCodeProjectFile = false;

// this component should realize our classes:
objComponent.InsertRealizationAt( -1, objClass );
objComponent.InsertRealizationAt( -1, objClass2 );
objComponent.InsertRealizationAt( -1, objBaseClass );

// do not forget to end modification and finish UndoStep
objDocument.EndModification();
}
catch( err )
{
    // rollback made changes
    objDocument.AbortModification();
    Exit("Error when creating UML model elements");
}

// update code from model
try
{
    // explicitly run a syntax check
    if ( objDocument.CheckProjectSyntax() )
    {

```

```

        // get dialog for code <=> model synchronizations and
        set the wanted options:
        var objSynchronizationSettingsDlg =
objUModel.Dialogs.SynchronizationSettingsDlg;

objSynchronizationSettingsDlg.CodeFromModel_Synchronization = 0; //
eSynchronization_Merge = 0

objSynchronizationSettingsDlg
.CodeFromModel_UserDefinedSPLTemplatesOverrideDefault = true;

        // update code from model
        if
( !objDocument
.SynchronizeCodeFromModel( objSynchronizationSettingsDlg ) )
            Exit("Update code from model failed");
        }
        else
            Exit("Syntax check failed");
    }
    catch( err )
    {
        Exit("Error when updating code from model");
    }

    // save project
    objDocument.SaveAs( GetUMPFFilePath() );

    WScript.Echo("Finished successfully");
}

// if something went wrong (and we did not save the project),
// we also do not want get asked for saving => set ModifiedFlag to
false
if ( objDocument != null )
    objDocument.ModifiedFlag = false;

objUModel.Visible = false;    // will shutdown application if it was
started by this script

```

UpdateDocumentation (JScript)

The following JScript sample reverse engineers all UModel API samples and creates HTML and RTF documentation as well as an XMI export of the UModel project.

Topics: start Umodel from script; reverse engineering dialog; documentation generation dialog

File: UModelExamples\API\UpdateDocumentation\UModelUpdateDocumentation.js:

```
// #####
```

```

// access running UModel.Application or
// launch new one and access it
// #####

// #####
// UpdateDocumentation sample
// *) When running the first time (= when no UMP file exists), reverse
engineer all C# UModelAPI samples
// and create HTML and RTF documentation, make XMI export and save
UMP file
// *) when UMP file already exists, open it and synchronize model from
code
// create HTML and RTF documentation and XMI export only if
something has been changed (listen to all different UML data events)
// #####

var bRunVisible = true;
var bShowDialogs = bRunVisible && false;

// /////////// global variables ///////////
var objUModel = null;
var objWshShell = null;
var objFSO = null;

var bChangedAnything = false;
var nAddedClasses = 0;
var nAddedInterfaces= 0;
var nAddedProperties= 0;
var nAddedOperations= 0;

// /////////// Helpers ///////////

function Exit(strErrorText)
{
    WScript.Echo(strErrorText);

    if (objUModel != null)
        objUModel.Quit();

    WScript.Quit(-1);
}

function CreateGlobalObjects ()
{
    // the Shell and FileSystemObject of the windows scripting host
    often always useful
    try
    {
        objWshShell = WScript.CreateObject("WScript.Shell");
        objFSO =
WScript.CreateObject("Scripting.FileSystemObject");

```

```

    }
    catch(err)
    {
        Exit("Can't create WScript.Shell object");
    }

    // create the UModel connection
    // if there is a running instance of UModel (that never had a
connection) - use it
    // otherwise, we automatically create a new instance
    try
    {
        objUModel = WScript.GetObject("", "UModel.Application");
    }
    catch(err)
    {
        Exit("Can't access or create UModel.Application");
    }
}

// ////////////////////////////////// get different filepathes / ensure folders
are created //////////////////////////////////
function GetScriptPath()
{
    return objUModel.PersonalDataDirectory + "\\UModelExamples\\API\\
UpdateDocumentation";
}

function GetFilePath( subdir, filename )
{
    var path = objFSO.BuildPath( GetScriptPath(), subdir );

    if ( !objFSO.FolderExists( path ) )
        objFSO.CreateFolder( path );

    return path + "\\ " + filename;
}

function GetUMPFilePath ( )      { return GetFilePath( "UMP",
"UModelAPI.ump" ); }
function GetXMIFilePath ( )      { return GetFilePath( "Output_XMI",
"UModelAPI.xmi" ); }
function GetHTMLFilePath()      { return
GetFilePath( "Output_HTML", "UModelAPI.html"); }
function GetRTFFilePath ( )      { return GetFilePath( "Output_RTF",
"UModelAPI.rtf" ); }

// ////////////////////////////////// UML data event
handlers //////////////////////////////////
function objRootPackage_OnChanged( objData, strHint )
{
    bChangedAnything = true;

```

```

    }

    // recursively count newly added classes, interfaces, properties and
    // operations
    function CountAddedElements( objNewChild )
    {
        if ( objNewChild != null )
        {
            if ( objNewChild.KindName == "Class" ) +
            +nAddedClasses;
            if ( objNewChild.KindName == "Interface" ) +
            +nAddedInterfaces;
            if ( objNewChild.KindName == "Property" ) +
            +nAddedProperties;
            if ( objNewChild.KindName == "Operation" ) +
            +nAddedOperations;

            var ownedElements = objNewChild.OwnedElements;
            var itr = new Enumerator( ownedElements );
            for ( ; !itr.atEnd(); itr.moveNext() )
                CountAddedElements( itr.item() );
        }
    }

    function objRootPackage_OnAfterAddChild( objParent, objNewChild )
    {
        bChangedAnything = true;

        // recursively count newly added classes, interfaces, properties
        // and operations
        CountAddedElements( objNewChild );
    }

    function objRootPackage_OnBeforeErase( objData )
    {
        bChangedAnything = true;
    }

    function objRootPackage_OnMoveData( objParent, objChild, bAttach )
    {
        bChangedAnything = true;
    }

    // ////////////////////////////////// MAIN //////////////////////////////////

    CreateGlobalObjects();

    if ( bRunVisible )
        objUModel.Visible = true;

    var objDocument = null;

```



```
try
{
    // open document if it exists; create new one otherwise
    var bDocumentExisted = false;

    if ( objFSO.FileExists( GetUMPFilePath() ) )
    {
        objDocument = objUModel.OpenDocument( GetUMPFilePath() );
        bDocumentExisted = true;
    }
    else
    {
        objDocument = objUModel.NewDocument();
        objDocument.SaveAs( GetUMPFilePath() );
    }

    if ( objDocument == null )
        Exit( "Cannot create or open UModel projectfile" );

    // connect to receive _IUMLDataEvents from the root-package and
    all its children:
    var objRootPackage = objDocument.RootPackage;
    WScript.ConnectObject (objRootPackage, "objRootPackage_" );

    // ensure we get *all* events from root-package and *all*
    children:
    objRootPackage.EventFilter = 2 + //
    eUMLDataEvent_EraseDataOrChild    = 2,           8 + //
    eUMLDataEvent_AddChildOrGrandChild = 8,           32 + //
    eUMLDataEvent_ChangeDataOrChild   = 32,          128; //
    eUMLDataEvent_MoveChildOrGrandChild = 128
    if ( bDocumentExisted )
    {
        // UModel projectfile already exists => update model from
code

        // get dialog for code <=> model synchronizations and set
the wanted options:
        var objSynchronizationSettingsDlg =
objUModel.Dialogs.SynchronizationSettingsDlg;
        objSynchronizationSettingsDlg.ShowDialog = bShowDialogs;

objSynchronizationSettingsDlg.ModelFromCode_Synchronization = 0; //
eSynchronization_Merge = 0

        // update model from code
        if
```

```
( !objDocument
.SynchronizeModelFromCode( objSynchronizationSettingsDlg ) )
    Exit("Update model from code failed");
}
else
{
    // UModel projectfile did not exist => newly import code
    into model

    var objImportSourceDirectoryDlg =
objUModel.Dialogs.ImportSourceDirectoryDlg;
    objImportSourceDirectoryDlg.ShowDialog = bShowDialogs;

    // set source code directory to import
    objImportSourceDirectoryDlg.Directory =
objFSO.GetParentFolderName( GetScriptPath() );
    objImportSourceDirectoryDlg.ProcessSubdirectories = true;
    // set source code language to import (C# 3.0)
    objImportSourceDirectoryDlg.Language = 5; //
eCodeLang_CSharp_3_0 = 5
    objImportSourceDirectoryDlg.Synchronization = 0; //
eSynchronization_Merge = 0
    // import in a new package
    objImportSourceDirectoryDlg.ImportInNewPackage = true;

    objImportSourceDirectoryDlg.DiagramGeneration = true;

    // content diagram generation settings
    objImportSourceDirectoryDlg.Content_GenerateSingleDiagram
        = true;

    objImportSourceDirectoryDlg.Content_GenerateDiagramPerPackage
        = true;

    objImportSourceDirectoryDlg.Content_ShowNestedClassifiersSeparately
        = false;

    objImportSourceDirectoryDlg.Content_ShowAnonymousBoundElements
        = false;

    objImportSourceDirectoryDlg.Content_HyperlinkPackagesToDiagrams
        = true;

    objImportSourceDirectoryDlg.Content_ShowAttributesCompartment
        = true;

    objImportSourceDirectoryDlg.Content_ShowOperationsCompartment
        = true;

    objImportSourceDirectoryDlg.Content_ShowNestedClassifiersCompartment
        = false;
```

```
objImportSourceDirectoryDlg.Content_ShowEnumerationLiteralsCompartment
    = true;
    objImportSourceDirectoryDlg.Content_ShowTaggedValues
        = true;
    objImportSourceDirectoryDlg.Content_Autolayout
        = 1; // eDiagramLayout_Hierarchic = 1
    // open diagrams that autolayout is done:
    objImportSourceDirectoryDlg.Content_OpenDiagrams
        = true;

    // package dependency diagram generation settings
(disabled)

objImportSourceDirectoryDlg.PackageDependency_GenerateDiagram = false;

    // import source directory
    if
( !objDocument.ImportSourceDirectory( objImportSourceDirectoryDlg ) )
    {
        // also delete newly created (empty) UMP file that
source code directory import is retried the next time
        objFSO.DeleteFile( GetUMPFilePath() );
        Exit( "Error on importing source directory" );
    }
}

    // disconnect from getting root-package events
WScript.DisconnectObject( objRootPackage );
}
catch( err )
{
    // also delete newly created (empty) UMP file that source code
directory import is retried the next time
    objFSO.DeleteFile( GetUMPFilePath() );
    Exit( "Error on importing source directory" );
}

//if something has changed, update the outputs:
if ( bChangedAnything )
{
    try
    {
        // make XMI export for UML2.1.2
        var objIExportXMIFileDlg =
objUModel.Dialogs.ExportXMIFileDlg;
        objIExportXMIFileDlg.ShowDialog = bShowDialogs;
        objIExportXMIFileDlg.XMIFile
GetXMIFilePath();
        objIExportXMIFileDlg.PrettyPrintXMIOutput = true;
        objIExportXMIFileDlg.ExportUUIDs
        objIExportXMIFileDlg.ExportExtensions
```

```

        objIExportXMIFileDlg.ExportDiagrams                = true;
        objIExportXMIFileDlg.XMIType                      = 1; //
eXMI21ForUML212 = 1

        // export to XMI file:
        if
( !objDocument.ExportToXMIFile( objIExportXMIFileDlg ) )
        {
            // error on XMI generation
        }
    }
    catch( err )
    {
        // error on XMI generation
    }

    try
    {
        var objIDocumentationGenerationDlg =
objUModel.Dialogs.GenerateDocumentationDlg;
        objIDocumentationGenerationDlg.ShowDialog = bShowDialogs;

        objIDocumentationGenerationDlg.GenerateLinksToLocalFiles =
1; // eDocumentationFilePath_RelativeToResultFile      = 1
        objIDocumentationGenerationDlg.SplitOutputToMultipleFiles
= true;

objIDocumentationGenerationDlg.ShowResultFileAfterGeneration = true;

        objIDocumentationGenerationDlg.Details_SelectAll();
        // show up to 10 base class/interface hierarchies

objIDocumentationGenerationDlg.Details_HierarchyDiagramNestingDepthUp
= 10;

        // only show directly derived classes/interfaces

objIDocumentationGenerationDlg
.Details_HierarchyDiagramNestingDepthDown = 1;
        // keep hierarchy diagram as small as possible => expand
each element only once

objIDocumentationGenerationDlg
.Details_HierarchyDiagramExpandItemsOnlyOnce = true;

objIDocumentationGenerationDlg.Include_SelectAllDiagrams();
        objIDocumentationGenerationDlg.Include_SelectNoElements();
        objIDocumentationGenerationDlg.Include_Index = true;
        objIDocumentationGenerationDlg.Include_IncludedSubprojects
= false;

        objIDocumentationGenerationDlg.Include_NamedElementsOnly =

```

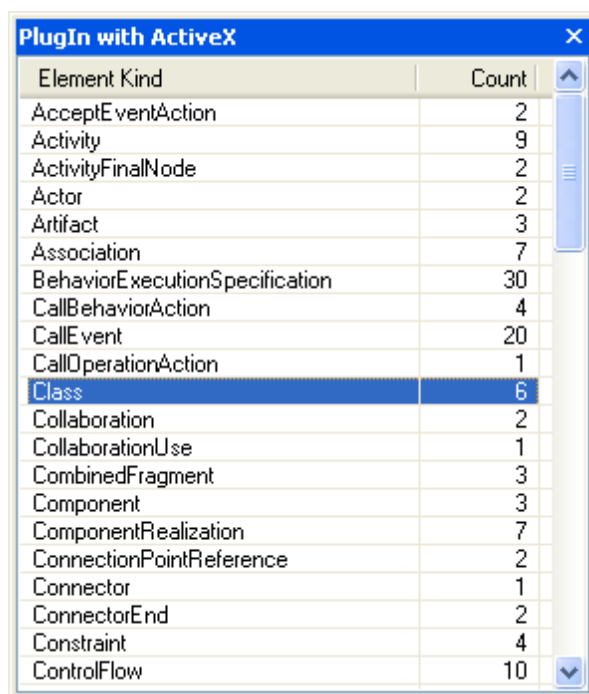
```
true;
                                objIDocumentationGenerationDlg.Include_UnknownExternals =
false;

                                var objIncludeElements =
objIDocumentationGenerationDlg.Include_Elements;
                                var itrIncludeElements = new
Enumerator( objIncludeElements );
                                for ( ; !itrIncludeElements.atEnd();
itrIncludeElements.moveNext() )
                                {
                                    var objElemSel = itrIncludeElements.item();

                                    if ( objElemSel.KindName == "Class"           ||
                                        objElemSel.KindName == "Interface"        ||
                                        objElemSel.KindName == "Enumeration"       ||
                                        objElemSel.KindName == "Operation"        ||
                                        objElemSel.KindName == "Package"          )
                                    {
                                        objElemSel.Selection = true;
                                    }
                                }

                                // generate HTML documentation (with PNG pictures)
                                objIDocumentationGenerationDlg.OutputFormat = 0; //
eDocumentationOutputFormat_HTML      = 0
                                objIDocumentationGenerationDlg.DiagramImageFormat = 0; //
eOutputImageFormat_PNG = 0
                                objIDocumentationGenerationDlg.EmbedDiagrams = false;
                                if
( !objDocument.GenerateDocumentation( objIDocumentationGenerationDlg,
GetHTMLFilePath() ) )
                                {
                                    // error on HTML documentation generation
                                }

                                // generate RTF documentation (with embeded EMF pictures)
                                objIDocumentationGenerationDlg.ShowDialog = false; //
don't show dialog again
                                objIDocumentationGenerationDlg.OutputFormat = 2; //
eDocumentationOutputFormat_RTF = 2
                                objIDocumentationGenerationDlg.DiagramImageFormat = 1; //
eOutputImageFormat_EMF = 1
                                objIDocumentationGenerationDlg.EmbedDiagrams = true;
                                if
( !objDocument.GenerateDocumentation( objIDocumentationGenerationDlg,
GetRTFFilePath() ) )
                                {
                                    // error on RTF documentation generation
                                }
                            }
```

Element Kind	Count
AcceptEventAction	2
Activity	9
ActivityFinalNode	2
Actor	2
Artifact	3
Association	7
BehaviorExecutionSpecification	30
CallBehaviorAction	4
CallEvent	20
CallOperationAction	1
Class	6
Collaboration	2
CollaborationUse	1
CombinedFragment	3
Component	3
ComponentRealization	7
ConnectionPointReference	2
Connector	1
ConnectorEnd	2
Constraint	4
ControlFlow	10

Keywords: UModel IDE plug-in, ActiveX, active document, transaction events

File: UModelExamples\IDEPlugIn\StatisticsActiveX\StatisticsActiveX.cs:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.Drawing;
using System.Runtime.InteropServices.ComTypes;
using System.Windows.Forms;
using UModelLib;
using UModelPlugInLib;

/*
 * StatisticsActiveX sample
 * listen for data modifications and count the elements of the
different element kinds
 * show the result in a listview of an ActiveX control
 */
namespace StatisticsActiveX
{
    /* StatisticsActiveX is the main class of this plugin and
implements UModelPlugInLib.IUModelPlugIn
 * it is also responsible for attaching/detaching
_IApplicationEvents and _ITransactionEvents
 */
    public partial class StatisticsActiveX : UserControl,
                                           IUModelPlugIn,
                                           _IApplicationEvents,
```

```

        _ITransactionEvents
    {
        // a sorted dictionary to count the different element kinds
        private Statistics m_Statistics;
        // reference to the transaction notifier of a UModel document
        private TransactionNotifier m_TransactionNotifier;
        // connection point to _IApplicationEvents
        private IConnectionPoint m_cpApplicationEvents = null;
        // connection cookie
        int m_nApplicationEventsCookie = 0;

        public StatisticsActiveX()
        {
            InitializeComponent();
        }

        #region IUModelPlugIn Members

        public string GetDescription()
        {
            return "PlugIn with ActiveX;This Plug-in demonstrates how
to show an ActiveX control inside UModel.";
        }

        public string GetUIModifications()
        {
            // We don't add any menu or toolbar modifications.
            return "<ConfigurationData><Modifications/></
ConfigurationData>";
        }

        public void OnInitialize(object pUModel)
        {
            // before processing DDE or batch commands
        }

        public void OnRunning(object pUModel)
        {
            // DDE or batch commands are processed; application is
fully initialized
            // and we can attach to get _IApplicationEvents

            IApplication iApp = (IApplication)pUModel;

            if (m_cpApplicationEvents == null && iApp != null)
            {
                // find connection point of _IApplicationEvents
                IConnectionPointContainer icpc =
(IConnectionPointContainer)iApp;
                Guid IID = typeof(UModelLib._IApplicationEvents).GUID;
                icpc.FindConnectionPoint(ref IID, out
m_cpApplicationEvents);
            }
        }
    }

```



```
        // advise UModelApplicationEvents as sink for
        _IApplicationEvents
            m_cpApplicationEvents.Advise(this, out
m_nApplicationEventsCookie);
    }

    AttachTransactionEvents(iApp.ActiveDocument);
}

public void OnShutdown(object pUModel)
{
    // detach application events; stop receiving events
    DetachTransactionEvents();

    if (m_cpApplicationEvents != null)
    {
        // terminate established connection to
        _IApplicationEvents
            m_cpApplicationEvents
.Unadvise(m_nApplicationEventsCookie);
        m_cpApplicationEvents = null;
    }

    // application will shutdown; release all unused objects
    GC.Collect();
}

public void OnCommand(int nID, object pUModel)
{
    // unused; we did not add any menu- or toolbar-commands
}

public UModelUpdateAction OnUpdateCommand(int nID, object
pUModel)
{
    // unused; we did not add any menu- or toolbar-commands
    return UModelUpdateAction.UModelUpdateAction_Disable;
}

#endregion

private void AttachTransactionEvents(IDocument iDoc)
{
    if (iDoc != null)
    {
        m_TransactionNotifier = iDoc.TransactionNotifier;
        if (m_TransactionNotifier != null)
        {
            // we are only interested in
            'OnEndDataModification' events so use and connect the delegate
            m_TransactionNotifier.OnEndDataModification += new
_ITransactionEvents_OnEndDataModificationEventHandler
```

```

        (OnEndDataModification);
    }
}

UpdateStatistics(iDoc);
}

// detach eventhandler from the transaction notifier
private void DetachTransactionEvents()
{
    if (m_TransactionNotifier != null)
    {
        m_TransactionNotifier.OnEndDataModification -=
OnEndDataModification;
        m_TransactionNotifier = null;
    }
    UpdateStatistics(null);
}

void UpdateStatistics(IDocument iDoc)
{
    // count current elements
    Statistics statistics = new Statistics();

    if (iDoc != null && iDoc.RootPackage != null)
        CountElements(iDoc.RootPackage, ref statistics);

    // anything changed to last update ?
    if (!statistics.IsEqual(m_Statistics))
    {
        m_Statistics = statistics;
        PopulateListView(m_Statistics);
    }

    // release unused objects
    GC.Collect();
}

private void CountElements(IUMLElement iElem, ref Statistics
statistics)
{
    // we only count editable elements
    if (iElem == null || iElem.IsEditable == false)
        return;

    string sKindName = iElem.KindName;

    if (!statistics.ContainsKey(sKindName))
        statistics[sKindName] = 1;
    else
        statistics[sKindName]++;

    foreach (IUMLElement iChild in iElem.OwnedElements)

```

```
        CountElements(iChild, ref statistics);
    }

    private void PopulateListView(Statistics statistics)
    {
        listView1.BeginUpdate();

        listView1.Items.Clear();
        foreach (KeyValuePair<string, int> kvp in statistics)
        {
            ListViewItem item = new ListViewItem(kvp.Key);
            item.SubItems.Add(Convert.ToString(kvp.Value));

            listView1.Items.Add(item);
        }

        listView1.EndUpdate();
    }

    #region _ITransactionEvents Members

    public void OnBeginDataModification(Document ipDocument)
    {
        // begin of transaction
    }

    public void OnEndDataModification(Document ipDocument)
    {
        // end of transaction - update statistics
        if (ipDocument != null && ipDocument.TransactionNotifier
== m_TransactionNotifier)
            UpdateStatistics(ipDocument);
    }

    #endregion

    #region _IApplicationEvents Members

    public void OnNewDocument(Document ipDocument)
    {
        // a new document has been created in UModel => (re-)
connect transaction events
        AttachTransactionEvents(ipDocument);
    }

    public void OnDocumentOpened(Document ipDocument)
    {
        // a document has been opened in UModel => (re-)connect
transaction events
        AttachTransactionEvents(ipDocument);
    }

    public void OnDocumentClosed(Document ipDocument)
```

```
{
    // document has been closed in UModel => disconnect
    transaction events
    if (ipDocument != null && ipDocument.TransactionNotifier
    == m_TransactionNotifier)
        DetachTransactionEvents();
}

public void OnShutdown()
{
}

#endregion

#region Statistics dictionary

private class Statistics : SortedDictionary<string, int>
{
    public bool IsEqual(Statistics other)
    {
        if (other == null)
            return false;

        if (Count != other.Count)
            return false;

        Enumerator e1 = GetEnumerator();
        Enumerator e2 = other.GetEnumerator();
        while (e1.MoveNext() && e2.MoveNext())
        {
            if ((e1.Current.Key != e2.Current.Key) ||
                (e1.Current.Value != e2.Current.Value))
                return false;
        }

        return true;
    }
}

#endregion
}
```

20.5 UModel API Reference

This section describes all interfaces, operations, enumerations and events of [UModel Plugins](#) and of the [UModel API](#).

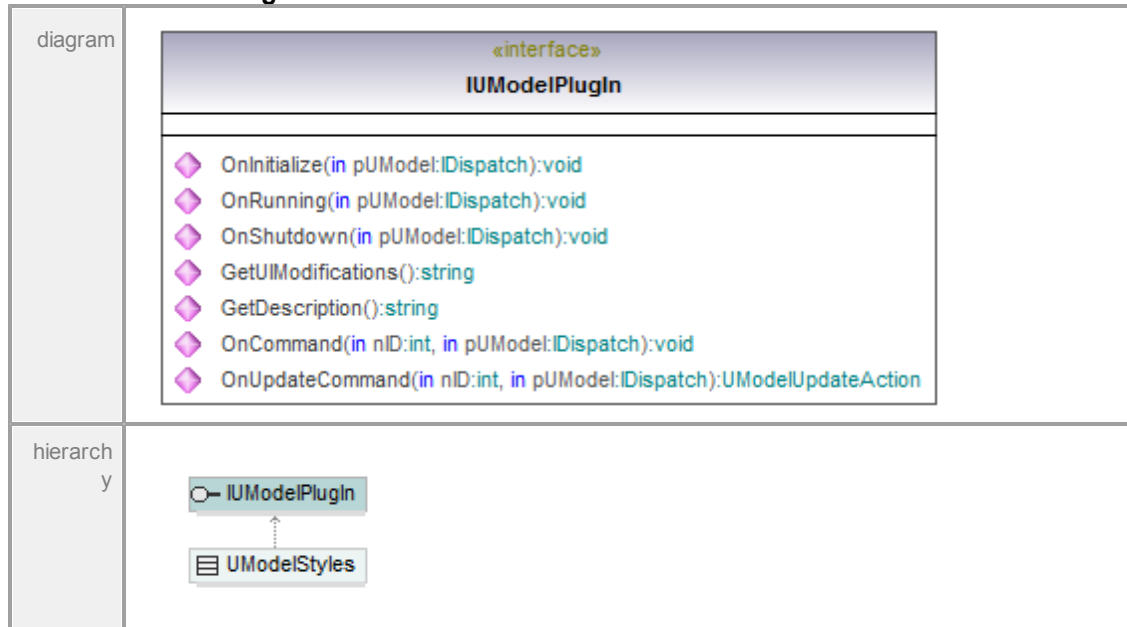
20.5.1 UModel Plugins

UModel allows you to create your own plug-ins and integrate them into UModel (also see [how UModel interacts with IUModelPlugin](#)).

A detailed description about how to create a plug-in can be found in section [How to create a plug-in in MS Visual Studio .NET](#).

UModelAPI - IUModelPlugin

Interface **IUModelPlugin**



Operation **IUModelPlugin::GetDescription**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUModelPlugin::GetUIModifications**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUModelPlugin::OnCommand**

parameter	name	direction	type	type modifier	multiplicity	default
	nID	in	int			
	pUModel	in	IDispatch			
	return	return	void			

Operation **IUModelPlugin::OnInitialize**

parameter	name	direction	type	type modifier	multiplicity	default
	pUModel	in	IDispatch			
	return	return	void			

Operation **IUModelPlugin::OnRunning**

parameter	name	direction	type	type modifier	multiplicity	default
r	pUModel	in	IDispatch			
	return	return	void			

Operation **IUModelPlugIn::OnShutdown**

parameter	name	direction	type	type modifier	multiplicity	default
r	pUModel	in	IDispatch			
	return	return	void			

Operation **IUModelPlugIn::OnUpdateCommand**

parameter	name	direction	type	type modifier	multiplicity	default
r	nID	in	int			
	pUModel	in	IDispatch			
	return	return	UModelUpdateAction			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - UModelUpdateAction

Enumeration **UModelUpdateAction**

diagram	«enumeration» UModelUpdateAction UModelUpdateAction_Enable = 1 UModelUpdateAction_Disable = 2 UModelUpdateAction_Check = 4 UModelUpdateAction_Uncheck = 8				
typedElements	Interface IUModelPlugIn Class UModelStyles		Operation OnUpdateCommand Operation OnUpdateCommand OnUpdateSetStyles		

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

20.5.2 UModel API

This chapter contains the reference of the UModel Type Library.

Errors

- 1000 The application object is no longer valid.
- 1001 Invalid parameter or invalid address for the return parameter was specified.
- 1002 UModel API is not available in the current edition.
- 1003 Model Transformations are not supported in the current edition.

- 1050 Macro not found
- 1051 Invalid (nested) macro execution

- 1100 Error when saving file, probably the file name is invalid.
- 1101 Invalid (duplicate) call to BeginModification.
- 1102 EndModification called without BeginModification

- 1200 Error deleting file at URL.
- 1201 Error creating directory at URL.

Errors of UMLData can be found [here](#).

UModelAPI - IApplication

Interface **IApplication**

diagram	«interface» IApplication ◆ Quit():void ◆ OpenDocument(in strModelFilePath:string):IDocument ◆ OpenDocumentFromURL(in IURLDlg:IURLDlg):IDocument ◆ ImportFromXMIFile(in strXMIFile:string):IDocument ◆ ImportFromXMIFileFromURL(in IURLDlg:IURLDlg):IDocument ◆ NewDocument():IDocument ◆ RunMacro(in strName:string):void ◆ «GetAccessor, property» Name():string ◆ «GetAccessor, property» Application():IDispatch ◆ «GetAccessor, property» Parent():IDispatch ◆ «GetAccessor, SetAccessor, property» Visible():bool ◆ «GetAccessor, property» WindowHandle():int ◆ «GetAccessor, property» ActiveDocument():IDocument ◆ «GetAccessor, property» Status():ENUMApplicationStatus ◆ «GetAccessor, property» MajorVersion():int ◆ «GetAccessor, property» MinorVersion():int ◆ «GetAccessor, property» Edition():string ◆ «GetAccessor, property» IsAPISupported():bool ◆ «GetAccessor, property» APIMajorVersion():int ◆ «GetAccessor, property» APIMinorVersion():int ◆ «GetAccessor, property» InstallationDirectory():string ◆ «GetAccessor, property» PersonalDataDirectory():string ◆ «GetAccessor, property» Dialogs():IDialogs ◆ «GetAccessor, property» Options():ILocalOptions ◆ «GetAccessor, property» ServicePackVersion():int	
typedElements	Class UModelStyles	Operation OnSetStyles OnUpdateSetStyles

Operation **IApplication::ActiveDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDocument			

Operation **IApplication::APIMajorVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			
documentation	A change in the APIMajorVersion of the type library (e.g. 1.0 => 2.0) means that non-scripting clients (e.g. IDE Plugins written in C#, VB.NET, C++,...) should be recompiled.					

Operation **IApplication::APIMinorVersion**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IApplication::Application**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **IApplication::Dialogs**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDialogs			

Operation **IApplication::Edition**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IApplication::ImportFromXMIFile**

parameter	name	direction	type	type modifier	multiplicity	default
r	strXMIFile return	in return	string IDocument			

Operation **IApplication::ImportFromXMIFileFromURL**

parameter	name	direction	type	type modifier	multiplicity	default
r	IURLDlg return	in return	IURLDlg IDocument			

Operation **IApplication::InstallationDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IApplication::IsAPISupported**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IApplication::MajorVersion**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IApplication::MinorVersion**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IApplication::Name**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IApplication::NewDocument**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDocument			

Operation **IApplication::OpenDocument**

parameter	name	direction	type	type modifier	multiplicity	default
r	strModelFilePath		string			
	return	return	IDocument			

Operation **IApplication::OpenDocumentFromURL**

parameter	name	direction	type	type modifier	multiplicity	default
r	IURLDlg	in	IURLDlg			
	return	return	IDocument			

Operation **IApplication::Options**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ILocalOptions			

Operation **IApplication::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **IApplication::PersonalDataDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IApplication::Quit**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IApplication::RunMacro**

parameter	name	direction	type	type modifier	multiplicity	default
r	strName	in	string			
	return	return	void			

Operation **IApplication::ServicePackVersion**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IApplication::Status**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMApplicationStatus			

Operation **IApplication::Visible**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	bool

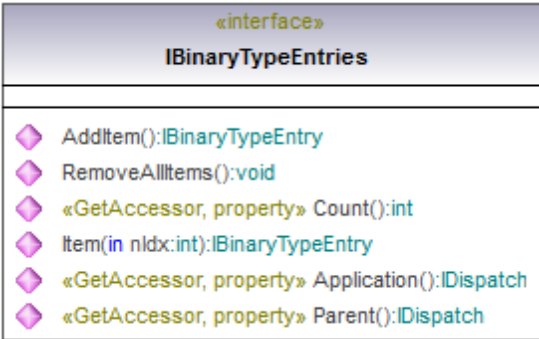
Operation **Application::WindowHandle**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IBinaryTypeEntriesInterface **IBinaryTypeEntries**

diagram						
typedElements	Interface ImportBinaryTypesDlg Operation CSharp_BinaryTypes Java_BinaryTypes VBasic_BinaryTypes					

Operation **IBinaryTypeEntries::AddItem**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IBinaryTypeEntry			

Operation **IBinaryTypeEntries::Application**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **IBinaryTypeEntries::Count**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IBinaryTypeEntries::Item**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx return	in return	int IBinaryTypeEntry			

Operation **IBinaryTypeEntries::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

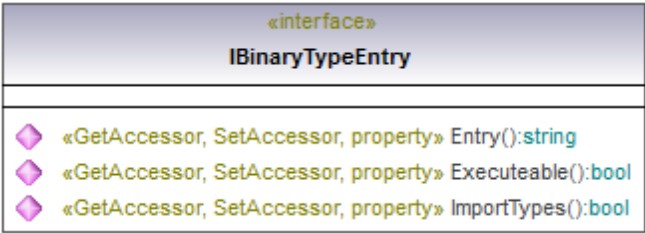
Operation IBinaryTypeEntries::RemoveAllItems

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IBinaryTypeEntry**Interface IBinaryTypeEntry**

diagram						
typedElements	Interface IBinaryTypeEntries Operation AddItem Item					

Operation IBinaryTypeEntry::Entry

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation IBinaryTypeEntry::Executable

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation IBinaryTypeEntry::ImportTypes

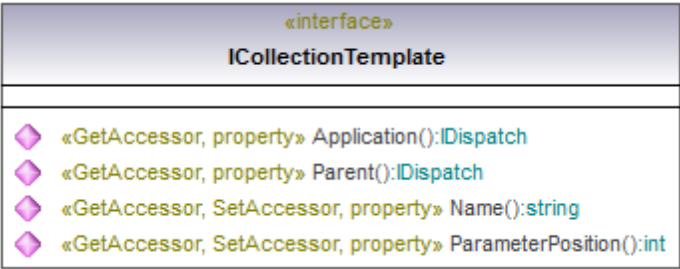
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ICollectionTemplate

Interface ICollectionTemplate

diagram		
typedElements	Interface ICollectionTemplates	Operation InsertItemAt Item

Operation ICollectionTemplate::Application

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation ICollectionTemplate::Name

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

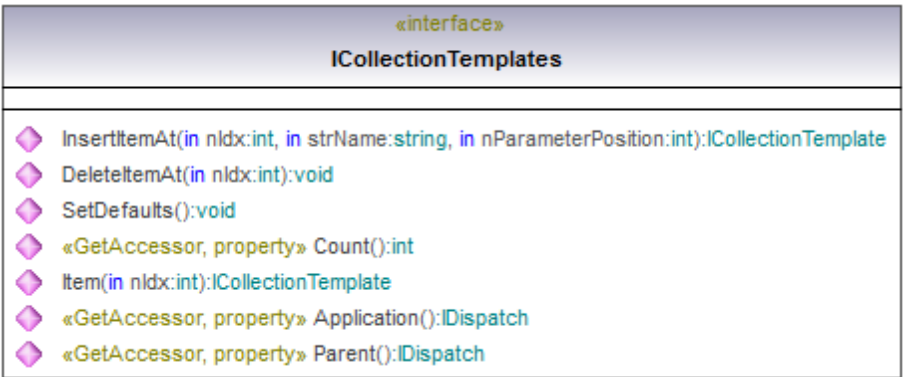
Operation ICollectionTemplate::ParameterPosition

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation ICollectionTemplate::Parent

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

UModelAPI - ICollectionTemplates**Interface ICollectionTemplates**

diagram	 <pre> «interface» ICollectionTemplates +InsertItemAt(in nIdx:int, in strName:string, in nParameterPosition:int):ICollectionTemplate +DeleteItemAt(in nIdx:int):void +SetDefaults():void +«GetAccessor, property» Count():int +Item(in nIdx:int):ICollectionTemplate +«GetAccessor, property» Application():IDispatch +«GetAccessor, property» Parent():IDispatch </pre>		
typedElements	Interface ILocalOptionsDiagramEditing	Operation	CollectionTemplates_CSharp CollectionTemplates_Java CollectionTemplates_UML CollectionTemplates_VBasic

Operation ICollectionTemplates::Application

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation ICollectionTemplates::Count

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation ICollectionTemplates::DeleteItemAt

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	void			

Operation ICollectionTemplates::InsertItemAt

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	strName	in	string			
	nParameterPosition	in	int			
	return	return	ICollectionTemplate			

Operation ICollectionTemplates::Item

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	ICollectionTemplate			

Operation ICollectionTemplates::Parent

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation **ICollectionTemplates::SetDefaults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	void			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IDiagramWindowInterface **IDiagramWindow**

diagram	«interface» IDiagramWindow UpdateWindow():void SetActiveDiagramWindow():void Close():void SelectGuiElement(in ipItemToSelect:IUMLGuiLink, in bClearSelectionBefore:bool):void ScrollToGuiElement(in ipGuiLink:IUMLGuiLink):void CopyAsBitmap():void CopySelectionAsBitmap():void SaveDiagramAsImage(in strFile:string, in ImageFormat:ENUMOutputImageFormat):void Autolayout(in nVal:ENUMDiagramLayoutKind):void AutolayoutSelection(in nVal:ENUMDiagramLayoutKind):void «GetAccessor, property» Name():string «GetAccessor, property» Application():IDispatch «GetAccessor, SetAccessor, property» ZoomFactor():int «GetAccessor, property» Diagram():IUMLGuiDiagram «GetAccessor, property» ScrollPosX():int «GetAccessor, property» ScrollPosY():int «GetAccessor, property» SelectedGuiElements():IUMLDataList «GetAccessor, property» FocusedGuiElement():IUMLGuiLink «GetAccessor, property» FocusedData():IUMLData «GetAccessor, property» Parent():IDispatch	
typedElements	Interface IDiagramWindowEvents Interface IDocumentEvents Interface IDiagramWindows Interface IDocument	Operation OnDiagramWindowClosed Operation OnActivateDiagramWindow Operation OnDiagramWindowClosed Operation OnDiagramWindowOpened Operation Item Operation ActiveDiagramWindow Operation OpenDiagram

Operation **IDiagramWindow::Application**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	IDispatch			

Operation **IDiagramWindow::Autolayout**

parameter	name	direction	type	type modifier	multiplicity	default
r	nVal	in	ENUMDiagramLayoutKind			
	return	return	void			

Operation **IDiagramWindow::AutolayoutSelection**

parameter	name	direction	type	type modifier	multiplicity	default
r	nVal	in	ENUMDiagramLayoutKind			
	return	return	void			

Operation **IDiagramWindow::Close**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IDiagramWindow::CopyAsBitmap**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IDiagramWindow::CopySelectionAsBitmap**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IDiagramWindow::Diagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMGuiDiagram			

Operation **IDiagramWindow::FocusedData**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMData			

Operation **IDiagramWindow::FocusedGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMGuiLink			

Operation **IDiagramWindow::Name**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IDiagramWindow::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **IDiagramWindow::SaveDiagramAsImage**

parameter	name	direction	type	type modifier	multiplicity	default
r	strFile	in	string			
	ImageFormat	in	ENUM OutputImageFormat			
	return	return	void			

Operation **IDiagramWindow::ScrollPosX**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IDiagramWindow::ScrollPosY**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IDiagramWindow::ScrollToGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipGuiLink	in	IUMLGuiLink			
	return	return	void			

Operation **IDiagramWindow::SelectedGuiElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLGuiElement .					

Operation **IDiagramWindow::SelectGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipItemToSelect	in	IUMLGuiLink			
	bClearSelectionBefore		bool			
	return	return	void			

Operation **IDiagramWindow::SetActiveDiagramWindow**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IDiagramWindow::UpdateWindow**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IDiagramWindow::ZoomFactor**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

UModelAPI - IDiagramWindows

Interface IDiagramWindows

diagram	«interface» IDiagramWindows «GetAccessor, property» Count():int «GetAccessor, property» Item(in nidx:int):IDiagramWindow «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch	
typedElements	Interface IDocument	Operation DiagramWindows

Operation IDiagramWindows::Application

parameter	name return	direction return	type IDispatch	type modifier	multiplicity	default

Operation IDiagramWindows::Count

parameter	name return	direction return	type int	type modifier	multiplicity	default

Operation IDiagramWindows::Item

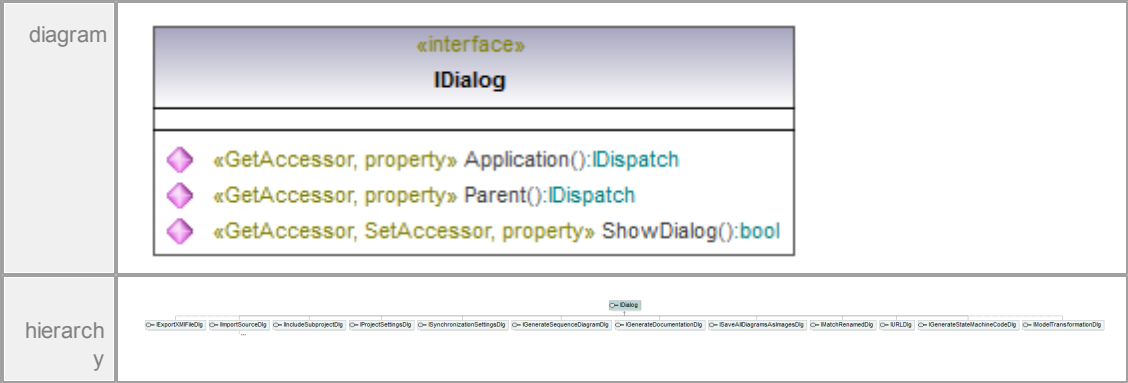
parameter	name nidx return	direction in return	type int IDiagramWindow	type modifier	multiplicity	default

Operation IDiagramWindows::Parent

parameter	name return	direction return	type IDispatch	type modifier	multiplicity	default

UModelAPI - IDialog

Interface IDialog



Operation IDialog::Application

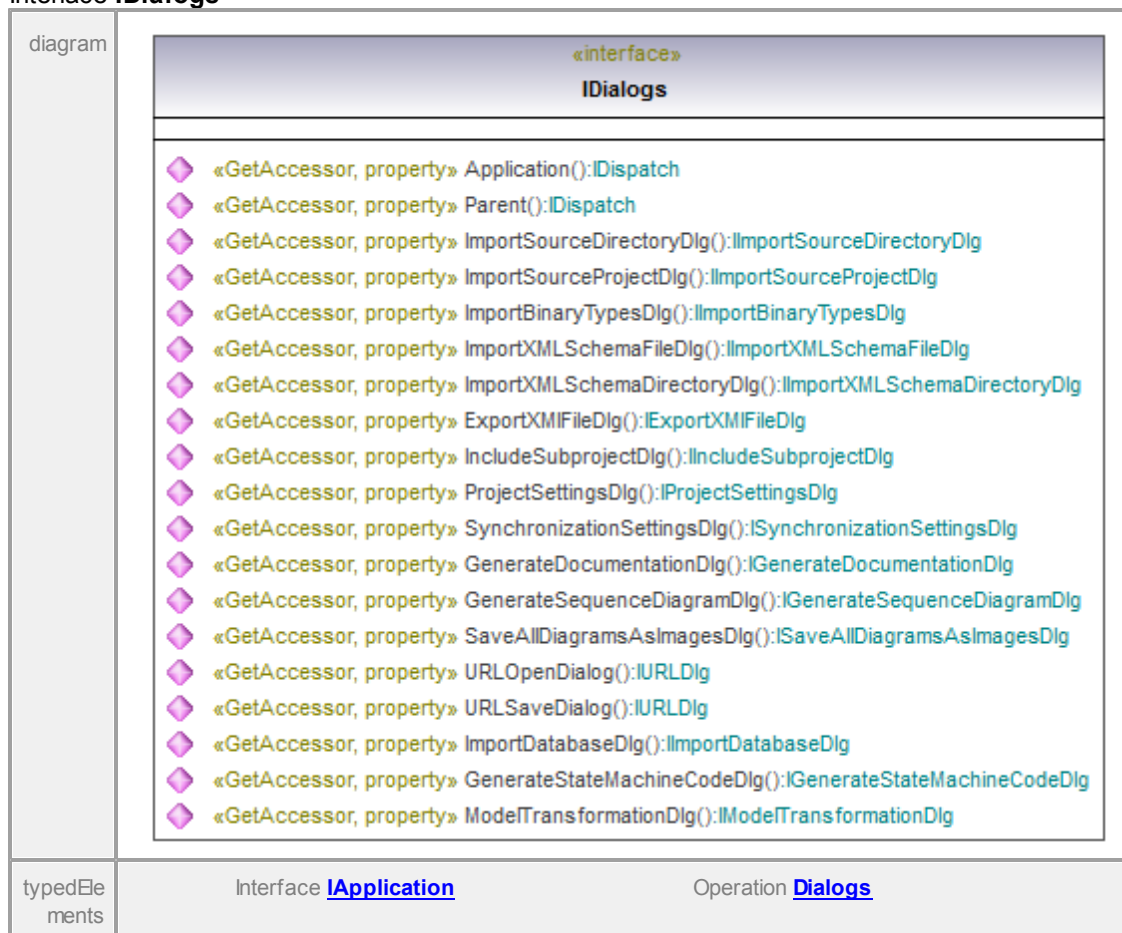
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation IDialog::Parent

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation IDialog::ShowDialog

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

UModelAPI - IDialogs**Interface IDialogs****Operation IDialogs::Application**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation IDialogs::ExportXMIFileDlg

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IExportXMIFileDlg			

Operation IDialogs::GenerateDocumentationDlg

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IGenerateDocumentationDlg			

Operation IDialogs::GenerateSequenceDiagramDlg

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IGenerateSequenceDiagramDlg			

	g

Operation **IDialogs::GenerateStateMachineCodeDlg**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IGenerateState MachineCodeD lg			

Operation **IDialogs::ImportBinaryTypesDlg**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IImportBinaryT ypesDlg			

Operation **IDialogs::ImportDatabaseDlg**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IImportDatabas eDlg			

Operation **IDialogs::ImportSourceDirectoryDlg**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IImportSource DirectoryDlg			

Operation **IDialogs::ImportSourceProjectDlg**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IImportSource ProjectDlg			

Operation **IDialogs::ImportXMLSchemaDirectoryDlg**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IImportXMLSc hemaDirectory Dlg			

Operation **IDialogs::ImportXMLSchemaFileDialog**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IImportXMLSc hemaFileDialog			

Operation **IDialogs::IncludeSubprojectDlg**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IIncludeSubpro jectDlg			

Operation **IDialogs::ModelTransformationDlg**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IModelTransfo rmationDlg			

Operation **IDialogs::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **IDialogs::ProjectSettingsDlg**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	<u>IProjectSettingsDlg</u>			

Operation **IDialogs::SaveAllDiagramsAsImagesDlg**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	<u>ISaveAllDiagramsAsImagesDlg</u>			

Operation **IDialogs::SynchronizationSettingsDlg**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	<u>ISynchronizationSettingsDlg</u>			

Operation **IDialogs::URLOpenDialog**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	<u>IURLDlg</u>			

Operation **IDialogs::URLSaveDialog**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	<u>IURLDlg</u>			

UModelAPI - IDocumentInterface **IDocument**

diagram	«interface» IDocument ◆ Save():void ◆ SaveAs(in strFileName:string):void ◆ OpenAllDiagrams():void ◆ BeginModification():bool ◆ AbortModification():void ◆ EndModification():void ◆ OpenDiagram(in ipUMLDiagram:IUMLGuiDiagram):IDialogWindow ◆ ElementFamilyStyles(in strKind:string):IUMLGuiStyles ◆ CheckProjectSyntax():bool ◆ ImportSourceDirectory(in ipDlg:IImportSourceDirectoryDlg):bool ◆ ImportSourceProject(in ipDlg:IImportSourceProjectDlg):bool ◆ ImportBinaryTypes(in ipDlg:IImportBinaryTypesDlg):bool ◆ ImportXMLSchemaFile(in ipDlg:IImportXMLSchemaFileDlg):bool ◆ ImportXMLSchemaDirectory(in ipDlg:IImportXMLSchemaDirectoryDlg):bool ◆ ExportToXMLFile(in ipDlg:IExportXMLFileDlg):bool ◆ SynchronizeModelFromCode(in ipDlg:ISynchronizationSettingsDlg):bool ◆ SynchronizeCodeFromModel(in ipDlg:ISynchronizationSettingsDlg):bool ◆ IncludeSubproject(in ipDlg:IIncludeSubprojectDlg):bool ◆ MergeProject(in strProjectFile:string):bool ◆ MergeProjectFromURL(in IUMLDlg:IURLDlg):bool ◆ SynchronizationSettings(in ipDlg:ISynchronizationSettingsDlg):void ◆ ProjectSettings(in ipDlg:IProjectSettingsDlg):void ◆ GenerateDocumentation(in ipDlg:IGenerateDocumentationDlg, in strResultFile:string):bool ◆ SaveAllDiagramsAsImages(in ipDlg:ISaveAllDiagramsAsImagesDlg):bool ◆ GenerateSequenceDiagram(in ipDlg:IGenerateSequenceDiagramDlg, in ipOp:IUMLOperation):bool ◆ SaveToURL(in IUMLDlg:IURLDlg):void ◆ SaveCopyAs(in strFileName:string):void ◆ CanFocusUMLDataInModelTree(in ipUMLData:IUMLData):bool ◆ FocusUMLDataInModelTree(in ipUMLData:IUMLData, in bFocusModelTree:bool, in bEnsureModelTreeVisible:bool):void ◆ Reload():bool ◆ ImportDatabase(in ipDlg:IImportDatabaseDlg):bool ◆ MergeProject3Way(in strProjectFile:string, in strCommonAncestorFile:string):bool ◆ GenerateStateMachineCode(in ipDlg:IGenerateStateMachineCodeDlg, in ipStateMachine:IUMLStateMachine):bool ◆ ModelTransformation(in ipDlg:IModelTransformationDlg):bool ◆ GenerateSequenceDiagramsForAllOperations(in bAllPublicOnly:bool, in bIncludeGettersAndSetters:bool):void ◆ «GetAccessor, property» Application():IDispatch ◆ «GetAccessor, property» Parent():IDispatch ◆ «GetAccessor, SetAccessor, property» FullName():string ◆ «GetAccessor, property» Name():string ◆ «GetAccessor, property» Path():string ◆ «GetAccessor, property» IsLoadedFromPreviousFileFormat():bool ◆ «GetAccessor, property» Saved():bool ◆ «GetAccessor, property» DiagramWindows():IDialogWindows ◆ «GetAccessor, property» TransactionNotifier():ITransactionNotifier ◆ «GetAccessor, SetAccessor, property» ModifiedFlag():bool ◆ «GetAccessor, property» RootPackage():IUMLPackage ◆ «GetAccessor, property» GuiRoot():IUMLGuiRootElement ◆ «GetAccessor, property» IsInUndoRedo():bool ◆ «GetAccessor, property» FocusedUMLData():IUMLData ◆ «GetAccessor, property» ActiveDiagramWindow():IDialogWindow ◆ «GetAccessor, property» ProjectStyles():IUMLGuiStyles ◆ «GetAccessor, property» NodeStyles():IUMLGuiStyles ◆ «GetAccessor, property» LineStyles():IUMLGuiStyles ◆ «GetAccessor, property» FocusedUMLDataNotifier():IFocusedUMLDataNotifier

typedElements	Interface IApplicationEvents	Operation OnDocumentClosed
		Operation OnDocumentOpened
		Operation OnNew Document
	Interface IDocumentEvents	Operation OnAfterReloadDocument
		Operation OnBeforeReloadDocument
		Operation OnDocumentClosed
		Operation OnDocumentSaved
		Operation OnDocumentSavedAs
		Operation OnModifiedFlagChanged
	Interface ITransactionEvents	Operation OnBeginDataModification
		Operation OnEndDataModification
	Interface IApplication	Operation Active Document
		Operation ImportFromXMIFile
		Operation ImportFromXMIFileFromURL
		Operation New Document
		Operation OpenDocument
		Operation OpenDocumentFromURL

Operation **IDocument::AbortModification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	void			

Operation **IDocument::ActiveDiagramWindow**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDiagramWindow			

Operation **IDocument::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation **IDocument::BeginModification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IDocument::CanFocusUMLDataInModelTree**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	in	IUMLData			
	return	return	bool			

Operation **IDocument::CheckProjectSyntax**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IDocument::DiagramWindows**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDiagramWindows			

Operation **IDocument::ElementFamilyStyles**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	return	return	IUMLGuiStyles			

Operation **IDocument::EndModification**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IDocument::ExportToXMIFile**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipDlg	in	IExportXMIFileDlg			
	return	return	bool			

Operation **IDocument::FocusedUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLData			
documentation	Get the focused UML data of the document. Normally this is the one which is shown in the "Properties" window.					

Operation **IDocument::FocusedUMLDataNotifier**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IFocusedUMLDataNotifier			

Operation **IDocument::FocusUMLDataInModelTree**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipUMLData	in	IUMLData			
	bFocusModelTree		bool			
	bEnsureModelTreeVisible		bool			
	return	return	void			

Operation **IDocument::FullName**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IDocument::GenerateDocumentation**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipDlg	in	IGenerateDocumentationDlg			
	strResultFile	in	string			
	return	return	bool			

Operation **IDocument::GenerateSequenceDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipDlg	in	IGenerateSequenceDiagramDlg			

			enceDiagramDlg
	ipOp	in	IUMLOperation
	return	return	bool

Operation **IDocument::GenerateSequenceDiagramsForAllOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	bAllPublicOnly	in	bool			
	bIncludeGettersAndSetters	in	bool			
	return	return	void			

Operation **IDocument::GenerateStateMachineCode**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	IGenerateStateMachineCodeDlg			
	ipStateMachine	in	IUMLStateMachine			
	return	return	bool			

Operation **IDocument::GuiRoot**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLGuiRootElement			

Operation **IDocument::ImportBinaryTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	IImportBinaryTypesDlg			
	return	return	bool			

Operation **IDocument::ImportDatabase**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	IImportDatabaseDlg			
	return	return	bool			

Operation **IDocument::ImportSourceDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	IImportSourceDirectoryDlg			
	return	return	bool			

Operation **IDocument::ImportSourceProject**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	IImportSourceProjectDlg			
	return	return	bool			

Operation **IDocument::ImportXMLSchemaDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	IImportXMLSchemaDirectoryDlg			
	return	return	bool			

Operation **IDocument::ImportXMLSchemaFile**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	IImportXMLSchemaFileDialog			
	return	return	bool			

Operation **IDocument::IncludeSubproject**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	IIncludeSubprojectDlg			
	return	return	bool			

Operation **IDocument::IsInUndoRedo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IDocument::IsLoadedFromPreviousFileFormat**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			
documentation	True, when the document has been loaded from a project file with a previous file format version. When saving the document, previous versions of UModel will not be able to load this file anymore.					

Operation **IDocument::LineStyle**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLGuiStyles			

Operation **IDocument::MergeProject**

parameter	name	direction	type	type modifier	multiplicity	default
	strProjectFile	in	string			
	return	return	bool			

Operation **IDocument::MergeProject3Way**

parameter	name	direction	type	type modifier	multiplicity	default
	strProjectFile	in	string			
	strCommonAncestorFile	in	string			
	return	return	bool			

Operation **IDocument::MergeProjectFromURL**

parameter	name	direction	type	type modifier	multiplicity	default

r	IURLDlg return	in return	IURLDlg bool			

Operation **IDocument::ModelTransformation**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipDlg	in	IModelTransformationDlg			
	return	return	bool			

Operation **IDocument::ModifiedFlag**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IDocument::Name**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IDocument::NodeStyles**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiStyles			

Operation **IDocument::OpenAllDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IDocument::OpenDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipUMLDiagram	in	IUMLGuiDiagram			
	return	return	IDiagramWindow			

Operation **IDocument::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **IDocument::Path**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IDocument::ProjectSettings**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipDlg	in	IProjectSettingsDlg			
	return	return	void			

Operation **IDocument::ProjectStyles**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiStyles			

Operation **IDocument::Reload**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IDocument::RootPackage**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLPackage			

Operation **IDocument::Save**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IDocument::SaveAllDiagramsAsImages**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipDlg	in	ISaveAllDiagramsAsImagesDlg			
	return	return	bool			

Operation **IDocument::SaveAs**

parameter	name	direction	type	type modifier	multiplicity	default
r	strFileName	in	string			
	return	return	void			

Operation **IDocument::SaveCopyAs**

parameter	name	direction	type	type modifier	multiplicity	default
r	strFileName	in	string			
	return	return	void			

Operation **IDocument::Saved**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IDocument::SaveToURL**

parameter	name	direction	type	type modifier	multiplicity	default
r	IURLDlg	in	IURLDlg			
	return	return	void			

Operation **IDocument::SynchronizationSettings**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipDlg	in	ISynchronizationSettingsDlg			
	return	return	void			

Operation **IDocument::SynchronizeCodeFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	ISynchronizationSettingsDlg			
	return	return	bool			

Operation **IDocument::SynchronizeModelFromCode**

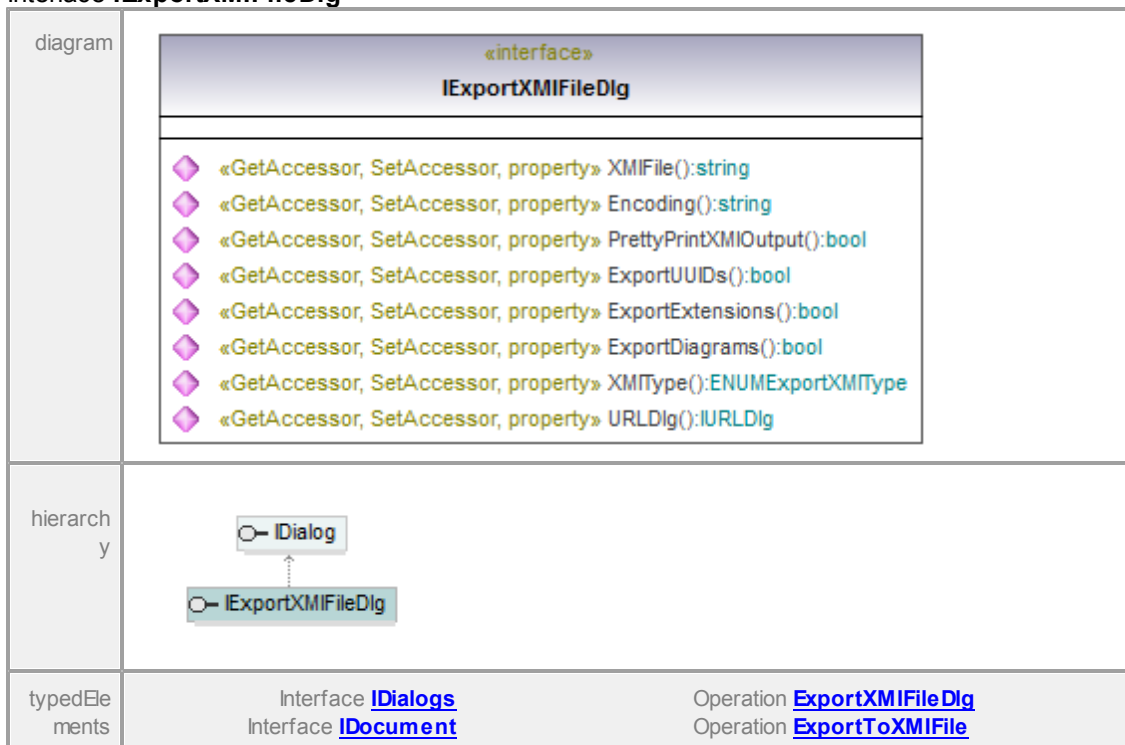
parameter	name	direction	type	type modifier	multiplicity	default
	ipDlg	in	ISynchronizationSettingsDlg			
	return	return	bool			

Operation **IDocument::TransactionNotifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ITransactionNotifier			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IExportXMIFileDlgInterface **IExportXMIFileDlg**Operation **IExportXMIFileDlg::Encoding**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IExportXMIFileDlg::ExportDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IExportXMIFileDlg::ExportExtensions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IExportXMIFileDlg::ExportUUIDs**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IExportXMIFileDlg::PrettyPrintXMIOutput**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IExportXMIFileDlg::URLDlg**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	URLDlg			

Operation **IExportXMIFileDlg::XMIFile**

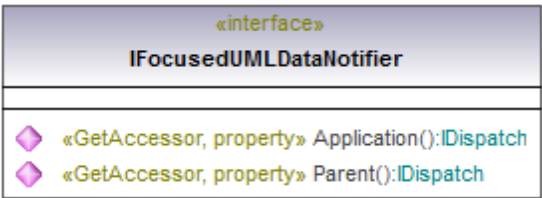
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IExportXMIFileDlg::XMISType**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMExportXMISType			

UModelAPI - IFocusedUMLDataNotifier

Interface **IFocusedUMLDataNotifier**

diagram					
typedElements	Interface IDocument		Operation FocusedUMLDataNotifier		

Operation **IFocusedUMLDataNotifier::Application**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

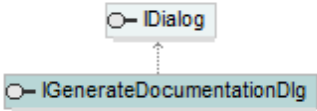
Operation **IFocusedUMLDataNotifier::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IGenerateDocumentationDlgInterface **IGenerateDocumentationDlg**

diagram	<i>[The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (http://manual.altova.com/UModel/umodelenterprise/).]</i>					
hierarchy						
typedElements	Interface IDialogs Interface IDocument		Operation GenerateDocumentationDlg Operation GenerateDocumentation			

Operation **IGenerateDocumentationDlg::CreateFolderForDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_Associations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_BoundElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_Constraints**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_Diagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_Documentation**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_DocumentationAsHTML**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IGenerateDocumentationDlg::Details_Generals**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IGenerateDocumentationDlg::Details_HierarchyDiagram**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IGenerateDocumentationDlg::Details_HierarchyDiagramExpandItemsOnlyOnce**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IGenerateDocumentationDlg::Details_HierarchyDiagramNestingDepthDown**

paramete r	name return	direction return	type int	type modifier	multiplicity	default

Operation **IGenerateDocumentationDlg::Details_HierarchyDiagramNestingDepthUp**

paramete r	name return	direction return	type int	type modifier	multiplicity	default

Operation **IGenerateDocumentationDlg::Details_Hyperlinks**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IGenerateDocumentationDlg::Details_ImplementedInterfaces**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IGenerateDocumentationDlg::Details_OperationParameters**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IGenerateDocumentationDlg::Details_OwnedDiagrams**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IGenerateDocumentationDlg::Details_OwnedMembers**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IGenerateDocumentationDlg::Details_Owner**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_Properties**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_SelectAll**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IGenerateDocumentationDlg::Details_SelectNone**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IGenerateDocumentationDlg::Details_ShownOnDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_SourceTargetOfRelations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_Specifics**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_TemplateParameters**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_TemplateParameterSubstitutions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Details_TypedElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::DiagramImageFormat**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUM OutputImageFormat			

Operation **IGenerateDocumentationDlg::EmbedCSSinHTML**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::EmbedDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Fonts_GetFace**

parameter	name	direction	type	type modifier	multiplicity	default
r	nSetting	in	ENUMDocumentationFontSetting			
	return	return	string			

Operation **IGenerateDocumentationDlg::Fonts_GetSize**

parameter	name	direction	type	type modifier	multiplicity	default
r	nSetting	in	ENUMDocumentationFontSetting			
	return	return	int			

Operation **IGenerateDocumentationDlg::Fonts_GetTextColor**

parameter	name	direction	type	type modifier	multiplicity	default
r	nSetting	in	ENUMDocumentationFontSetting			
	return	return	int			

Operation **IGenerateDocumentationDlg::Fonts_IsBold**

parameter	name	direction	type	type modifier	multiplicity	default
r	nSetting	in	ENUMDocumentationFontSetting			
	return	return	bool			

Operation **IGenerateDocumentationDlg::Fonts_IsItalic**

parameter	name	direction	type	type modifier	multiplicity	default
r	nSetting	in	ENUMDocumentationFontSetting			
	return	return	bool			

Operation **IGenerateDocumentationDlg::Fonts_IsUnderline**

parameter	name	direction	type	type modifier	multiplicity	default
r	nSetting	in	ENUMDocumentationFontSetting			
	return	return	bool			

Operation **IGenerateDocumentationDlg::Fonts_SetBold**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	in	ENUMDocumentationFontSetting			
	bVal	in	bool			
	return	return	void			

Operation **IGenerateDocumentationDlg::Fonts_SetDefaults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	void			

Operation **IGenerateDocumentationDlg::Fonts_SetFace**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	in	ENUMDocumentationFontSetting			
	strNewVal	in	string			
	return	return	void			

Operation **IGenerateDocumentationDlg::Fonts_SetItalic**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	in	ENUMDocumentationFontSetting			
	bVal	in	bool			
	return	return	void			

Operation **IGenerateDocumentationDlg::Fonts_SetSize**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	in	ENUMDocumentationFontSetting			
	nNewVal	in	int			
	return	return	void			

Operation **IGenerateDocumentationDlg::Fonts_SetTextColor**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	in	ENUMDocumentationFontSetting			
	nNewVal	in	int			
	return	return	void			

Operation **IGenerateDocumentationDlg::Fonts_SetUnderline**

parameter	name	direction	type	type modifier	multiplicity	default
	nSetting	in	ENUMDocumentationFontSetting			
	bVal	in	bool			
	return	return	void			

Operation **IGenerateDocumentationDlg::GenerateLinksToLocalFiles**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMDocumentationFilePathKind			

Operation **IGenerateDocumentationDlg::Include_Diagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IKindSelectionList			

Operation **IGenerateDocumentationDlg::Include_Elements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IKindSelectionList			

Operation **IGenerateDocumentationDlg::Include_IncludedPredefinedSubprojects**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Include_IncludedSubprojects**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Include_Index**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Include_NamedElementsOnly**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::Include_SelectAllDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IGenerateDocumentationDlg::Include_SelectAllElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IGenerateDocumentationDlg::Include_SelectAllKindsOf**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKindName	in	string			
	bVal	in	bool			
	return	return	void			

Operation **IGenerateDocumentationDlg::Include_SelectDefaults**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IGenerateDocumentationDlg::Include_SelectKind**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKindName	in	string			
	bVal	in	bool			
	return	return	void			

Operation **IGenerateDocumentationDlg::Include_SelectNoDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IGenerateDocumentationDlg::Include_SelectNoElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IGenerateDocumentationDlg::Include_UnknownExternals**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::OutputFormat**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMDocumentationOutputFormat			

Operation **IGenerateDocumentationDlg::ShowResultFileAfterGeneration**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::SplitOutputToMultipleFiles**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateDocumentationDlg::SPSFile**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IGenerateDocumentationDlg::UseFixedDesign**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

UModelAPI - IGenerateSequenceDiagramDlg**Interface IGenerateSequenceDiagramDlg**

diagram	«interface» IGenerateSequenceDiagramDlg «GetAccessor, SetAccessor, property» TypeIgnoreList():string «GetAccessor, SetAccessor, property» MaximalInvocationDepth():int «GetAccessor, SetAccessor, property» ShowUnknownInvocations():bool «GetAccessor, SetAccessor, property» ShowCodeInNotes():bool «GetAccessor, SetAccessor, property» ShowEmptyCombinedFragments():bool «GetAccessor, SetAccessor, property» ShowCodeOfMessagesDisplayedDirectlyBelow():bool «GetAccessor, SetAccessor, property» AddNotesOnSeparateLayer():bool «GetAccessor, SetAccessor, property» UseDedicatedLineForStaticCalls():bool «GetAccessor, SetAccessor, property» UseSpecialColorForNotesOfNotDisplayableInvocations():bool «GetAccessor, SetAccessor, property» NotDisplayableInvocationNoteColor():string «GetAccessor, SetAccessor, property» DiagramOwner():IUMLElement «GetAccessor, SetAccessor, property» AutomaticallyUpdateDiagramWhenModelsUpdatedFromCode():bool «GetAccessor, SetAccessor, property» SplitIntoSmallerDiagrams():bool «GetAccessor, SetAccessor, property» OperationIgnoreList():string	
hierarchy	<pre> classDiagram class IDialog class IGenerateSequenceDiagramDlg IDialog < -- IGenerateSequenceDiagramDlg </pre>	
typedElements	Interface IDialogs Interface IDocument	Operation GenerateSequenceDiagramDlg Operation GenerateSequenceDiagram

Operation IGenerateSequenceDiagramDlg::AddNotesOnSeparateLayer

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation**IGenerateSequenceDiagramDlg::AutomaticallyUpdateDiagramWhenModelsUpdatedFromCode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IGenerateSequenceDiagramDlg::DiagramOwner

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLElement			

Operation IGenerateSequenceDiagramDlg::MaximalInvocationDepth

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IGenerateSequenceDiagramDlg::NotDisplayableInvocationNoteColor**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IGenerateSequenceDiagramDlg::OperationIgnoreList**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IGenerateSequenceDiagramDlg::ShowCodeInNotes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateSequenceDiagramDlg::ShowCodeOfMessagesDisplayedDirectlyBelow**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateSequenceDiagramDlg::ShowEmptyCombinedFragments**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateSequenceDiagramDlg::ShowUnknownInvocations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateSequenceDiagramDlg::SplitIntoSmallerDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateSequenceDiagramDlg::TypeIgnoreList**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

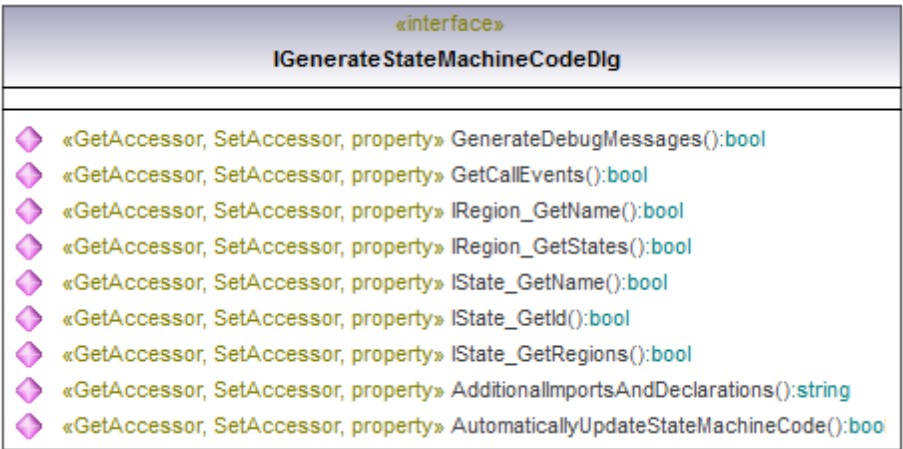
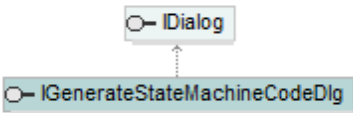
Operation **IGenerateSequenceDiagramDlg::UseDedicatedLineForStaticCalls**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation
IGenerateSequenceDiagramDlg::UseSpecialColorForNotesOfNotDisplayableInvocations

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

UModelAPI - IGenerateStateMachineCodeDlg**Interface IGenerateStateMachineCodeDlg**

diagram		
hierarch y		
typedEle ments	Interface IDialogs	Operation GenerateStateMachineCodeDlg
	Interface IDocument	Operation GenerateStateMachineCode

Operation IGenerateStateMachineCodeDlg::AdditionalImportsAndDeclarations

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation IGenerateStateMachineCodeDlg::AutomaticallyUpdateStateMachineCode

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IGenerateStateMachineCodeDlg::GenerateDebugMessages

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IGenerateStateMachineCodeDlg::GetCallEvents

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IGenerateStateMachineCodeDlg::IRegion_GetName

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IGenerateStateMachineCodeDlg::IRegion_GetStates

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateStateMachineCodeDlg::IState_GetId**

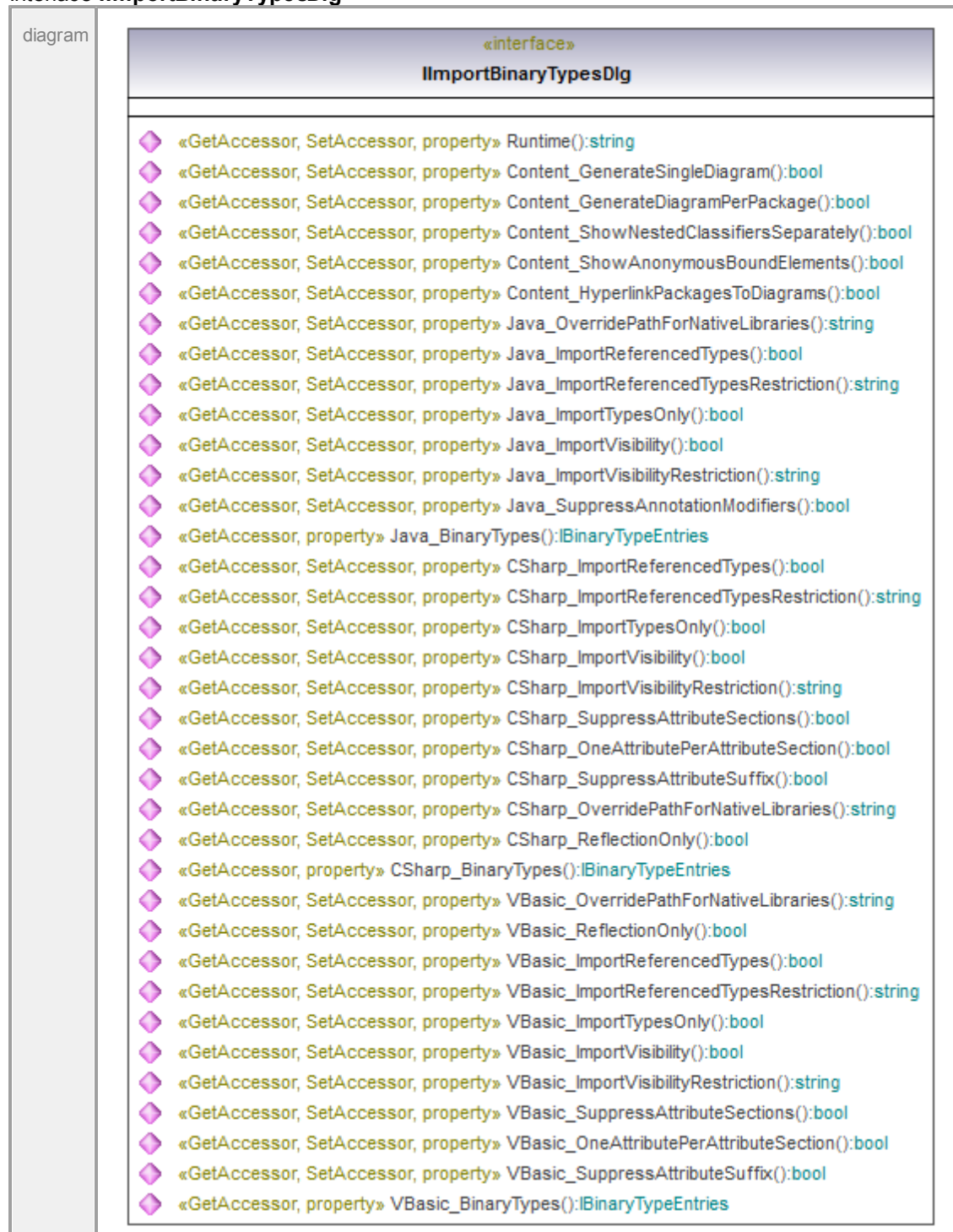
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

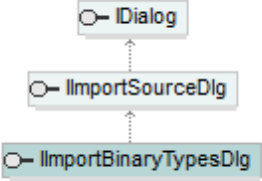
Operation **IGenerateStateMachineCodeDlg::IState_GetName**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IGenerateStateMachineCodeDlg::IState_GetRegions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

UModelAPI - IImportBinaryTypesDlg**Interface IImportBinaryTypesDlg**

hierarc hy						
typedE lements	Interface IDialogs Interface IDocument		Operation ImportBinaryTypesDlg Operation ImportBinaryTypes			

Operation **ImportBinaryTypesDlg::Content_GenerateDiagramPerPackage**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **ImportBinaryTypesDlg::Content_GenerateSingleDiagram**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **ImportBinaryTypesDlg::Content_HyperlinkPackagesToDiagrams**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **ImportBinaryTypesDlg::Content_ShowAnonymousBoundElements**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **ImportBinaryTypesDlg::Content_ShowNestedClassifiersSeparately**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **ImportBinaryTypesDlg::CSharp_BinaryTypes**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IBinaryTypeEnt ries			

Operation **ImportBinaryTypesDlg::CSharp_ImportReferencedTypes**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **ImportBinaryTypesDlg::CSharp_ImportReferencedTypesRestriction**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **ImportBinaryTypesDlg::CSharp_ImportTypesOnly**

paramete	name	direction	type	type modifier	multiplicity	default

r	return	return	bool			

Operation **ImportBinaryTypesDlg::CSharp_ImportVisibility**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::CSharp_ImportVisibilityRestriction**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::CSharp_OneAttributePerAttributeSection**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::CSharp_OverridePathForNativeLibraries**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::CSharp_ReflectionOnly**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::CSharp_SuppressAttributeSections**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::CSharp_SuppressAttributeSuffix**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::Java_BinaryTypes**

paramete r	name return	direction return	type IBinaryTypeEntities	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::Java_ImportReferencedTypes**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::Java_ImportReferencedTypesRestriction**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::Java_ImportTypesOnly**

paramete	name	direction	type	type modifier	multiplicity	default

r	return	return	bool			

Operation **ImportBinaryTypesDlg::Java_ImportVisibility**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::Java_ImportVisibilityRestriction**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::Java_OverridePathForNativeLibraries**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::Java_SuppressAnnotationModifiers**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::Runtime**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::VBasic_BinaryTypes**

paramete r	name return	direction return	type IBinaryTypeEntities	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::VBasic_ImportReferencedTypes**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::VBasic_ImportReferencedTypesRestriction**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::VBasic_ImportTypesOnly**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::VBasic_ImportVisibility**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ImportBinaryTypesDlg::VBasic_ImportVisibilityRestriction**

paramete	name	direction	type	type modifier	multiplicity	default

r	return	return	string

Operation **ImportBinaryTypesDlg::VBasic_OneAttributePerAttributeSection**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportBinaryTypesDlg::VBasic_OverridePathForNativeLibraries**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **ImportBinaryTypesDlg::VBasic_ReflectionOnly**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportBinaryTypesDlg::VBasic_SuppressAttributeSections**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportBinaryTypesDlg::VBasic_SuppressAttributeSuffix**

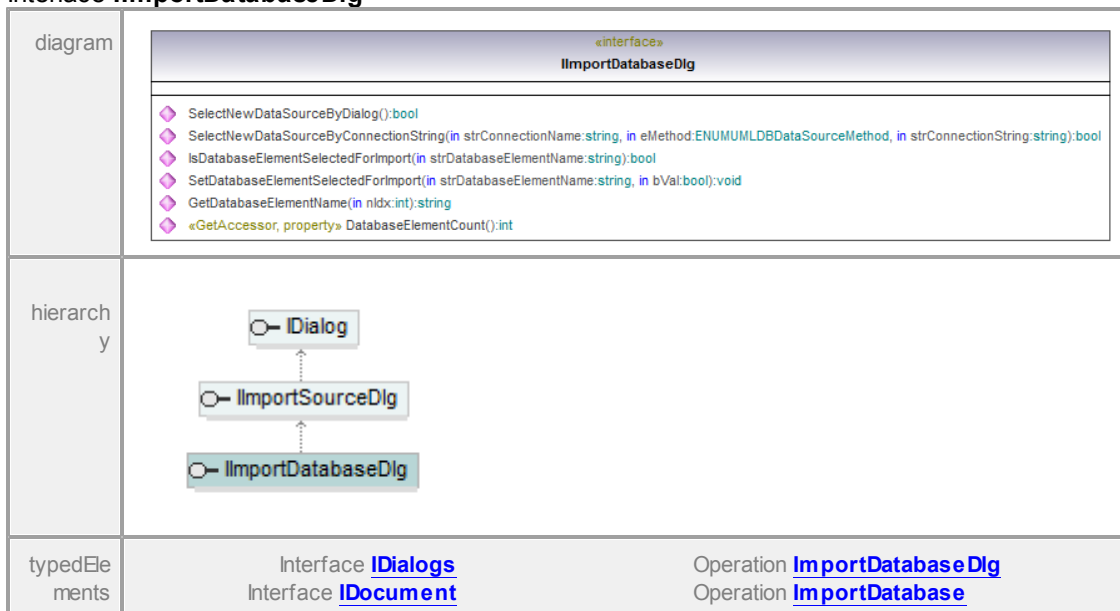
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IImportDatabaseDlg

Interface **IImportDatabaseDlg**



Operation **IImportDatabaseDlg::DatabaseElementCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **ImportDatabaseDlg::GetDatabaseElementName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	string			

Operation **ImportDatabaseDlg::IsDatabaseElementSelectedForImport**

parameter	name	direction	type	type modifier	multiplicity	default
	strDatabaseElementName		string			
	return	return	bool			

Operation **ImportDatabaseDlg::SelectNewDataSourceByConnectionString**

parameter	name	direction	type	type modifier	multiplicity	default
	strConnectionString	in	string			
	eMethod	in	ENUM UMLDBDataSourceMethod			
	strConnectionString	in	string			
	return	return	bool			

Operation **ImportDatabaseDlg::SelectNewDataSourceByDialog**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **ImportDatabaseDlg::SetDatabaseElementSelectedForImport**

parameter	name	direction	type	type modifier	multiplicity	default
	strDatabaseElementName		string			
	bVal	in	bool			
	return	return	void			

UModelAPI - IImportSourceDirectoryDlg**Interface IImportSourceDirectoryDlg**

diagram		
hierarch y		
typedEle ments	Interface IDialogs Interface IDocument	Operation IImportSourceDirectoryDlg Operation IImportSourceDirectory

Operation IImportSourceDirectoryDlg::Content_GenerateDiagramPerPackage

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IImportSourceDirectoryDlg::Content_GenerateSingleDiagram

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IImportSourceDirectoryDlg::Content_HyperlinkPackagesToDiagrams

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IImportSourceDirectoryDlg::Content_ShowAnonymousBoundElements

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IImportSourceDirectoryDlg::Content_ShowNestedClassifiersSeparately

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **ImportSourceDirectoryDlg::Directory**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **ImportSourceDirectoryDlg::ImportDirectoriesRelative**

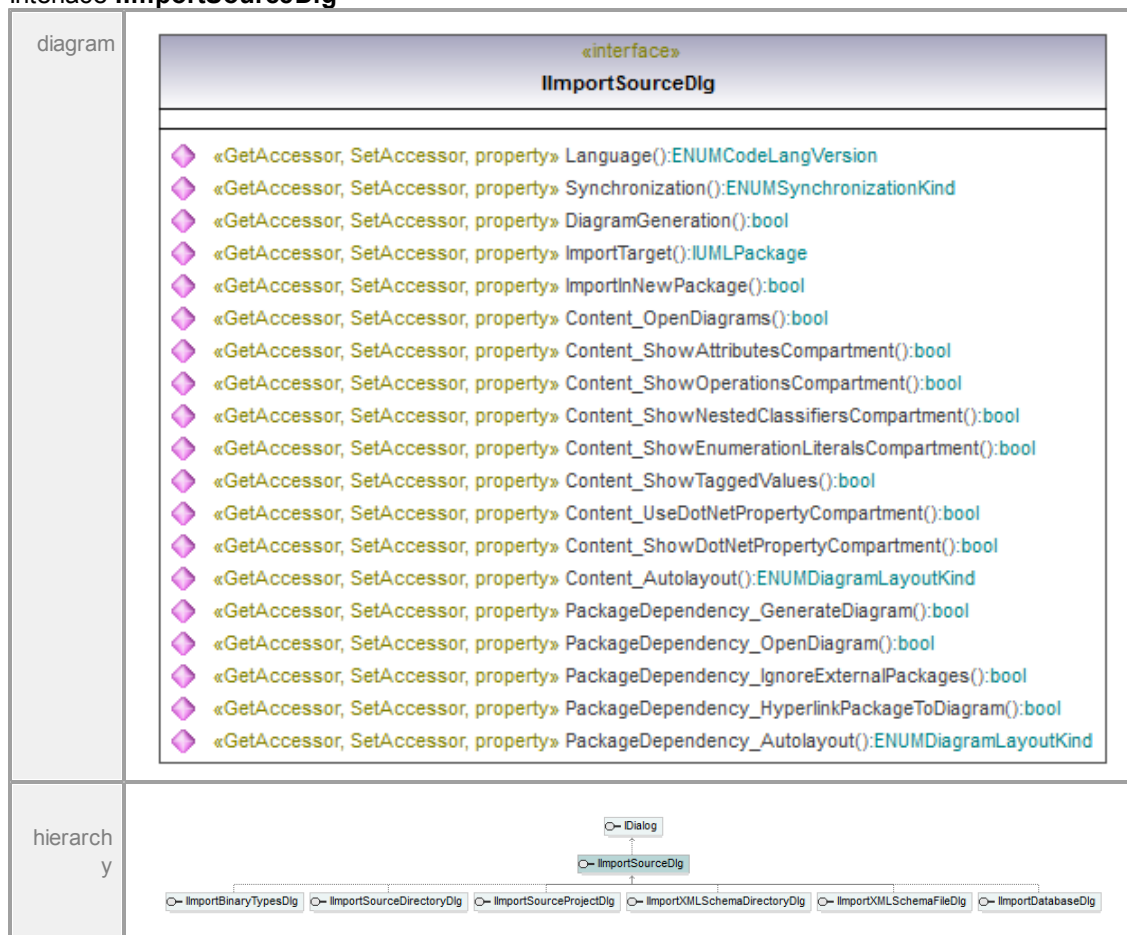
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDirectoryDlg::ProcessSubdirectories**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ImportSourceDlgInterface **ImportSourceDlg**Operation **ImportSourceDlg::Content_Autolayout**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	ENUMDiagramLayoutKind			

Operation **ImportSourceDlg::Content_OpenDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::Content_ShowAttributesCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::Content_ShowDotNetPropertyCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::Content_ShowEnumerationLiteralsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::Content_ShowNestedClassifiersCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::Content_ShowOperationsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::Content_ShowTaggedValues**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::Content_UseDotNetPropertyCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::DiagramGeneration**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::ImportInNewPackage**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::ImportTarget**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	IUMLPackage			

Operation **ImportSourceDlg::Language**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMCodeLanguageVersion			

Operation **ImportSourceDlg::PackageDependency_Autolayout**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMDiagramLayoutKind			

Operation **ImportSourceDlg::PackageDependency_GenerateDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::PackageDependency_HyperlinkPackageToDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::PackageDependency_IgnoreExternalPackages**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

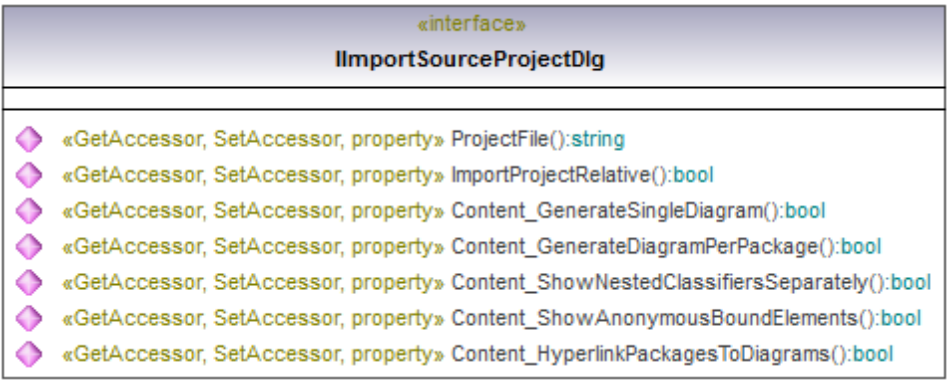
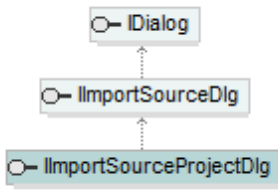
Operation **ImportSourceDlg::PackageDependency_OpenDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceDlg::Synchronization**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMSynchronizationKind			

UModelAPI - IImportSourceProjectDlg**Interface IImportSourceProjectDlg**

diagram		
hierarch y		
typedEle ments	Interface IDialogs Interface IDocument	Operation IImportSourceProjectDlg Operation IImportSourceProject

Operation IImportSourceProjectDlg::Content_GenerateDiagramPerPackage

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IImportSourceProjectDlg::Content_GenerateSingleDiagram

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IImportSourceProjectDlg::Content_HyperlinkPackagesToDiagrams

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IImportSourceProjectDlg::Content_ShowAnonymousBoundElements

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IImportSourceProjectDlg::Content_ShowNestedClassifiersSeparately

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IImportSourceProjectDlg::ImportProjectRelative

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ImportSourceProjectDlg::ProjectFile**

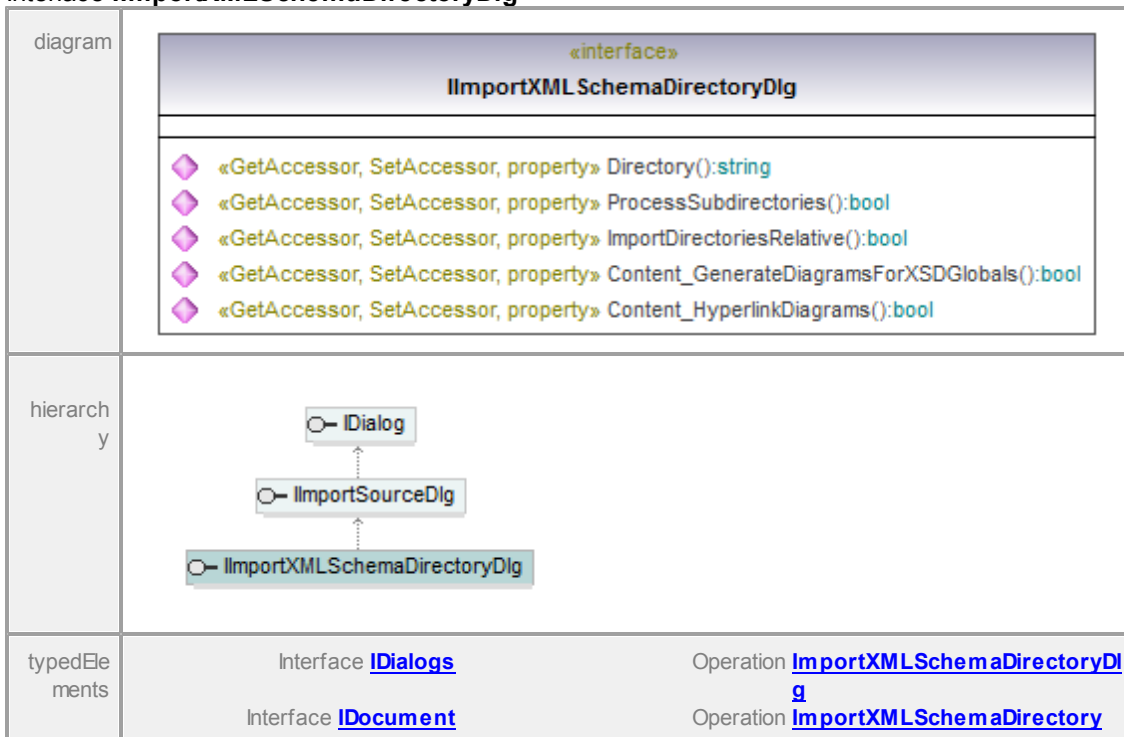
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IImportXMLSchemaDirectoryDlg

Interface **IImportXMLSchemaDirectoryDlg**



Operation **IImportXMLSchemaDirectoryDlg::Content_GenerateDiagramsForXSDGlobals**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IImportXMLSchemaDirectoryDlg::Content_HyperlinkDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IImportXMLSchemaDirectoryDlg::Directory**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **ImportXMLSchemaDirectoryDlg::ImportDirectoriesRelative**

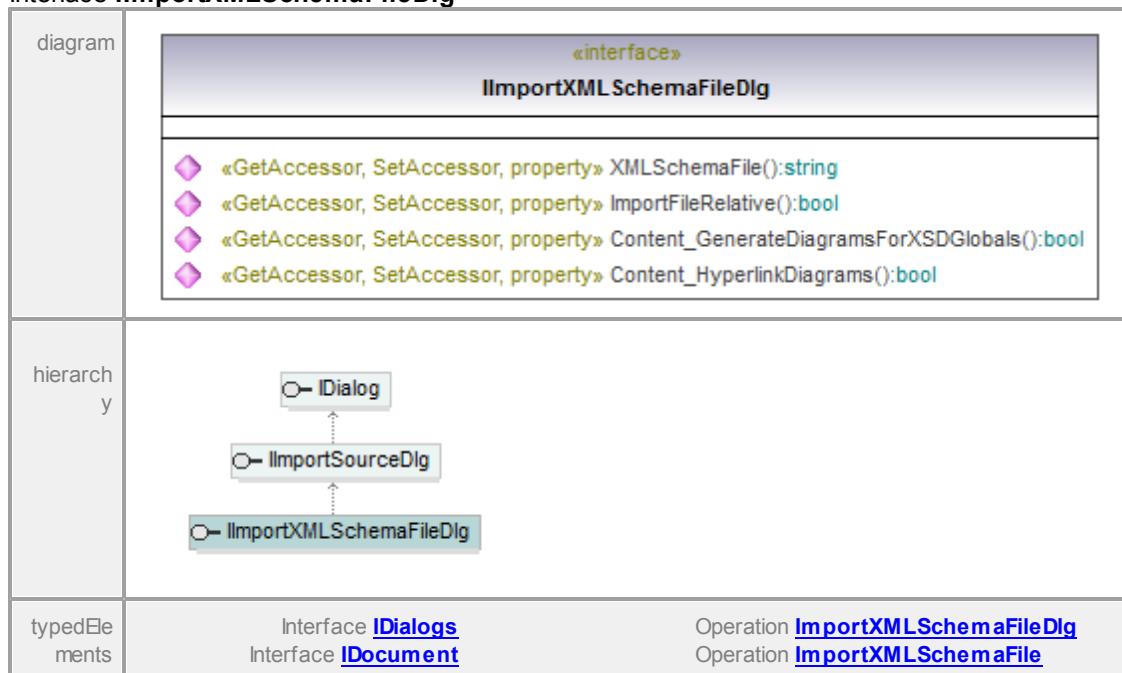
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **ImportXMLSchemaDirectoryDlg::ProcessSubdirectories**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IImportXMLSchemaFileDialogInterface **IImportXMLSchemaFileDialog**Operation **IImportXMLSchemaFileDialog::Content_GenerateDiagramsForXSDGlobals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IImportXMLSchemaFileDialog::Content_HyperlinkDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IImportXMLSchemaFileDialog::ImportFileRelative**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IImportXMLSchemaFileDialog::XMLSchemaFile**

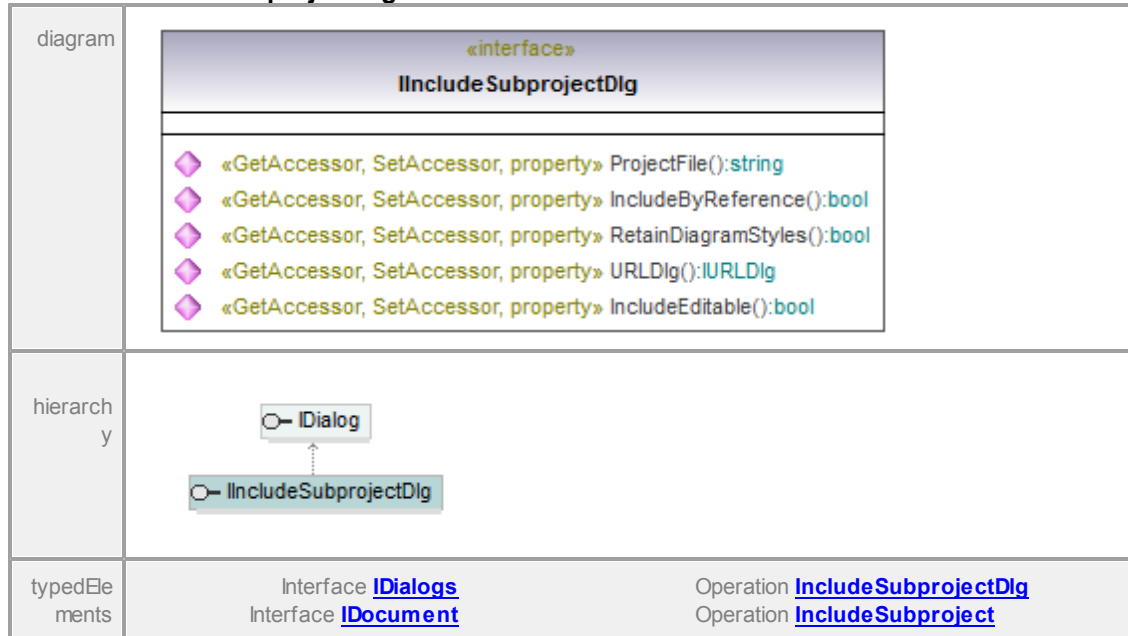
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IIncludeSubprojectDlg

Interface IIncludeSubprojectDlg



Operation IIncludeSubprojectDlg::IncludeByReference

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IIncludeSubprojectDlg::IncludeEditable

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IIncludeSubprojectDlg::ProjectFile

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation IIncludeSubprojectDlg::RetainDiagramStyles

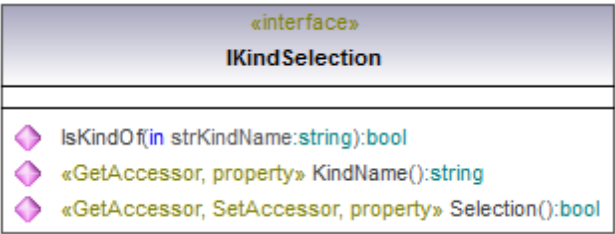
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IIncludeSubprojectDlg::URLDlg

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IURLDlg			

UModelAPI - IKindSelection

Interface IKindSelection

diagram	 <pre>graph TD subgraph IKindSelection direction TB I1[«interface» IKindSelection] I1 --> M1[IsKindOf(in strKindName:string):bool] I1 --> M2[«GetAccessor, property» KindName():string] I1 --> M3[«GetAccessor, SetAccessor, property» Selection():bool] end</pre>					
typedElements	Interface IKindSelectionList Operation Item					

Operation IKindSelection::IsKindOf

parameter	name	direction	type	type modifier	multiplicity	default
	strKindName	in	string			
	return	return	bool			

Operation IKindSelection::KindName

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation IKindSelection::Selection

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

UModelAPI - IKindSelectionList

Interface IKindSelectionList

diagram	 <pre>graph TD subgraph IKindSelectionList direction TB I2[«interface» IKindSelectionList] I2 --> M4[«GetAccessor, property» Count():int] I2 --> M5[Item(in nIdx:int):IKindSelection] I2 --> M6[«GetAccessor, property» Application():IDispatch] I2 --> M7[«GetAccessor, property» Parent():IDispatch] end</pre>					
typedElements	Interface IGenerateDocumentationDlg Operation Include_Diagrams Include_Elements					

Operation **IKindSelectionList::Application**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **IKindSelectionList::Count**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IKindSelectionList::Item**

parameter	name	direction	type	type modifier	multiplicity	default
r	idx return	in return	int IKindSelection			

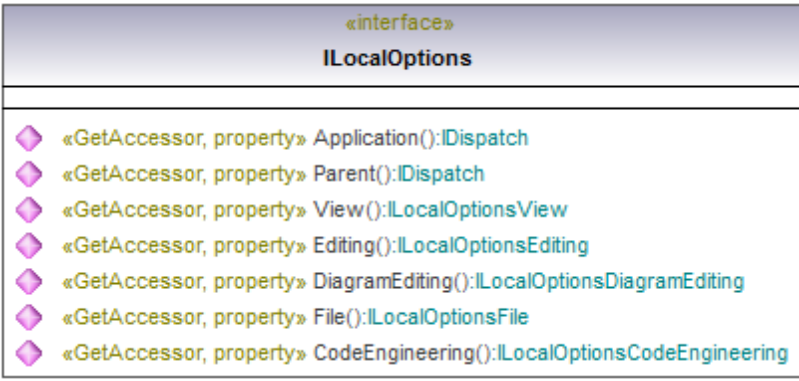
Operation **IKindSelectionList::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ILocalOptionsInterface **ILocalOptions**

diagram						
typedElements	Interface IApplication Operation Options					

Operation **ILocalOptions::Application**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **ILocalOptions::CodeEngineering**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ILocalOptionsCodeEngineering			

Operation **ILocalOptions::DiagramEditing**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ILocalOptionsDiagramEditing			

Operation **ILocalOptions::Editing**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ILocalOptionsEditing			

Operation **ILocalOptions::File**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ILocalOptionsFile			

Operation **ILocalOptions::Parent**

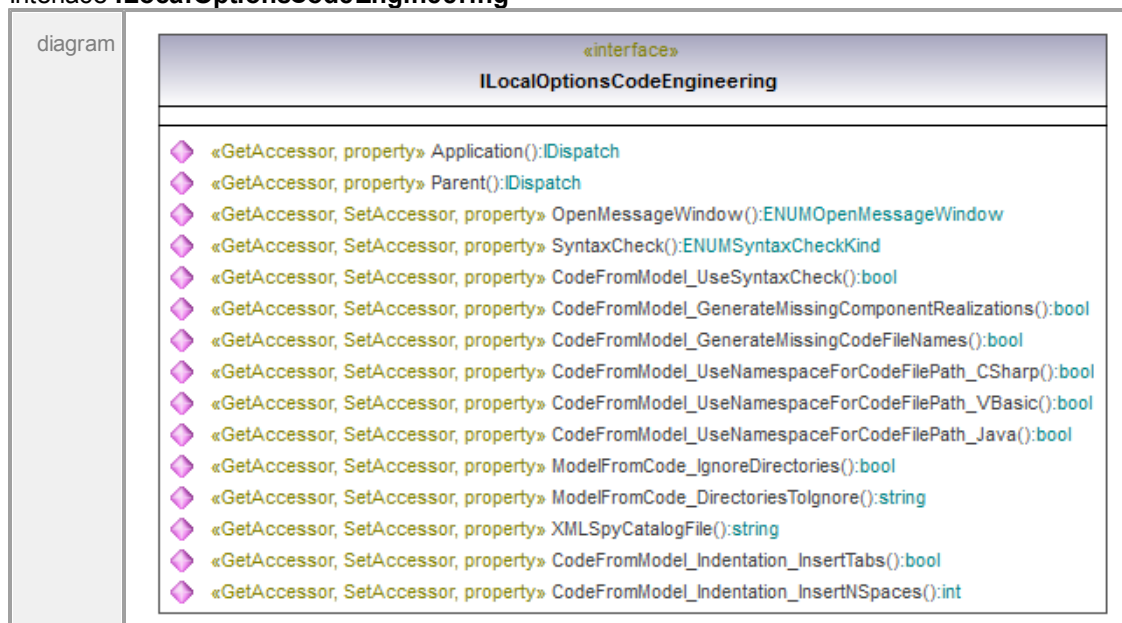
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation **ILocalOptions::View**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ILocalOptionsView			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ILocalOptionsCodeEngineeringInterface **ILocalOptionsCodeEngineering**

typedElements	Interface ILocalOptions	Operation CodeEngineering

Operation **ILocalOptionsCodeEngineering::Application**

parameter	name return	direction return	type IDispatch	type modifier	multiplicity	default

Operation

ILocalOptionsCodeEngineering::CodeFromModel_GenerateMissingCodeFileNames

parameter	name return	direction return	type bool	type modifier	multiplicity	default

Operation

ILocalOptionsCodeEngineering::CodeFromModel_GenerateMissingComponentRealizations

parameter	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ILocalOptionsCodeEngineering::CodeFromModel_Indentation_InsertNSpaces**

parameter	name return	direction return	type int	type modifier	multiplicity	default

Operation **ILocalOptionsCodeEngineering::CodeFromModel_Indentation_InsertTabs**

parameter	name return	direction return	type bool	type modifier	multiplicity	default

Operation

ILocalOptionsCodeEngineering::CodeFromModel_UseNamespaceForCodeFilePath_CSharp

parameter	name return	direction return	type bool	type modifier	multiplicity	default

Operation

ILocalOptionsCodeEngineering::CodeFromModel_UseNamespaceForCodeFilePath_Java

parameter	name return	direction return	type bool	type modifier	multiplicity	default

Operation

ILocalOptionsCodeEngineering::CodeFromModel_UseNamespaceForCodeFilePath_VBasic

parameter	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ILocalOptionsCodeEngineering::CodeFromModel_UseSyntaxCheck**

parameter	name return	direction return	type bool	type modifier	multiplicity	default

Operation **ILocalOptionsCodeEngineering::ModelFromCode_DirectoriesToIgnore**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **ILocalOptionsCodeEngineering::ModelFromCode_IgnoreDirectories**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsCodeEngineering::OpenMessageWindow**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMOpenMessageWindow			

Operation **ILocalOptionsCodeEngineering::Parent**

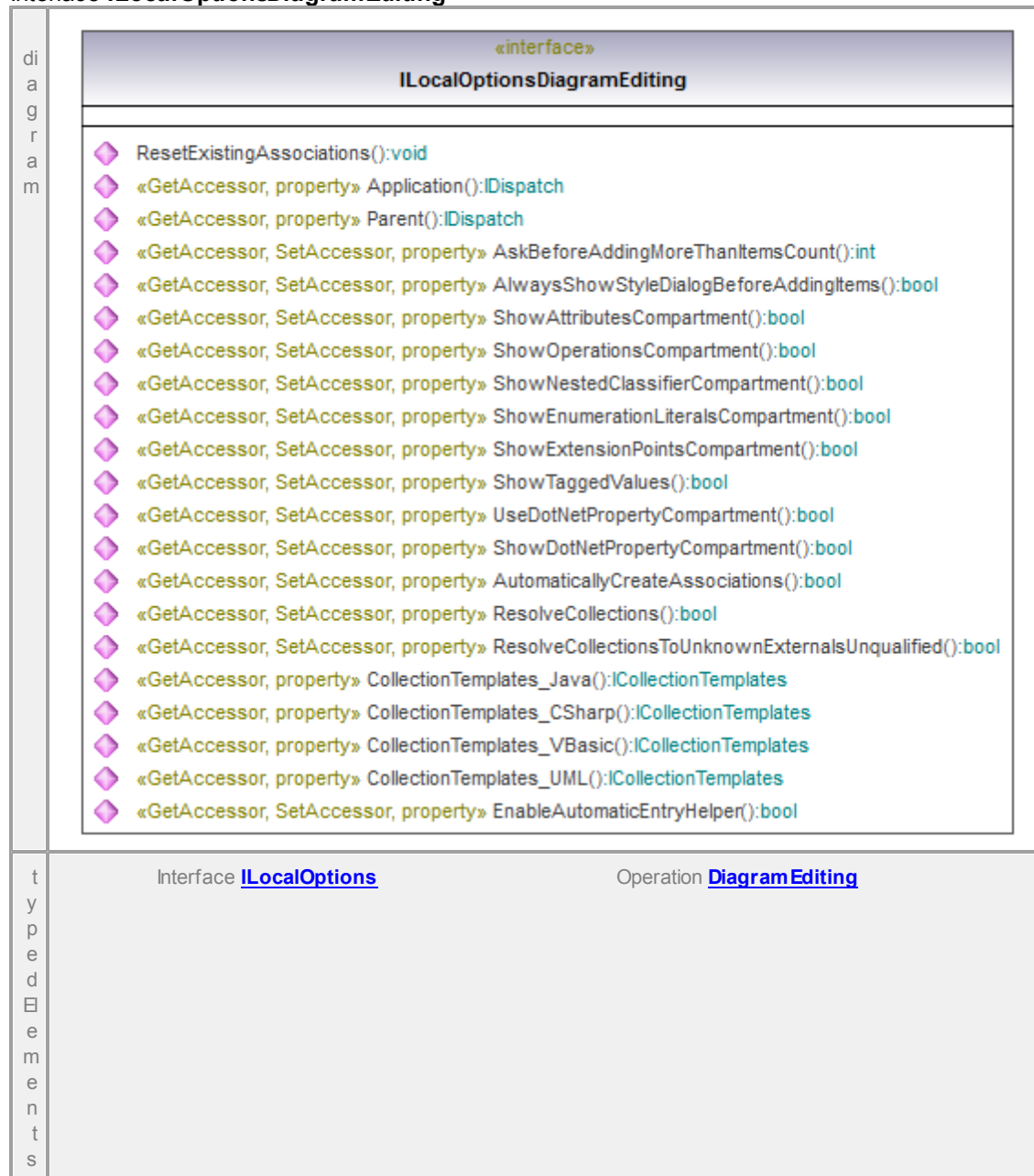
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **ILocalOptionsCodeEngineering::SyntaxCheck**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMSyntaxCheckKind			

Operation **ILocalOptionsCodeEngineering::XMLSpyCatalogFile**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

UModelAPI - ILocalOptionsDiagramEditing**Interface ILocalOptionsDiagramEditing****Operation ILocalOptionsDiagramEditing::AlwaysShowStyleDialogBeforeAddingItems**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation ILocalOptionsDiagramEditing::Application

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation **ILocalOptionsDiagramEditing::AskBeforeAddingMoreThanItemsCount**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **ILocalOptionsDiagramEditing::AutomaticallyCreateAssociations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_CSharp**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ICollectionTemplates			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_Java**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ICollectionTemplates			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_UML**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ICollectionTemplates			

Operation **ILocalOptionsDiagramEditing::CollectionTemplates_VBasic**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ICollectionTemplates			

Operation **ILocalOptionsDiagramEditing::EnableAutomaticEntryHelper**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsDiagramEditing::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **ILocalOptionsDiagramEditing::ResetExistingAssociations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **ILocalOptionsDiagramEditing::ResolveCollections**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation
ILocalOptionsDiagramEditing::ResolveCollectionsToUnknownExternalsUnqualified

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsDiagramEditing::ShowAttributesCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsDiagramEditing::ShowDotNetPropertyCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsDiagramEditing::ShowEnumerationLiteralsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsDiagramEditing::ShowExtensionPointsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsDiagramEditing::ShowNestedClassifierCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsDiagramEditing::ShowOperationsCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsDiagramEditing::ShowTaggedValues**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsDiagramEditing::UseDotNetPropertyCompartment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

UModelAPI - ILocalOptionsEditing**Interface ILocalOptionsEditing**

diagram	<pre> «interface» ILocalOptionsEditing «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch «GetAccessor, SetAccessor, property» OpenNewDiagrams():bool «GetAccessor, SetAccessor, property» PropertiesDefaultVisibility():ENUMUMLVisibilityKind «GetAccessor, SetAccessor, property» OperationsDefaultVisibility():ENUMUMLVisibilityKind «GetAccessor, SetAccessor, property» ComponentsDefaultCodeLangVersion():ENUMCodeLangVersion «GetAccessor, SetAccessor, property» ConstrainOwnerOfNewConstraints():bool «GetAccessor, SetAccessor, property» AskBeforeEraseFromProjectInDiagrams():bool «GetAccessor, SetAccessor, property» AskBeforeEraseFromProjectInFavoritesTree():bool «GetAccessor, SetAccessor, property» SyntaxErrorBubbleDisappearDelay():int </pre>
typedElements	Interface ILocalOptions Operation Editing

Operation ILocalOptionsEditing::Application

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation ILocalOptionsEditing::AskBeforeEraseFromProjectInDiagrams

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation ILocalOptionsEditing::AskBeforeEraseFromProjectInFavoritesTree

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation ILocalOptionsEditing::ComponentsDefaultCodeLangVersion

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ENUMCodeLangVersion			

Operation ILocalOptionsEditing::ConstrainOwnerOfNewConstraints

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation ILocalOptionsEditing::OpenNewDiagrams

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation ILocalOptionsEditing::OperationsDefaultVisibility

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	ENUMUMLVisibilityKind			

Operation **ILocalOptionsEditing::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **ILocalOptionsEditing::PropertiesDefaultVisibility**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLVisibilityKind			

Operation **ILocalOptionsEditing::SyntaxErrorBubbleDisappearDelay**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ILocalOptionsFileInterface **ILocalOptionsFile**

diagram					
typedElements	Interface ILocalOptions	Operation File			

Operation **ILocalOptionsFile::Application**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **ILocalOptionsFile::AskBeforeReloadChangedFile**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsFile::LoadAndSaveFavoritesWithProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation ILocalOptionsFile::LoadAndSaveOpenDiagramsWithProjectFile

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation ILocalOptionsFile::OpenLastProjectOnProgramStart

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation ILocalOptionsFile::Parent

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation ILocalOptionsFile::PrettyPrintFileContentWhenSaving

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation ILocalOptionsFile::WatchForFileChanges

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

UModelAPI - ILocalOptionsView

Interface ILocalOptionsView

diagram	«interface» ILocalOptionsView	
	«GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch «GetAccessor, SetAccessor, property» ShowProgramLogoOnStartup():bool «GetAccessor, SetAccessor, property» ShowProgramLogoOnPrint():bool «GetAccessor, SetAccessor, property» ShowProgramLogoOnDiagram():bool «GetAccessor, SetAccessor, property» FrameTitle():ENUMApplicationFrameTitle «GetAccessor, SetAccessor, property» ListClassifiersNotUsedInAnyDiagram():bool «GetAccessor, SetAccessor, property» ListRelationsNotUsedInAnyDiagram():bool «GetAccessor, SetAccessor, property» ListPackagesNotUsedInAnyDiagram():bool «GetAccessor, SetAccessor, property» ListInstanceSpecificationsNotUsedInAnyDiagram():bool «GetAccessor, SetAccessor, property» ListElementsNotUsedInAnyDiagram_IgnoreIncludedElements():bool «GetAccessor, SetAccessor, property» SelectFocusedDiagramItemInModelTree():bool «GetAccessor, SetAccessor, property» HierarchyWindow_NestingDepthUp():int «GetAccessor, SetAccessor, property» HierarchyWindow_NestingDepthDown():int «GetAccessor, SetAccessor, property» HierarchyWindow_ExpandItemsOnlyOnce():bool «GetAccessor, SetAccessor, property» AutolayoutHierarchic_MinXDistance():int «GetAccessor, SetAccessor, property» AutolayoutHierarchic_MinYDistance():int «GetAccessor, SetAccessor, property» AutolayoutHierarchic_GrowDirection():ENUMAutolayoutGrowDirectionKind «GetAccessor, SetAccessor, property» EnableSnapLines():bool	
typedElements	Interface ILocalOptions	Operation View

Operation ILocalOptionsView::Application

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation ILocalOptionsView::AutolayoutHierarchic_GrowDirection

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMAutolayoutGrowDirectionKind			

Operation ILocalOptionsView::AutolayoutHierarchic_MinXDistance

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation ILocalOptionsView::AutolayoutHierarchic_MinYDistance

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation ILocalOptionsView::EnableSnapLines

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsView::FrameTitle**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMApplicationFrameTitle			

Operation **ILocalOptionsView::HierarchyWindow_ExpandItemsOnlyOnce**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsView::HierarchyWindow_NestingDepthDown**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **ILocalOptionsView::HierarchyWindow_NestingDepthUp**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **ILocalOptionsView::ListClassifiersNotUsedInAnyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation

ILocalOptionsView::ListElementsNotUsedInAnyDiagram_IgnoreIncludedElements

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsView::ListInstanceSpecificationsNotUsedInAnyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsView::ListPackagesNotUsedInAnyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsView::ListRelationsNotUsedInAnyDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsView::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **ILocalOptionsView::SelectFocusedDiagramItemInModelTree**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsView::ShowProgramLogoOnDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsView::ShowProgramLogoOnPrint**

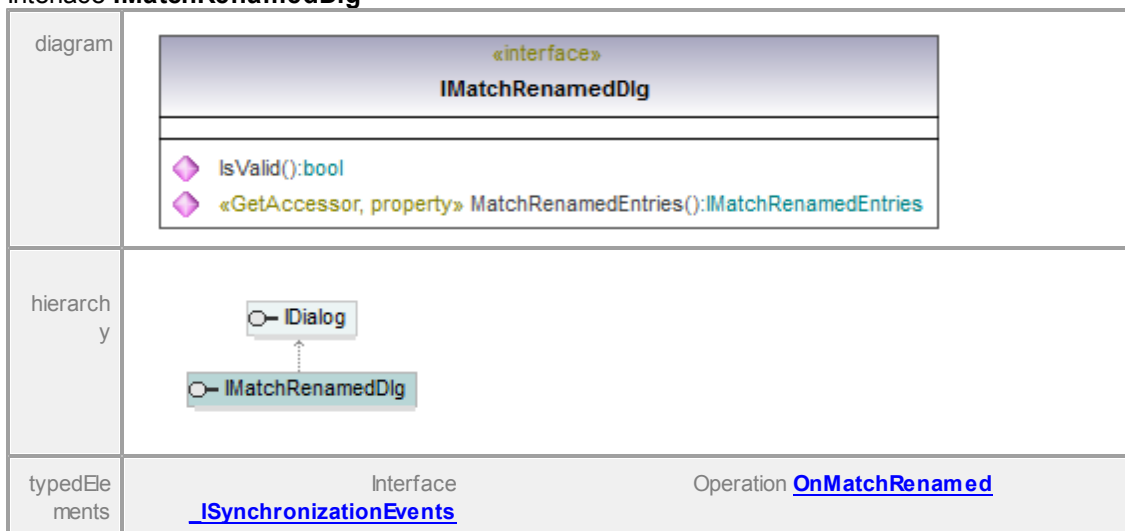
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **ILocalOptionsView::ShowProgramLogoOnStartup**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IMatchRenamedDlgInterface **IMatchRenamedDlg**Operation **IMatchRenamedDlg::IsValid**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

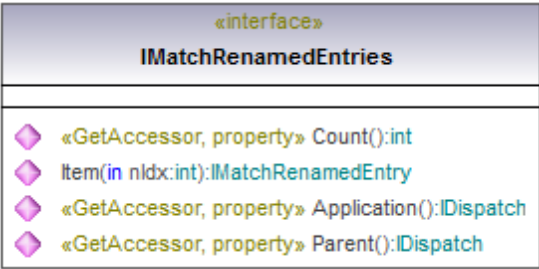
Operation **IMatchRenamedDlg::MatchRenamedEntries**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IMatchRenamedEntries			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IMatchRenamedEntries**Interface IMatchRenamedEntries**

diagram		
typedElements	Interface IMatchRenamedDlg	Operation MatchRenamedEntries

Operation IMatchRenamedEntries::Application

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation IMatchRenamedEntries::Count

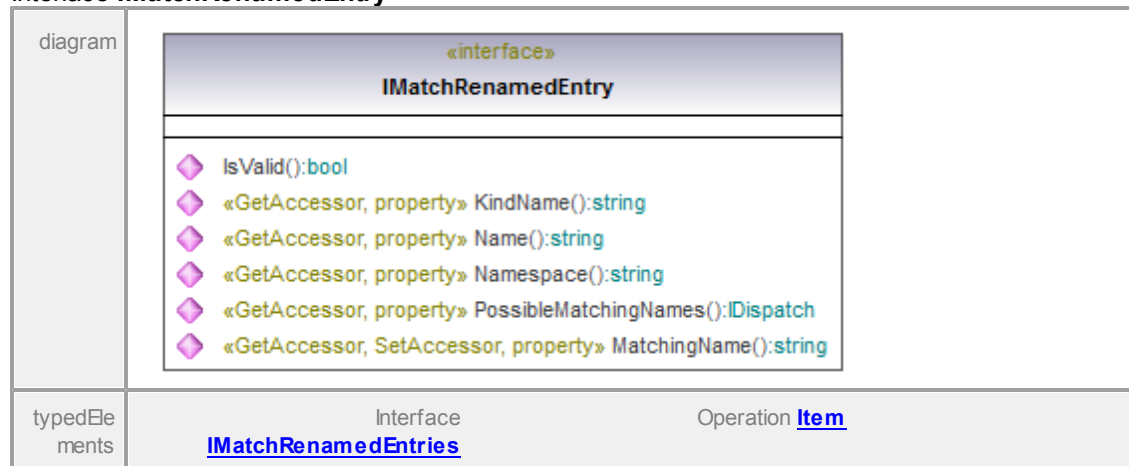
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation IMatchRenamedEntries::Item

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	IMatchRenamedEntry			

Operation IMatchRenamedEntries::Parent

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

UModelAPI - IMatchRenamedEntry**Interface IMatchRenamedEntry****Operation IMatchRenamedEntry::IsValid**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation IMatchRenamedEntry::KindName

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation IMatchRenamedEntry::MatchingName

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation IMatchRenamedEntry::Name

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation IMatchRenamedEntry::Namespace

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation IMatchRenamedEntry::PossibleMatchingNames

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			
documentation	Returns an array of values of type string .					

UModelAPI - IModelTransformationDlg

Interface IModelTransformationDlg

diagram	«interface» IModelTransformationDlg «GetAccessor, SetAccessor, property» SourcePackage():IUMLPackage «GetAccessor, SetAccessor, property» TargetPackage():IUMLPackage «GetAccessor, SetAccessor, property» TransformInNewPackage():bool «GetAccessor, SetAccessor, property» TransformFromLanguage():ENUMCodeLang «GetAccessor, SetAccessor, property» TransformToLanguage():ENUMCodeLang «GetAccessor, SetAccessor, property» Synchronization():ENUMSynchronizationKind «GetAccessor, SetAccessor, property» TransformClassDiagrams():bool «GetAccessor, SetAccessor, property» OpenNewDiagrams():bool «GetAccessor, SetAccessor, property» GenerateComponentsAndComponentRealizations():bool «GetAccessor, SetAccessor, property» AutomaticallyUpdateTransformationAfterUpdateModelFromCode():bool «GetAccessor, SetAccessor, property» AutomaticallyUpdateTransformationBeforeUpdateCodeFromModel():bool «GetAccessor, property» TypeMappings():IModelTransformationTypeMappings

Operation

IModelTransformationDlg::AutomaticallyUpdateTransformationAfterUpdateModelFromCode

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation

IModelTransformationDlg::AutomaticallyUpdateTransformationBeforeUpdateCodeFromModel

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation IModelTransformationDlg::GenerateComponentsAndComponentRealizations

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation IModelTransformationDlg::OpenNewDiagrams

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation IModelTransformationDlg::SourcePackage

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLPackage			

Operation **IModelTransformationDlg::Synchronization**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMSynchronizationKind			

Operation **IModelTransformationDlg::TargetPackage**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLPackage			

Operation **IModelTransformationDlg::TransformClassDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IModelTransformationDlg::TransformFromLanguage**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMCodeLanguage			

Operation **IModelTransformationDlg::TransformInNewPackage**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IModelTransformationDlg::TransformToLanguage**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMCodeLanguage			

Operation **IModelTransformationDlg::TypeMappings**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IModelTransformationTypeMappings			

UModelAPI - IModelTransformationTypeMapping

Interface IModelTransformationTypeMapping

diagram	«interface» IModelTransformationTypeMapping «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch «GetAccessor, SetAccessor, property» FromType():string «GetAccessor, SetAccessor, property» ToType():string		
typedElements	Interface IModelTransformationTypeMappings	Operation InsertItemAt Item	

Operation IModelTransformationTypeMapping::Application

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation IModelTransformationTypeMapping::FromType

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

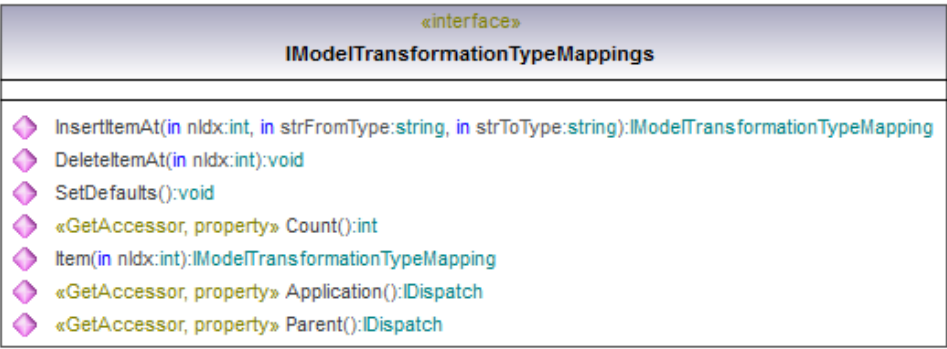
Operation IModelTransformationTypeMapping::Parent

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation IModelTransformationTypeMapping::ToType

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

UModelAPI - IModelTransformationTypeMappings**Interface IModelTransformationTypeMappings**

diagram	 <pre> «interface» IModelTransformationTypeMappings +InsertItemAt(in nldx:int, in strFromType:string, in strToType:string):IModelTransformationTypeMapping +DeleteItemAt(in nldx:int):void +SetDefaults():void +«GetAccessor, property» Count():int +Item(in nldx:int):IModelTransformationTypeMapping +«GetAccessor, property» Application():IDispatch +«GetAccessor, property» Parent():IDispatch </pre>					
typedElements	Interface	Operation TypeMappings				
	IModelTransformationDlg					

Operation IModelTransformationTypeMappings::Application

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation IModelTransformationTypeMappings::Count

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation IModelTransformationTypeMappings::DeleteItemAt

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	void			

Operation IModelTransformationTypeMappings::InsertItemAt

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	strFromType	in	string			
	strToType	in	string			
	return	return	IModelTransformationTypeMapping			

Operation IModelTransformationTypeMappings::Item

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	IModelTransformationTypeMapping			

Operation IModelTransformationTypeMappings::Parent

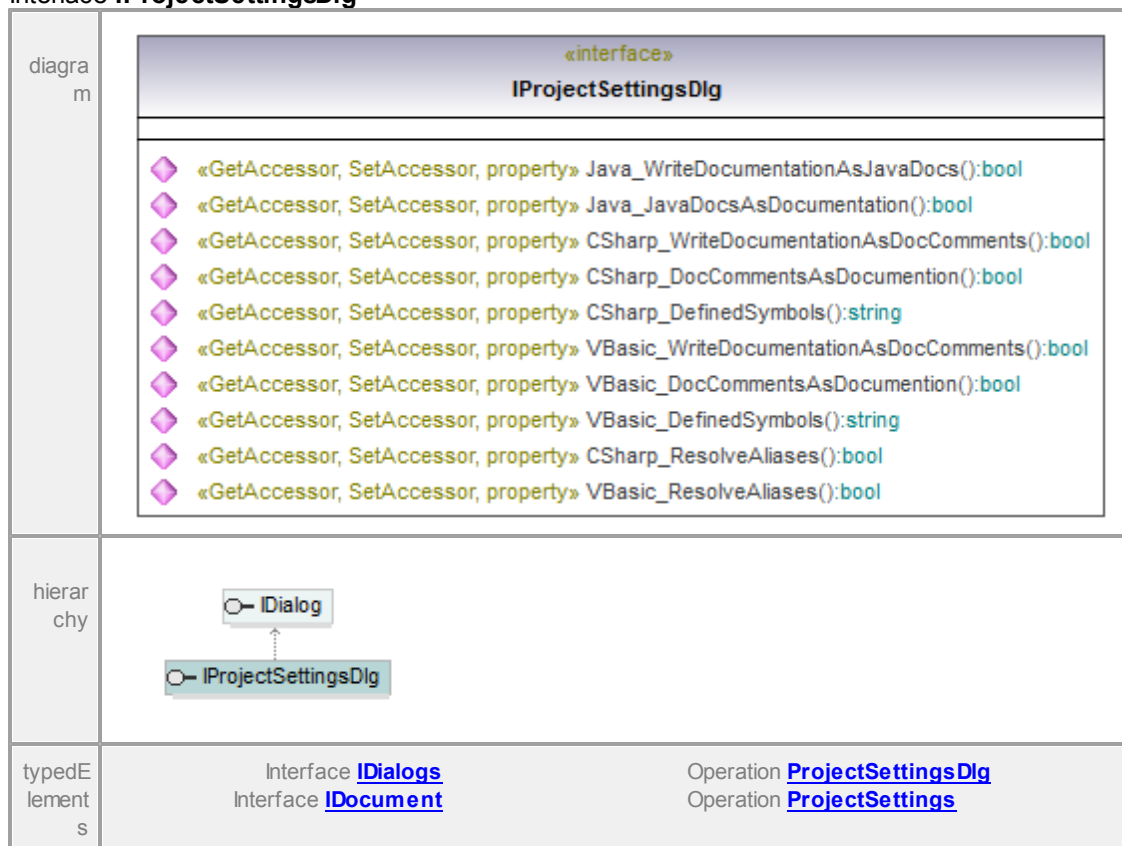
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation **IModelTransformationTypeMappings::SetDefaults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	void			

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IProjectSettingsDlgInterface **IProjectSettingsDlg**Operation **IProjectSettingsDlg::CSharp_DefinedSymbols**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IProjectSettingsDlg::CSharp_DocCommentsAsDocumentation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IProjectSettingsDlg::CSharp_ResolveAliases**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IProjectSettingsDlg::CSharp_WriteDocumentationAsDocComments**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IProjectSettingsDlg::Java_JavaDocsAsDocumentation**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IProjectSettingsDlg::Java_WriteDocumentationAsJavaDocs**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IProjectSettingsDlg::VBasic_DefinedSymbols**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IProjectSettingsDlg::VBasic_DocCommentsAsDocumentation**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IProjectSettingsDlg::VBasic_ResolveAliases**

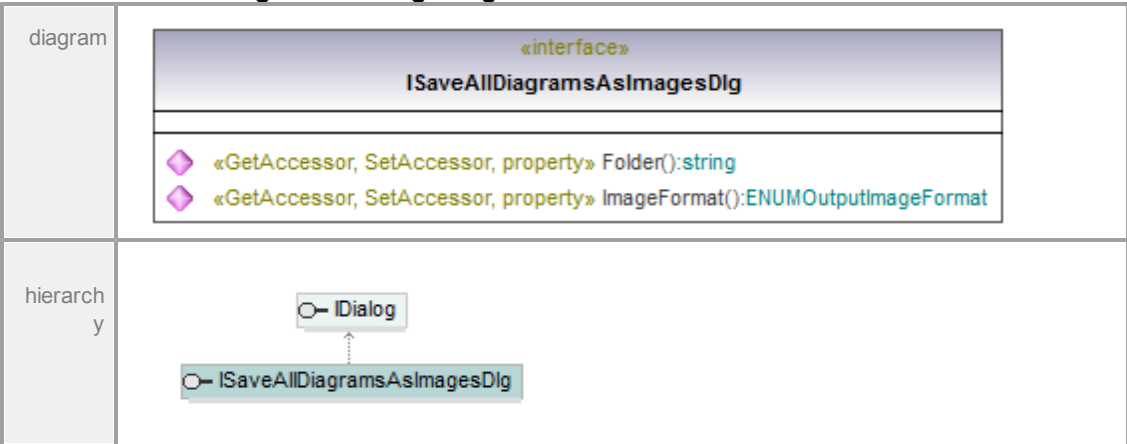
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IProjectSettingsDlg::VBasic_WriteDocumentationAsDocComments**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

UModelAPI - ISaveAllDiagramsAsImagesDlg

Interface **ISaveAllDiagramsAsImagesDlg**



typedElements	Interface IDialogs Interface IDocument	Operation SaveAllDiagramsAsImagesDlg Operation SaveAllDiagramsAsImages

Operation **ISaveAllDiagramsAsImagesDlg::Folder**

parameter	name return	direction return	type string	type modifier	multiplicity	default

Operation **ISaveAllDiagramsAsImagesDlg::ImageFormat**

parameter	name return	direction return	type ENUMOutputImageFormat	type modifier	multiplicity	default

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ISynchronizationSettingsDlgInterface **ISynchronizationSettingsDlg**

diagram	«interface» I SynchronizationSettingsDlg «GetAccessor, SetAccessor, property» ModelFromCode_Synchronization():ENUMSynchronizationKind «GetAccessor, SetAccessor, property» CodeFromModel_Synchronization():ENUMSynchronizationKind «GetAccessor, SetAccessor, property» CodeFromModel_UserDefinedSPLTemplatesOverrideDefault():bool «GetAccessor, SetAccessor, property» CodeFromModel_DeletingCode():ENUMSynchronizationDeleteKind	
hierarchy	— IDialog — ISynchronizationSettingsDlg	
typedElements	Interface IDialogs Interface IDocument	Operation SynchronizationSettingsDlg Operation SynchronizationSettings SynchronizeCodeFromModel SynchronizeModelFromCode

Operation **ISynchronizationSettingsDlg::CodeFromModel_DeletingCode**

parameter	name return	direction return	type ENUMSynchronizationDeleteKind	type modifier	multiplicity	default

Operation **ISynchronizationSettingsDlg::CodeFromModel_Synchronization**

parameter	name return	direction return	type ENUMSynchronizationKind	type modifier	multiplicity	default

Operation

ISynchronizationSettingsDlg::CodeFromModel_UserDefinedSPLTemplatesOverrideDefault

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation ISynchronizationSettingsDlg::ModelFromCode_Synchronization

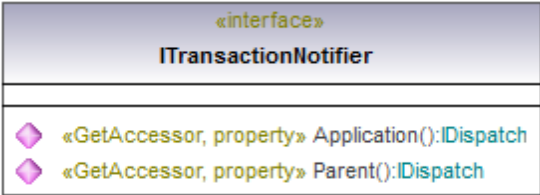
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMSynchronizationKind			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ITransactionNotifier

Interface ITransactionNotifier

diagram					
typedElements	Interface IDocument Operation TransactionNotifier				
documentation	Use this interface to register for ITransactionEvents .				

Operation ITransactionNotifier::Application

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation ITransactionNotifier::Parent

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IURLDIg

Interface IURLDIg

diagram	«interface» IURLDIg ◆ Delete(in strURL:string):void ◆ NewFolder(in strURL:string):void ◆ «GetAccessor, SetAccessor, property» URL():string ◆ «GetAccessor, SetAccessor, property» UserName():string ◆ «GetAccessor, SetAccessor, property» Password():string ◆ «GetAccessor, SetAccessor, property» NoCache():bool	
hierarch y	<pre>graph BT; IDialog -- > IURLDIg</pre>	
typedEle ments	Interface IApplication Interface IDialogs Interface IDocument Interface IExportXMIFileDIg Interface IIncludeSubprojectDIg	Operation ImportFromXMIFileFromURL Operation OpenDocumentFromURL Operation URLOpenDialog Operation URLSaveDialog Operation MergeProjectFromURL Operation SaveToURL Operation URLDIg Operation URLDIg

Operation IURLDIg::Delete

paramete r	name strURL return	direction in return	type string void	type modifier	multiplicity	default

Operation IURLDIg::NewFolder

paramete r	name strURL return	direction in return	type string void	type modifier	multiplicity	default

Operation IURLDIg::NoCache

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation IURLDIg::Password

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation IURLDIg::URL

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IURLDig::UserName**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

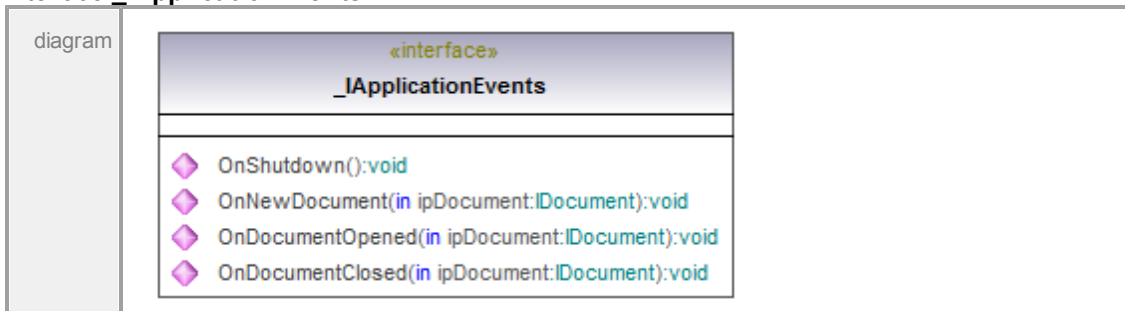
08/03/15 13:42:06

Events

This is a list of all events sent by the UModel API.

A list of events sent on UMLData level can be found [here](#).

UModelAPI - _ApplicationEvents

Interface **_ApplicationEvents**Operation **_ApplicationEvents::OnDocumentClosed**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipDocument return	in return	<u>IDocument</u> void			

Operation **_ApplicationEvents::OnDocumentOpened**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipDocument return	in return	<u>IDocument</u> void			

Operation **_ApplicationEvents::OnNewDocument**

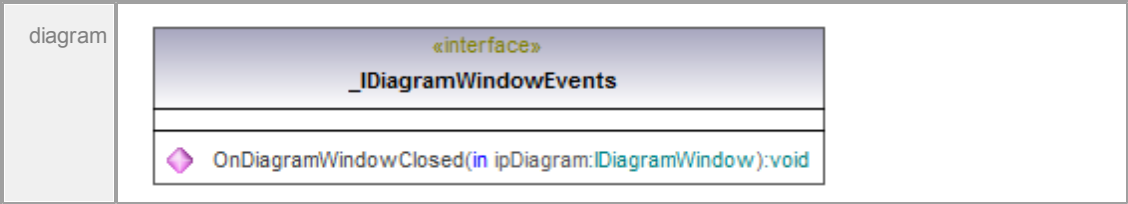
parameter	name	direction	type	type modifier	multiplicity	default
r	ipDocument return	in return	<u>IDocument</u> void			

Operation **_ApplicationEvents::OnShutdown**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

UModelAPI - _IDiagramWindowEvents

Interface **_IDiagramWindowEvents**

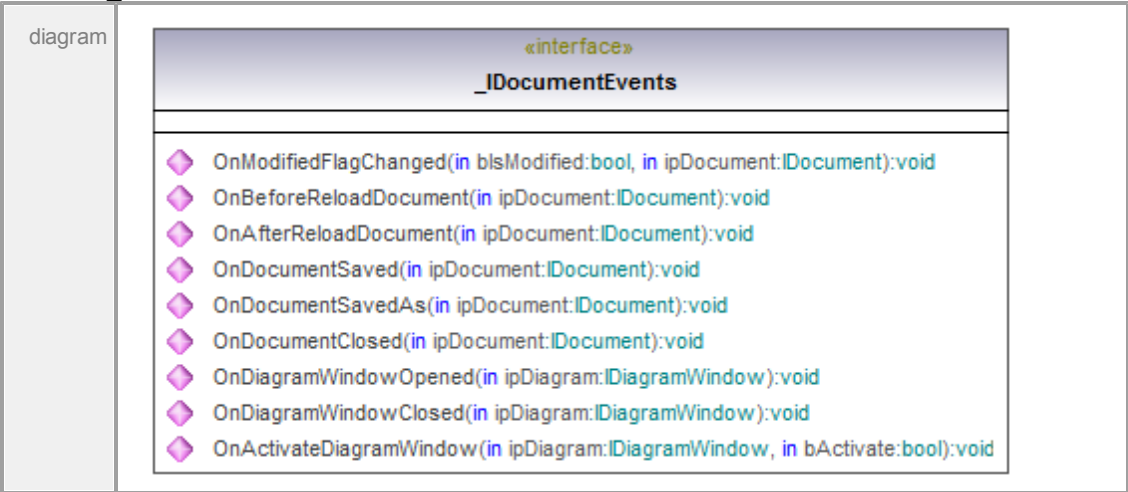


Operation **_IDiagramWindowEvents::OnDiagramWindowClosed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	in	IDiagramWindow			
	return	return	void			

UModelAPI - _IDocumentEvents

Interface **_IDocumentEvents**



Operation **_IDocumentEvents::OnActivateDiagramWindow**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	in	IDiagramWindow			
	bActivate	in	bool			
	return	return	void			

Operation **_IDocumentEvents::OnAfterReloadDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	in	IDocument			

	return	return	void

Operation **IDocumentEvents::OnBeforeReloadDocument**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	in	IDocument			
	return	return	void			

Operation **IDocumentEvents::OnDiagramWindowClosed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	in	IDiagramWindow			
	return	return	void			

Operation **IDocumentEvents::OnDiagramWindowOpened**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDiagram	in	IDiagramWindow			
	return	return	void			

Operation **IDocumentEvents::OnDocumentClosed**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	in	IDocument			
	return	return	void			

Operation **IDocumentEvents::OnDocumentSaved**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	in	IDocument			
	return	return	void			

Operation **IDocumentEvents::OnDocumentSavedAs**

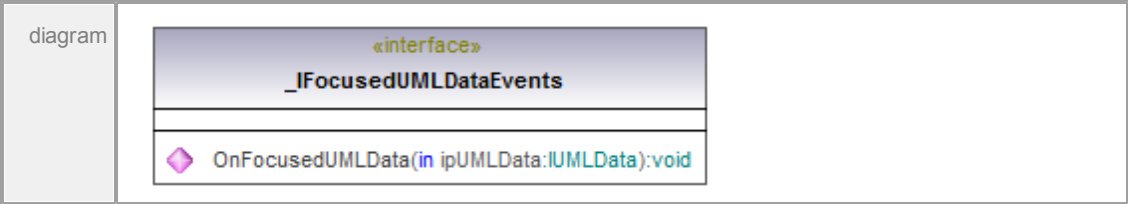
parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	in	IDocument			
	return	return	void			

Operation **IDocumentEvents::OnModifiedFlagChanged**

parameter	name	direction	type	type modifier	multiplicity	default
	isModified	in	bool			
	ipDocument	in	IDocument			
	return	return	void			

UModelAPI - _IFocusedUMLDataEvents

Interface **_IFocusedUMLDataEvents**



Operation **_IFocusedUMLDataEvents::OnFocusedUMLData**

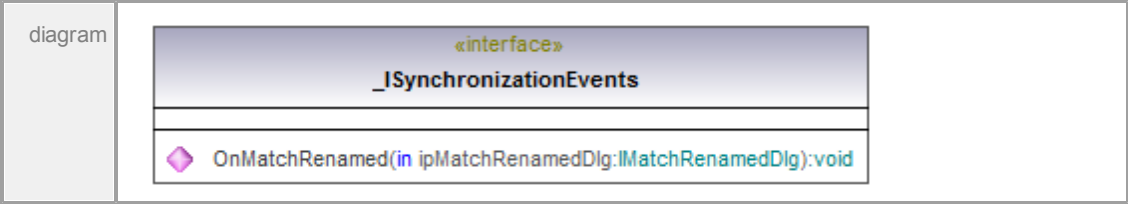
parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	in	IUMLData			
	return	return	void			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - _ISynchronizationEvents

Interface **_ISynchronizationEvents**



Operation **_ISynchronizationEvents::OnMatchRenamed**

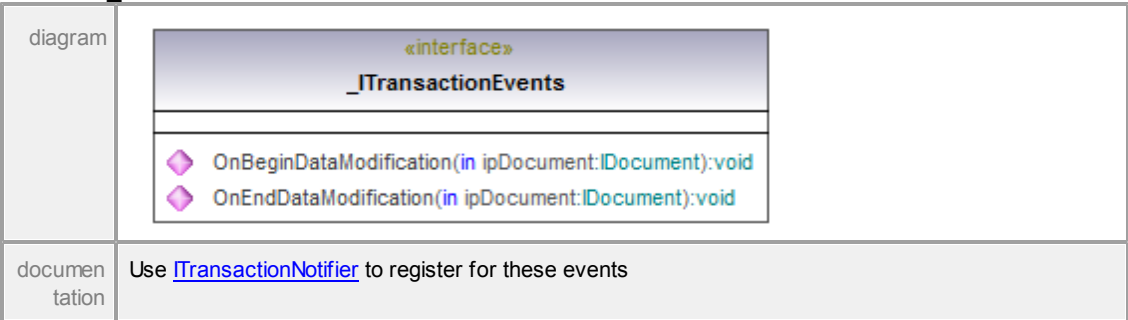
parameter	name	direction	type	type modifier	multiplicity	default
	ipMatchRenamedDlg	in	IMatchRenamedDlg			
	return	return	void			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - _ITransactionEvents

Interface **_ITransactionEvents**



Operation **_ITransactionEvents::OnBeginDataModification**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	in	IDocument			
	return	return	void			

Operation **_ITransactionEvents::OnEndDataModification**

parameter	name	direction	type	type modifier	multiplicity	default
	ipDocument	in	IDocument			
	return	return	void			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

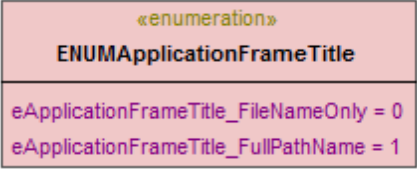
Enumerations

This is a list of all enumerations used by the UModel API. If your scripting environment does not support enumerations use the number-values instead.

A list of enumerations defined on UMLData level can be found [here](#).

UModelAPI - ENUMApplicationFrameTitle

Enumeration **ENUMApplicationFrameTitle**

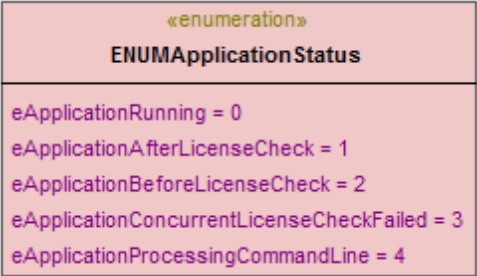
diagram					
typedElements	Interface ILocalOptionsView		Operation FrameTitle		

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMApplicationStatus

Enumeration **ENUMApplicationStatus**

diagram					
typedElements	Interface IApplication		Operation Status		

UModelAPI - ENUMAutolayoutGrowDirectionKind

Enumeration **ENUMAutolayoutGrowDirectionKind**

diagram	«enumeration» ENUMAutolayoutGrowDirectionKind eAutolayoutGrowDirection_TopDown = 0 eAutolayoutGrowDirection_BottomUp = 1 eAutolayoutGrowDirection_LeftToRight = 2 eAutolayoutGrowDirection_RightToLeft = 3	
typedElements	Interface ILocalOptionsView	Operation AutolayoutHierarchic_GrowDirection

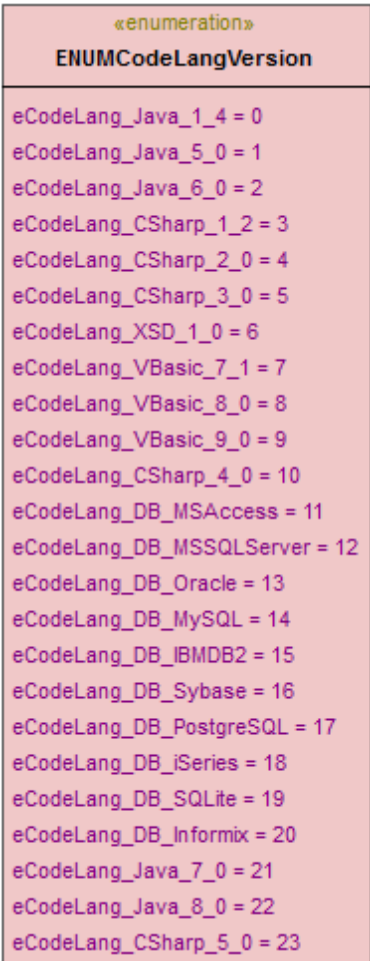
UModelAPI - ENUMCodeLang

Enumeration **ENUMCodeLang**

diagram	«enumeration» ENUMCodeLang eCodeLang_UML = -1 eCodeLang_Java = 0 eCodeLang_CSharp = 1 eCodeLang_XSD = 2 eCodeLang_VBasic = 3 eCodeLang_DB = 4	
typedElements	Interface IModelTransformationDlg Interface IUMLComponent Interface IUMLDataAll Interface IUMLPackage	Operation TransformFromLanguage Operation TransformToLanguage Operation CodeLang Operation CodeLang Operation IsCodeLangNamespace Operation IsCodeLangNamespaceRoot Operation IsCodeLangNamespace Operation IsCodeLangNamespaceRoot

UModelAPI - ENUMCodeLangVersion

Enumeration **ENUMCodeLangVersion**

diagram		
typedElements	Interface IImportSourceDlg Interface ILocalOptionsEditing Interface IUMLComponent Interface IUMLDataAll	Operation Language Operation ComponentsDefaultCodeLangVersion Operation CodeLangVersion Operation CodeLangVersion

UModelAPI - ENUMDiagramLayoutKind

Enumeration **ENUMDiagramLayoutKind**

diagram	«enumeration» ENUMDiagramLayoutKind eDiagramLayout_None = -1 eDiagramLayout_ForceDirected = 0 eDiagramLayout_Hierarchic = 1 eDiagramLayout_Block = 2 eDiagramLayout_XSDGlobals = 3	
typedElements	Interface IDiagramWindow Interface IImportSourceDlg	Operation Autolayout AutolayoutSelection Operation Content_Autolayout PackageDependency_Autolayout

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMDocumentationFilePathKind

Enumeration **ENUMDocumentationFilePathKind**

diagram	«enumeration» ENUMDocumentationFilePathKind eDocumentationFilePath_Absolute = 0 eDocumentationFilePath_RelativeToResultFile = 1	
typedElements	Interface IGenerateDocumentationDlg	Operation GenerateLinksToLocalFiles

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMDocumentationFontSetting

Enumeration **ENUMDocumentationFontSetting**

diagram	«enumeration» ENUMDocumentationFontSetting eDocumentationFontSetting_Header = 0 eDocumentationFontSetting_Header2 = 1 eDocumentationFontSetting_ElementHeader = 2 eDocumentationFontSetting_ElementHeader2 = 3 eDocumentationFontSetting_TableLineTitle = 4 eDocumentationFontSetting_TableLineData = 5 eDocumentationFontSetting_SubTableLineTitle = 6 eDocumentationFontSetting_SubTableLineData = 7 eDocumentationFontSetting_Footer = 8 eDocumentationFontSetting_Footer2 = 9
typedElements	Interface IGenerateDocumentationDlg Operation Fonts_GetFace Fonts_GetSize Fonts_GetTextColor Fonts_IsBold Fonts_IsItalic Fonts_IsUnderline Fonts_SetBold Fonts_SetFace Fonts_SetItalic Fonts_SetSize Fonts_SetTextColor Fonts_SetUnderline

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMDocumentationImageFormat

Enumeration **ENUMDocumentationImageFormat**

diagram	«enumeration» ENUMDocumentationImageFormat eDocumentationImageFormat_PNG = 0 eDocumentationImageFormat_EMF = 1

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMDocumentationOutputFormat

Enumeration **ENUMDocumentationOutputFormat**

diagram	«enumeration» ENUMDocumentationOutputFormat eDocumentationOutputFormat_HTML = 0 eDocumentationOutputFormat_Word = 1 eDocumentationOutputFormat_RTF = 2 eDocumentationOutputFormat_PDF = 3	
typedElements	Interface IGenerateDocumentationDlg	Operation OutputFormat

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMExportXMType

Enumeration **ENUMExportXMType**

diagram	«enumeration» ENUMExportXMType eXMI21ForUML20 = 0 eXMI21ForUML212 = 1 eXMI21ForUML22 = 2 eXMI21ForUML23 = 3 eXMI24ForUML24 = 4	
typedElements	Interface IExportXMIFileDlg	Operation XMType

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMOpenMessageWindow

Enumeration **ENUMOpenMessageWindow**

diagram	«enumeration» ENUMOpenMessageWindow eOpenMessageWindow_Always = 0 eOpenMessageWindow_ForErrorsAndWarnings = 1 eOpenMessageWindow_ForErrors = 2	
typedElements	Interface ILocalOptionsCodeEnginein g	Operation OpenMessageWindow

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

09/04/12 14:02:42

UModelAPI - ENUMOutputImageFormat

Enumeration **ENUMOutputImageFormat**

diagram	«enumeration» ENUMOutputImageFormat eOutputImageFormat_PNG = 0 eOutputImageFormat_EMF = 1	
typedElements	Interface IDiagramWindow Interface IGenerateDocumentationDlg Interface ISaveAllDiagramsAsImagesDlg	Operation SaveDiagramAsImage Operation DiagramImageFormat Operation ImageFormat

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMSynchronizationDeleteKind

Enumeration **ENUMSynchronizationDeleteKind**

diagram	«enumeration» ENUMSynchronizationDeleteKind eSynchronizationDelete_CommentOut = 0 eSynchronizationDelete_Delete = 1	
typedElements	Interface ISynchronizationSettingsDlg	Operation CodeFromModel_DeletingCode

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMSynchronizationKind

Enumeration **ENUMSynchronizationKind**

diagram	«enumeration» ENUMSynchronizationKind eSynchronization_Merge = 0 eSynchronization_Overwrite = 1	
typedElements	Interface IImportSourceDlg Interface IModelTransformationDlg	Operation Synchronization Operation Synchronization Operation CodeFromModel_Synchronization

	ISynchronizationSettingsDlg	tion ModelFromCode_Synchroniza tion

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - ENUMSyntaxCheckKind

Enumeration **ENUMSyntaxCheckKind**

diagram	«enumeration» ENUMSyntaxCheckKind eSyntaxCheck_AllCodingElements = 0 eSyntaxCheck_ElementsUsedForCodeEngineering = 1
typedElements	Interface ILocalOptionsCodeEngineering Operation SyntaxCheck

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UMLData

The UMLData interfaces allow direct UML-level access to a document. You can read and directly modify the UML representation of the document.

[IUMMLData](#) is the common base interface of [IUMMLElement](#) and [IUMMLGuiElement](#).

[IUMMLElements](#) contains elements as defined by OMG in the UML 2.4 Superstructure Specification (also see <http://www.uml.org>)

[IUMMLGuiElements](#) contains elements from Altova for diagrams and to show [IUMMLElements](#) on diagrams

Also see [Object model UMLData](#) how to read and modify UML elements and GUI elements.

Errors

- 1000 The application object is no longer valid.
- 1001 Invalid parameter or invalid address for the return parameter was specified.
- 1002 UModel API is not available in the current edition.
- 1400 Invalid UMLData modification.
- 1401 Invalid Waypoint modification.
- 1402 No changes allowed.
- 1403 No changes allowed during Undo/Redo.
- 1404 Element is hidden by Element Style (visibility).
- 1405 Predefined element not found.
- 1406 Predefined element is of invalid kind.

Errors of the UModel API can be found here.

UModelAPI - IUMLData

Interface **IUMLData**

diagram		
hierarch y		
typedEle ments	Interface _IFocusedUMLDataEvents Interface IUMLDataEvents Interface IDiagramWindow Interface IDocument Interface IUMLCommentTextHyperlink Interface IUMLData Interface IUMLDataAll Interface IUMLDataList Interface IUMLElement Interface IUMLGuiDiagram Interface IUMLGuiRootElement Interface IUMLGuiTextHyperlink Interface IUMLHyperlink2Model	Operation OnFocusedUMLData Operation OnAfterAddChild Operation OnBeforeErase Operation OnChanged Operation OnMoveData Operation FocusedData Operation CanFocusUMLDataInModelTree Operation FocusUMLDataInModelTree Operation SetHyperlinkModelElementAddress Operation IsSameUMLData Operation AddUMLGuiNodeLink Operation FindPredefinedOwnedElement Operation InsertOwnedDiagramAt Operation InsertOwnedHyperlink2ModelAt Operation IsSameUMLDataLinkedModelElement Operation SetHyperlinkModelElementAddress Operation ContainsUMLDataItem Operation FindPredefinedOwnedElement Operation AddUMLGuiNodeLink Operation InsertOwnedDiagramAt Operation SetHyperlinkModelElementAddress Operation LinkedModelElement

	Interface IUMLNamedElement	Operation InsertOwnedHyperlink2ModelAt

Operation **IUMLData::Application**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **IUMLData::EventFilter**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLData::IsEditable**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLData::IsKindOf**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string bool			

Operation **IUMLData::IsSameUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipUMLDataToC in ompare return	return	IUMLData bool			

Operation **IUMLData::KindName**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLData::Parent**

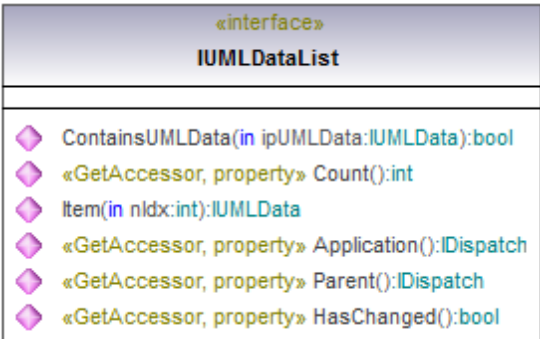
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **IUMLData::UUID**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

UModelAPI - IUMLDataList

Interface **IUMLDataList**

diagram	 <pre> «interface» IUMLDataList ContainsUMLData(in ipUMLData:IUMLData):bool «GetAccessor, property» Count():int Item(in nIdx:int):IUMLData «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch «GetAccessor, property» HasChanged():bool </pre>	
typedElements	Interface IDiagramWindow Interface IUMLAcceptEventAction Interface IUMLAction Interface IUMLActivity Interface IUMLActivityEdge Interface IUMLActivityGroup Interface IUMLActivityNode Interface IUMLActivityPartition Interface IUMLArtifact Interface IUMLAssociation Interface IUMLBehavior Interface IUMLBehavioralFeature Interface IUMLBehavioredClassifier Interface IUMLCallAction Interface IUMLClass Interface IUMLClassifier Interface IUMLClassifierTemplateParameter Interface IUMLCollaboration Interface IUMLCollaborationUse	Operation SelectedGuiElements Operation ActionTriggers Operation EventActionResult Operation InputPins Operation LocalPostConditions Operation LocalPreConditions Operation OutputPins Operation ActivityEdges ActivityGroups Operation ActivityNodes Operation ActivityPartitions Operation InterruptibleActivityRegions Operation StructuredActivityNodes Operation ContainedEdges Operation ContainedNodes SubGroups Operation IncomingEdges Operation OutgoingEdges Operation Edges Nodes SubPartitions Operation Manifestations Operation NestedArtifacts Operation OwnedAttributes Operation OwnedOperations Operation EndTypes MemberEnds Operation NavigableOwnedEnds Operation OwnedEnds Operation OwnedParameters Operation Postconditions Preconditions Operation Methods OwnedParameters Operation RaiseExceptions Operation InterfaceRealizations Operation OwnedBehaviors Operation Results Operation NestedClassifiers Operation OwnedOperations Operation SuperClasses Operation Attributes CollaborationUses Operation Features Generalizations Operation Generals InheritedMembers Operation OwnedUseCases Specifics Operation UseCases Operation ConstrainingClassifiers Operation CollaborationRoles Operation RoleBindings

Interface IUMLCreatedFragment	Operation Operands
Interface IUMLComment	Operation AnnotatedElements
Interface IUMLComponent	Operation OwnedHyperlinks
Interface IUMLConnectionPointReference	Operation Realizations
Interface IUMLConstraint	Operation Entries Exits
Interface IUMLDataAll	Operation ConstrainedElements
	Operation ActionTriggers ActivityEdges
	ActivityGroups ActivityNodes
	ActivityPartitions ActualGates
	AllApplicableStereotypes
	AllWaypoints
	AnnotatedElements
	AppliedStereotypes
	Arguments AttachedNodes
	Attributes
	ClientDependencies Clients
	CollaborationRoles
	CollaborationUses
	ConnectionPoints
	Connections
	ConstrainedElements
	ConstrainingClassifiers
	ContainedEdges
	ContainedNodes Conveyed
	DeployedElements
	Deployments Edges
	ElementImports EndTypes
	Entries EventActionResults
	ExceptionHandler
	ExceptionTypes Exits Extends
	ExtensionLocations
	ExtensionPoints Features
	FeaturingClassifiers
	FormalGates Fragments
	Generalizations Generals
	GetOwnedElementsOfKind
	GuiLinks Handlers
	ImportedMembers Includes
	IncomingEdges Incomings
	InformationFlowRealizations
	InformationSources
	InformationTargets
	InheritedMembers
	InputElements InputPins
	InputValues InStates
	InterfaceRealizations
	InterruptibleActivityRegions
	InterruptingEdges Layers
	Lifelines
	LineConnectionWaypoints
	LineLinks
	LocalPostConditions
	LocalPreConditions
	LowerValues Manifestations
	MemberEnds Members
	Messages Methods
	NavigableOwnedEnds

		NestedArtifacts NestedClassifiers NestedNodes NestedPackages Nodes Observations Operands OutgoingEdges Outgoings OutputElements OutputPins OutputValues OwnedArguments OwnedAttributes OwnedBehaviors OwnedComments OwnedConnectors OwnedDiagrams OwnedElements OwnedEnds OwnedGuiElements OwnedGuiNodeLinks OwnedHyperlinks OwnedLiterals OwnedMembers OwnedOperations OwnedParameters OwnedPorts OwnedRules OwnedStereotypes OwnedTemplateBindings OwnedTemplateParameters OwnedTypes OwnedUseCases PackagedElements PackageImports PackageMerges ParameterSubstitutions Postconditions Preconditions ProfileApplications Qualifiers RaisedExceptions Realizations RealizingConnectors Referred Regions RelatedElements RelativeNodes Results RoleBindings Slots Sources Specifics StereotypeApplications StructuredActivityNodes SubGroups Subjects SubmachineStates SubPartitions SubVertices SuperClasses SupplierDependencies Suppliers Targets TextLabels Transitions Triggers TypedElements UpperValues UseCases Values Waypoints
	Interface IUMLDatatype	Operation OwnedAttributes
	Interface IUMLDependency	Operation OwnedOperations
	Interface IUMLDeploymentTarget	Operation Clients Suppliers
	Interface IUMLDirectedRelationship	Operation DeployedElements
		Operation Deployments
		Operation Sources Targets

Interface IUMLDuration	Operation Observations
Interface IUMLElement	Operation AllApplicableStereotypes
	Operation AppliedStereotypes
	Operation GetOwnedElementsOfKind
	Operation OwnedComments
	Operation OwnedElements
	Operation StereotypeApplications
Interface IUMLEncapsulatedClassifier	Operation OwnedPorts
Interface IUMLEnumeration	Operation OwnedLiterals
Interface IUMLExceptionHandler	Operation ExceptionTypes
Interface IUMLExecutableNode	Operation Handlers
Interface IUMLExpansionRegion	Operation InputElements
Interface IUMLExpression	Operation OutputElements
Interface IUMLExtend	Operation Operands
Interface IUMLFeature	Operation ExtensionLocations
Interface IUMLGuiDiagram	Operation FeaturingClassifiers
	Operation GuiLinks
	Operation Layers
	Operation OwnedHyperlinks
Interface IUMLGuiElement	Operation OwnedGuiElements
Interface IUMLGuiLineLink	Operation AllWaypoints
	Operation LineConnectionWaypoints
	Operation Waypoints
Interface IUMLGuiLink	Operation AttachedNodes
Interface IUMLGuiNodeLink	Operation RelativeNodes
Interface IUMLGuiNote	Operation OwnedGuiNodeLinks
Interface IUMLGuiRootElement	Operation OwnedHyperlinks
	Operation OwnedDiagrams
	Operation TextLabels
Interface IUMLGuiTextLabelWaypoint	Operation LineLinks
Interface IUMLGuiWaypoint	Operation Conveyed
Interface IUMLInformationFlow	Operation InformationFlowRealizations
	Operation InformationSources
	Operation InformationTargets
	Operation RealizingConnectors
	Operation Slots
Interface IUMLInstanceSpecification	Operation FormalGates
Interface IUMLInteraction	Operation Fragments
	Operation Lifelines
Interface IUMLInteractionUse	Operation Messages
Interface IUMLInterface	Operation ActualGates
	Operation NestedClassifiers
	Operation OwnedAttributes
	Operation OwnedOperations
Interface IUMLInterruptibleActivityRegion	Operation InterruptingEdges
on	Operation Nodes
Interface IUMLInvocationAction	Operation Arguments
	Operation OwnedArguments
Interface IUMLMessage	Operation LowerValues
Interface IUMLMultiplicityElement	Operation UpperValues
Interface IUMLNamedElement	Operation ClientDependencies
	Operation OwnedHyperlinks
	Operation SupplierDependencies
	Operation ElementImports
	Operation ImportedMembers
	Operation Members
	Operation OwnedMembers
	Operation OwnedRules
	Operation PackageImports
	Operation PackageMerges
	Operation NestedNodes
Interface IUMLNode	Operation ExceptionHandlers
Interface IUMLObjectNode	Operation InStates

Interface IUMLOpaqueAction	Operation InputValues OutputValues
Interface IUMLPackage	Operation NestedPackages
	OwnedStereotypes
	OwnedTypes
	PackagedElements
	ProfileApplications
Interface IUMLProperty	Operation Qualifiers
Interface IUMLProtocolTransition	Operation Referred
Interface IUMLRegion	Operation SubVertices Transitions
Interface IUMLRelationship	Operation RelatedElements
Interface IUMLSignal	Operation OwnedAttributes
Interface IUMLSlot	Operation Values
Interface IUMLState	Operation ConnectionPoints
	Connections Regions
Interface IUMLStateMachine	Operation ConnectionPoints Regions
	SubmachineStates
Interface IUMLStructuredActivityNode	Operation Edges Nodes
Interface IUMLStructuredClassifier	Operation OwnedAttributes
Interface IUMLTemplateableElement	Operation OwnedConnectors
Interface IUMLTemplateBinding	Operation OwnedTemplateBindings
Interface IUMLTemplateSignature	Operation ParameterSubstitutions
Interface IUMLTimeExpression	Operation OwnedTemplateParameters
Interface IUMLTransition	Operation Observations
Interface IUMLType	Operation Triggers
Interface IUMLUseCase	Operation TypedElements
	Operation Extends ExtensionPoints
	Includes Subjects
Interface IUMLVertex	Operation Incomings Outgoings

Operation **IUMLDataList::Application**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation **IUMLDataList::ContainsUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLData	in	IUMLData			
	return	return	bool			

Operation **IUMLDataList::Count**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLDataList::HasChanged**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLDataList::Item**

parameter	name	direction	type	type modifier	multiplicity	default

r	nidx return	in return	int IUMLData

Operation **IUMLDataList::Parent**

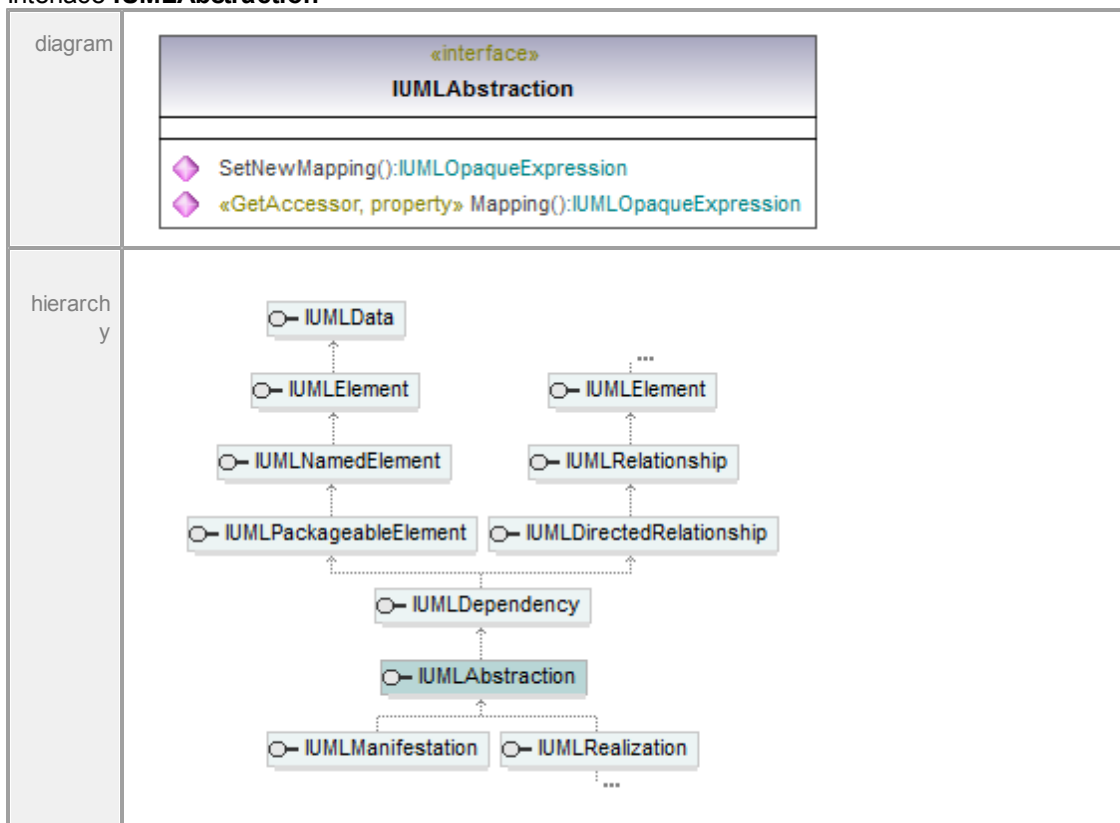
paramete r	name return	direction return	type IDispatch	type modifier	multiplicity	default

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

IUMLElement

This is a list of elements as defined by OMG in the UML 2.4 Superstructure Specification (also see <http://www.uml.org>).

UModelAPI - IUMLAbstraction**Interface IUMLAbstraction**Operation **IUMLAbstraction::Mapping**

paramete r	name return	direction return	type IUMLOpaqueExpression	type modifier	multiplicity	default

Operation **IUMLAbstraction::SetNewMapping**

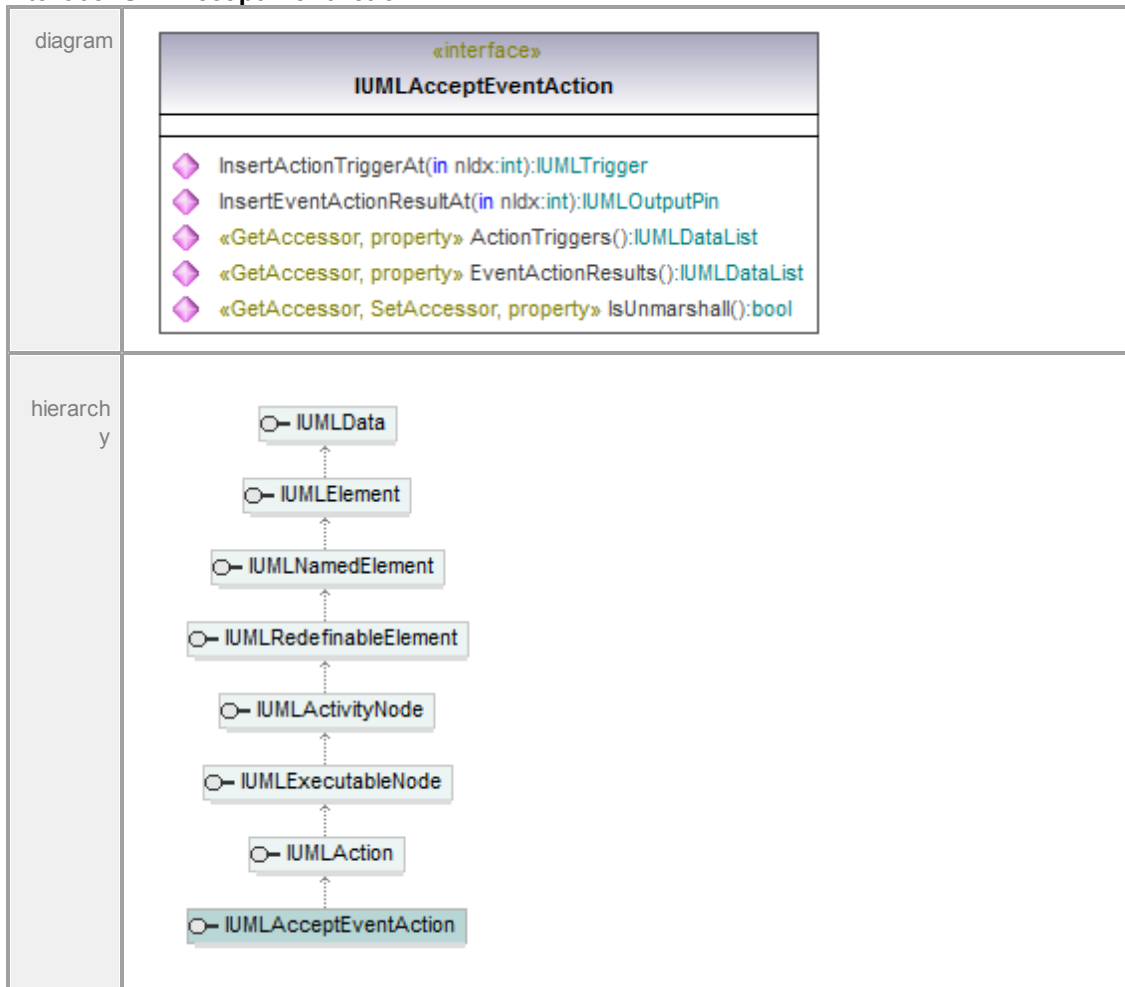
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLOpaqueExpression			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLAcceptEventAction

Interface **IUMLAcceptEventAction**



Operation **IUMLAcceptEventAction::ActionTriggers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLTrigger .					

Operation **IUMLAcceptEventAction::EventActionResults**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

document ation	A list of elements of type IUMLOutputPin .

Operation **IUMLAcceptEventAction::InsertActionTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	return	return	IUMLTrigger			

Operation **IUMLAcceptEventAction::InsertEventActionResultAt**

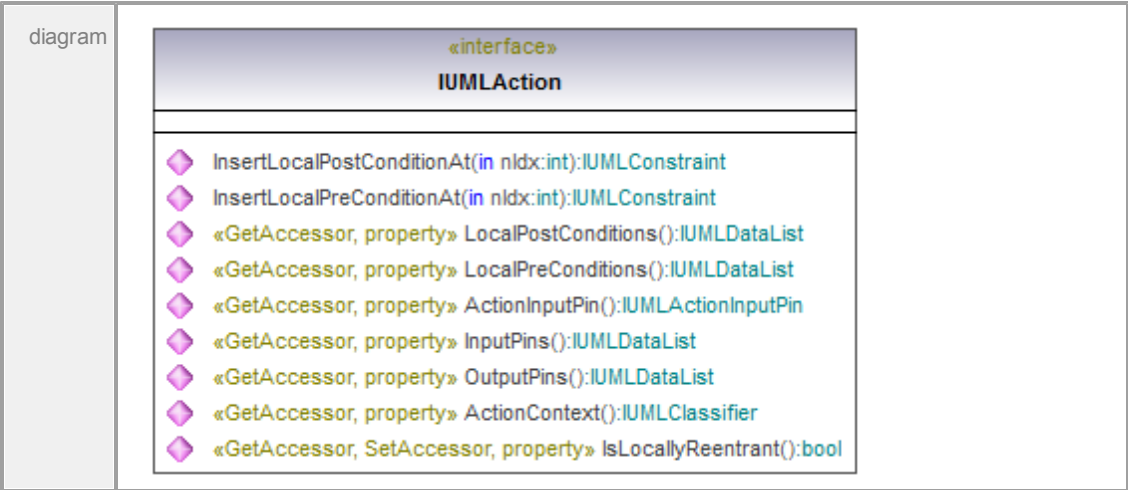
parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	return	return	IUMLOutputPin			

Operation **IUMLAcceptEventAction::IsUnmarshall**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

UModelAPI - IUMLAction

Interface **IUMLAction**



document ation	A list of elements of type IUMLConstraint .

Operation **IUMLAction::LocalPreConditions**

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
document ation	A list of elements of type IUMLConstraint .					

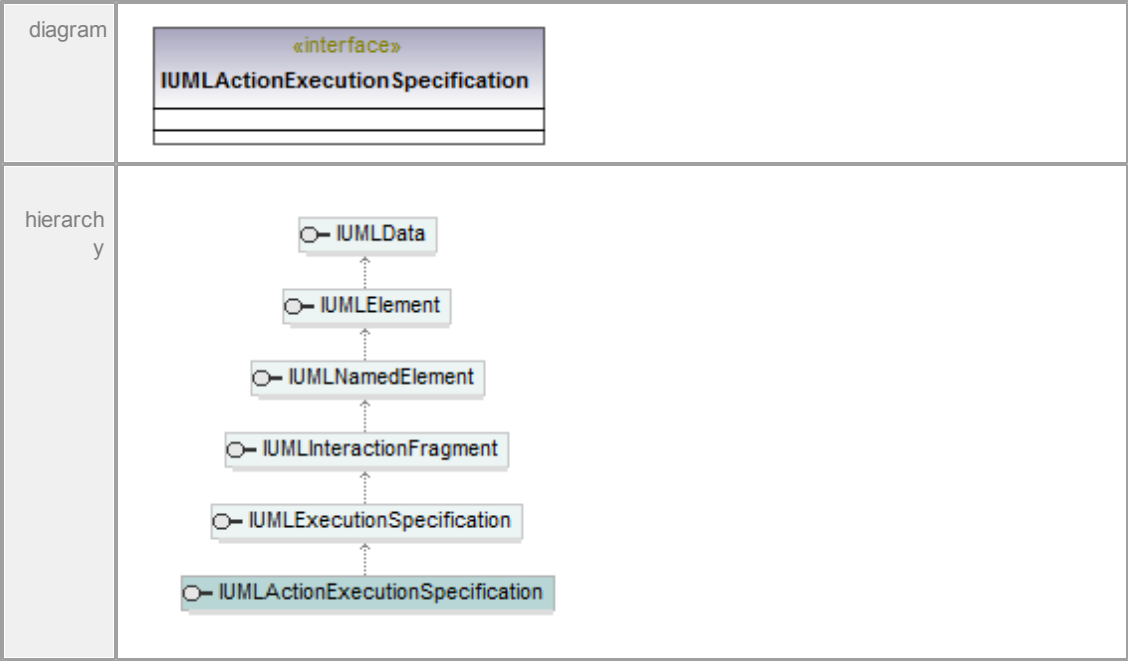
Operation **IUMLAction::OutputPins**

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
document ation	A list of elements of type IUMLOutputPin .					

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLActionExecutionSpecification

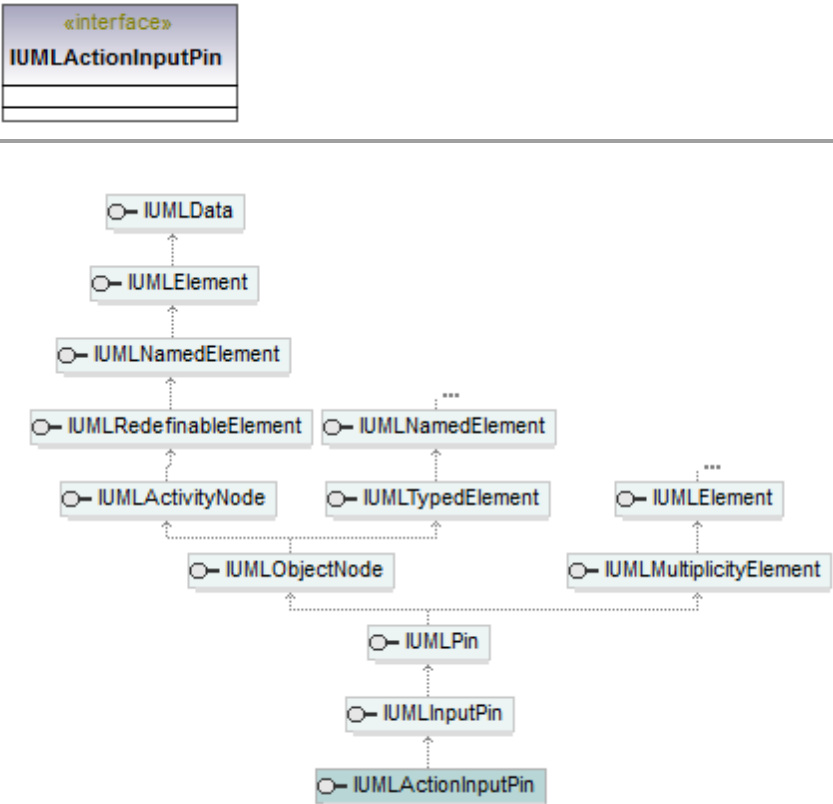
Interface **IUMLActionExecutionSpecification**



UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLActionInputPin

Interface **IUMLActionInputPin**

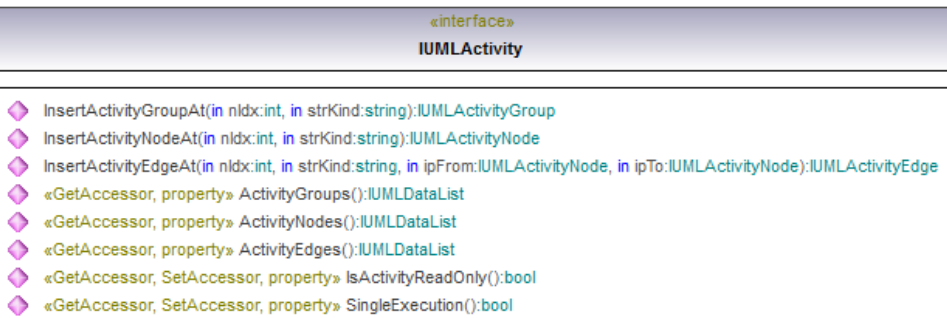
diagram	 <pre> classDiagram class IUMLData class IUMLElement class IUMLNamedElement class IUMLRedefinableElement class IUMLActivityNode class IUMLObjectNode class IUMLTypedElement class IUMLPin class IUMLInputPin class IUMLActionInputPin class IUMLMultiplicityElement IUMLData < .. IUMLElement IUMLElement < .. IUMLNamedElement IUMLNamedElement < .. IUMLRedefinableElement IUMLRedefinableElement < .. IUMLActivityNode IUMLActivityNode < .. IUMLObjectNode IUMLObjectNode < .. IUMLTypedElement IUMLTypedElement < .. IUMLPin IUMLPin < .. IUMLInputPin IUMLInputPin < .. IUMLActionInputPin IUMLObjectNode < .. IUMLMultiplicityElement IUMLTypedElement < .. IUMLMultiplicityElement IUMLMultiplicityElement < .. IUMLElement </pre>	
typedElements	Interface IUMLAction Interface IUMLDataAll	Operation ActionInputPin Operation ActionInputPin

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLActivity

Interface **IUMLActivity**

diagram	 <pre> classDiagram class IUMLActivity { InsertActivityGroupAt(in nldx:int, in strKind:string):IUMLActivityGroup InsertActivityNodeAt(in nldx:int, in strKind:string):IUMLActivityNode InsertActivityEdgeAt(in nldx:int, in strKind:string, in ipFrom:IUMLActivityNode, in ipTo:IUMLActivityNode):IUMLActivityEdge «GetAccessor, property» ActivityGroups():IUMLDataList «GetAccessor, property» ActivityNodes():IUMLDataList «GetAccessor, property» ActivityEdges():IUMLDataList «GetAccessor, SetAccessor, property» IsActivityReadOnly():bool «GetAccessor, SetAccessor, property» SingleExecution():bool } </pre>

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	strKind	in	string			
	return	return	IUMLActivityGroup			

Operation **IUMLActivity::InsertActivityNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	strKind	in	string			
	return	return	IUMLActivityNode			

Operation **IUMLActivity::IsActivityReadOnly**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLActivity::SingleExecution**

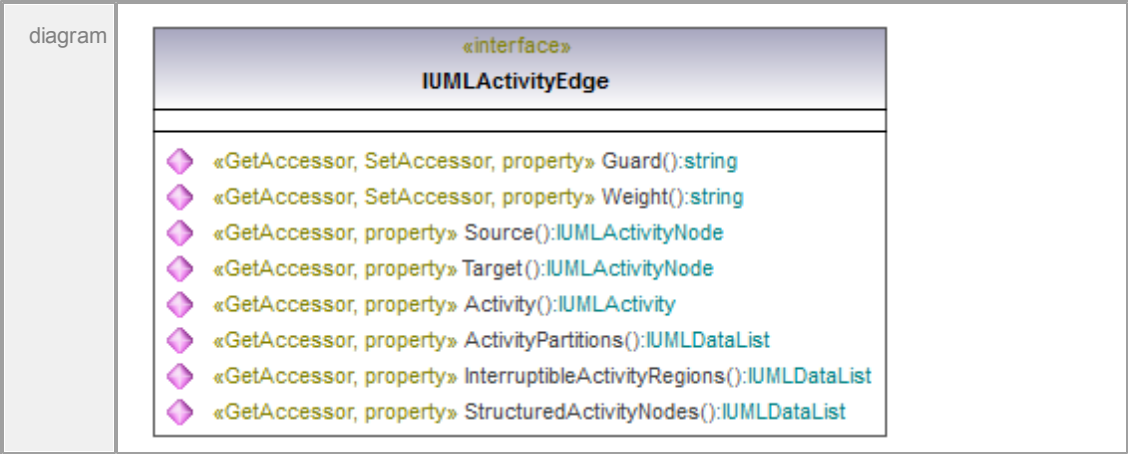
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLActivityEdge

Interface **IUMLActivityEdge**



hierarchy	<pre>graph BT IUMLElement < -- IUMLData IUMLElement < -- IUMLNamedElement IUMLElement < -- IUMLRedefinableElement IUMLElement < -- IUMLActivityEdge IUMLActivityEdge < -- IUMLControlFlow IUMLActivityEdge < -- IUMLObjectFlow</pre>		
typedElements	Interface IUMLElement Interface IUMLNamedElement Interface IUMLRedefinableElement Interface IUMLActivityEdge Interface IUMLControlFlow Interface IUMLObjectFlow	Operation InsertActivityEdgeAt Operation InsertEdgeAt Operation InsertActivityEdgeAt InsertEdgeAt InsertInterruptingEdgeAt InsertInterruptingEdgeAt Operation InsertEdgeAt	

Operation **IUMLActivityEdge::Activity**

paramete r	name return	direction return	type IUMLElement	type modifier	multiplicity	default

Operation **IUMLActivityEdge::ActivityPartitions**

paramete r	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documen tation	A list of elements of type IUMLActivityPartition .					

Operation **IUMLActivityEdge::Guard**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **IUMLActivityEdge::InterruptibleActivityRegions**

paramete r	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documen tation	A list of elements of type IUMLInterruptibleActivityRegion .					

Operation **IUMLActivityEdge::Source**

paramete r	name return	direction return	type IUMLActivityNo	type modifier	multiplicity	default

	de

Operation **IUMLActivityEdge::StructuredActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLStructuredActivityNode .					

Operation **IUMLActivityEdge::Target**

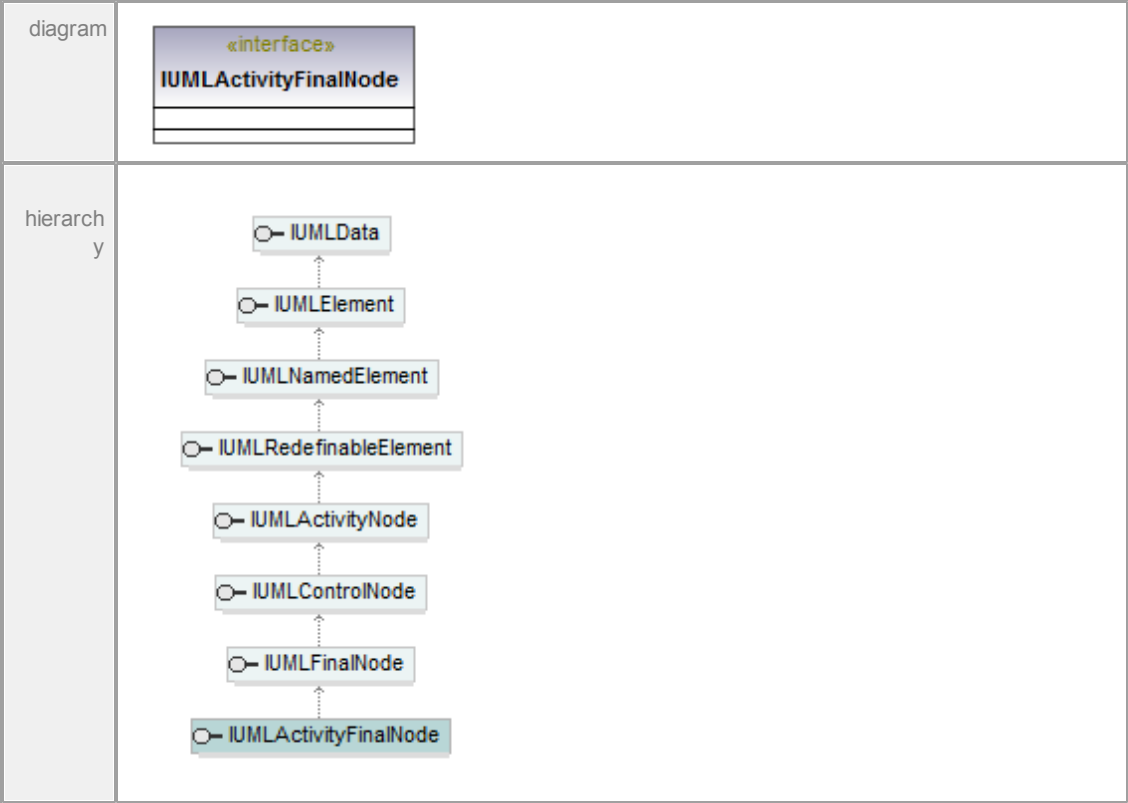
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLActivityNode			

Operation **IUMLActivityEdge::Weight**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

UModelAPI - IUMLActivityFinalNode

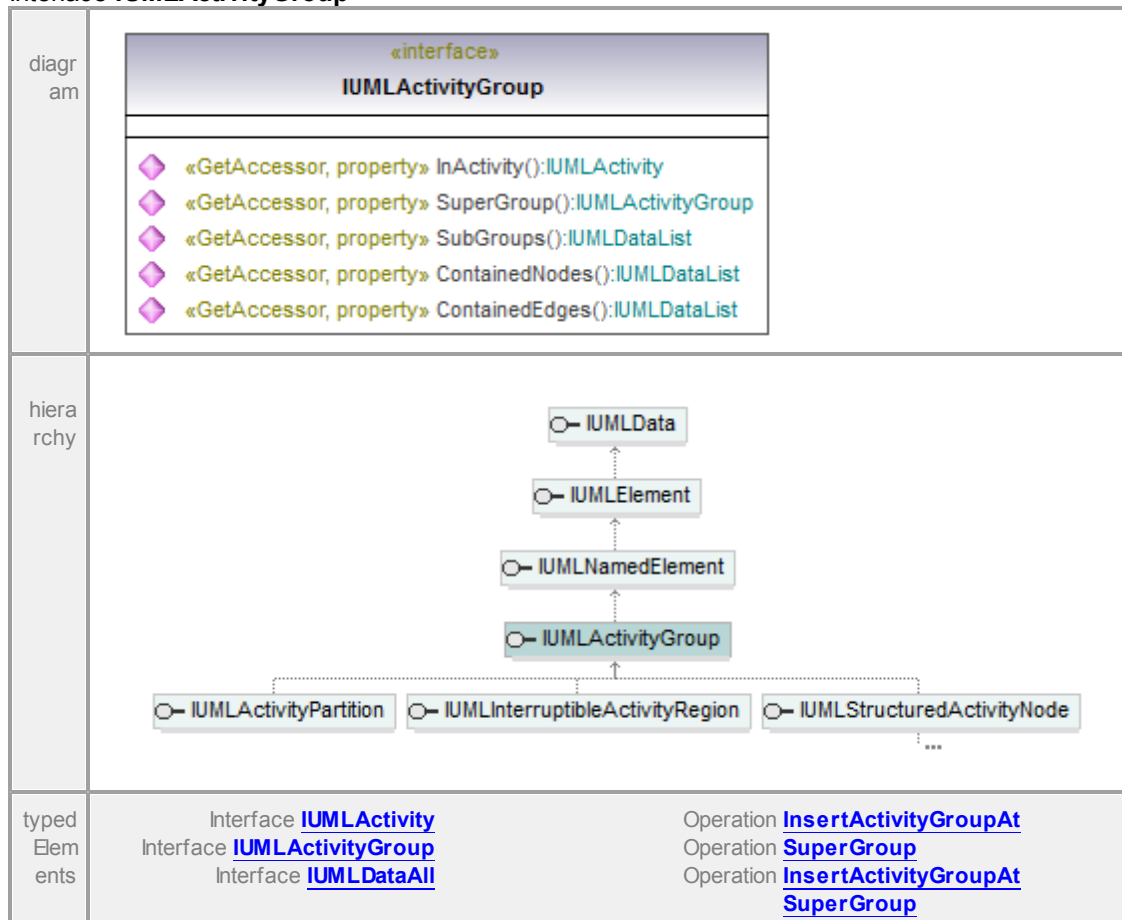
Interface **IUMLActivityFinalNode**



UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLActivityGroup

Interface **IUMLActivityGroup**Operation **IUMLActivityGroup::ContainedEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLActivityEdge .					

Operation **IUMLActivityGroup::ContainedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLActivityNode .					

Operation **IUMLActivityGroup::InActivity**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLActivity			

Operation **IUMLActivityGroup::SubGroups**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLActivityGroup .					

Operation **IUMLActivityGroup::SuperGroup**

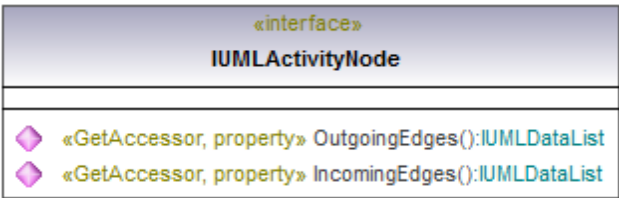
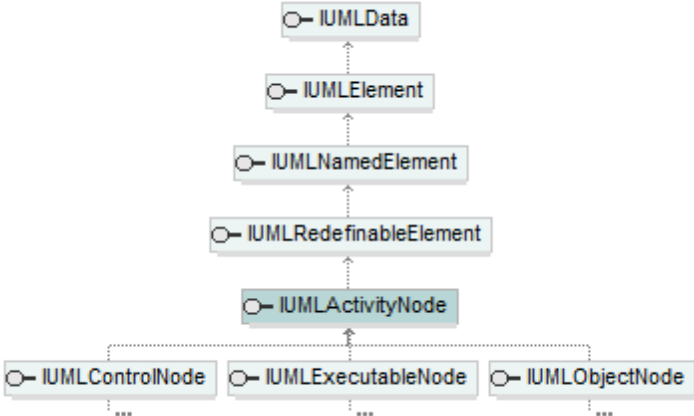
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLActivityGroup			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLActivityNode

Interface **IUMLActivityNode**

diagram		
hierarchy		
typedElements	Interface IUMLActivity Interface IUMLActivityEdge Interface IUMLActivityPartition Interface IUMLDataAll Interface IUMLInterruptibleActivityRegion Interface	Operation InsertActivityEdgeAt Operation InsertActivityNodeAt Operation Source Target Operation InsertNodeAt Operation InsertActivityEdgeAt Operation InsertActivityNodeAt Operation InsertNodeAt Source Target Operation InsertNodeAt Operation InsertNodeAt

IUMLStructuredActivityNodeOperation **IUMLActivityNode::IncomingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLActivityEdge .					

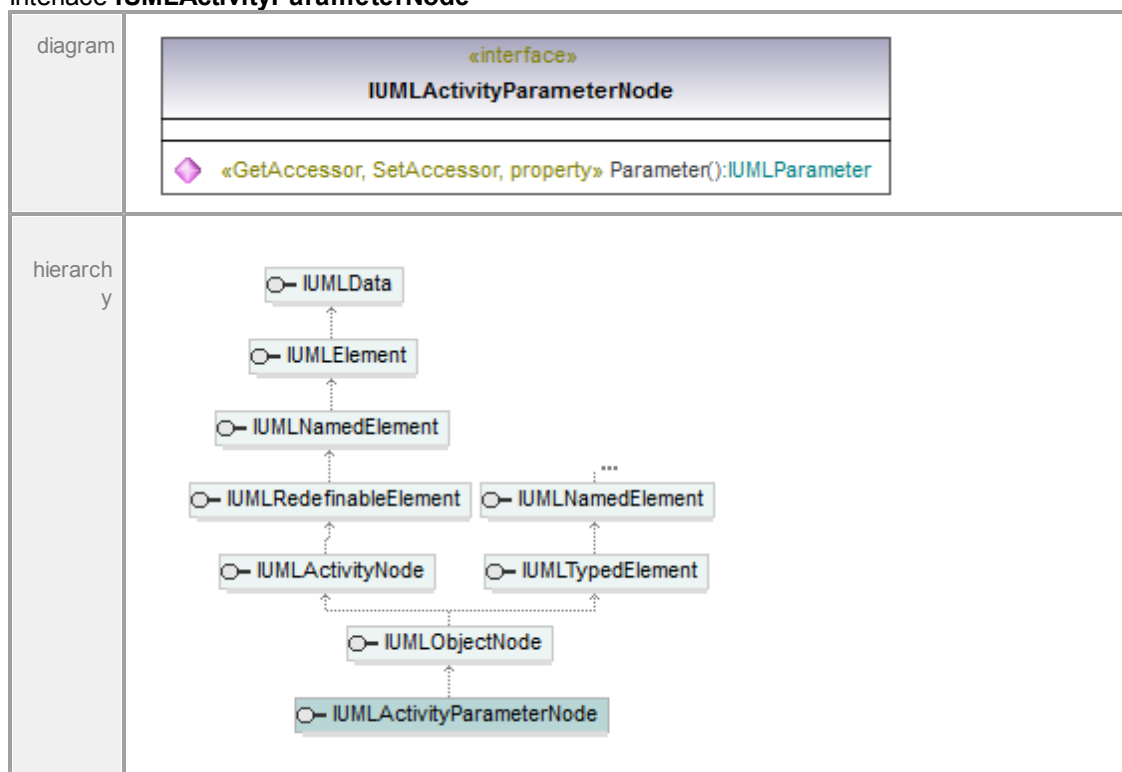
Operation **IUMLActivityNode::OutgoingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLActivityEdge .					

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLActivityParameterNode

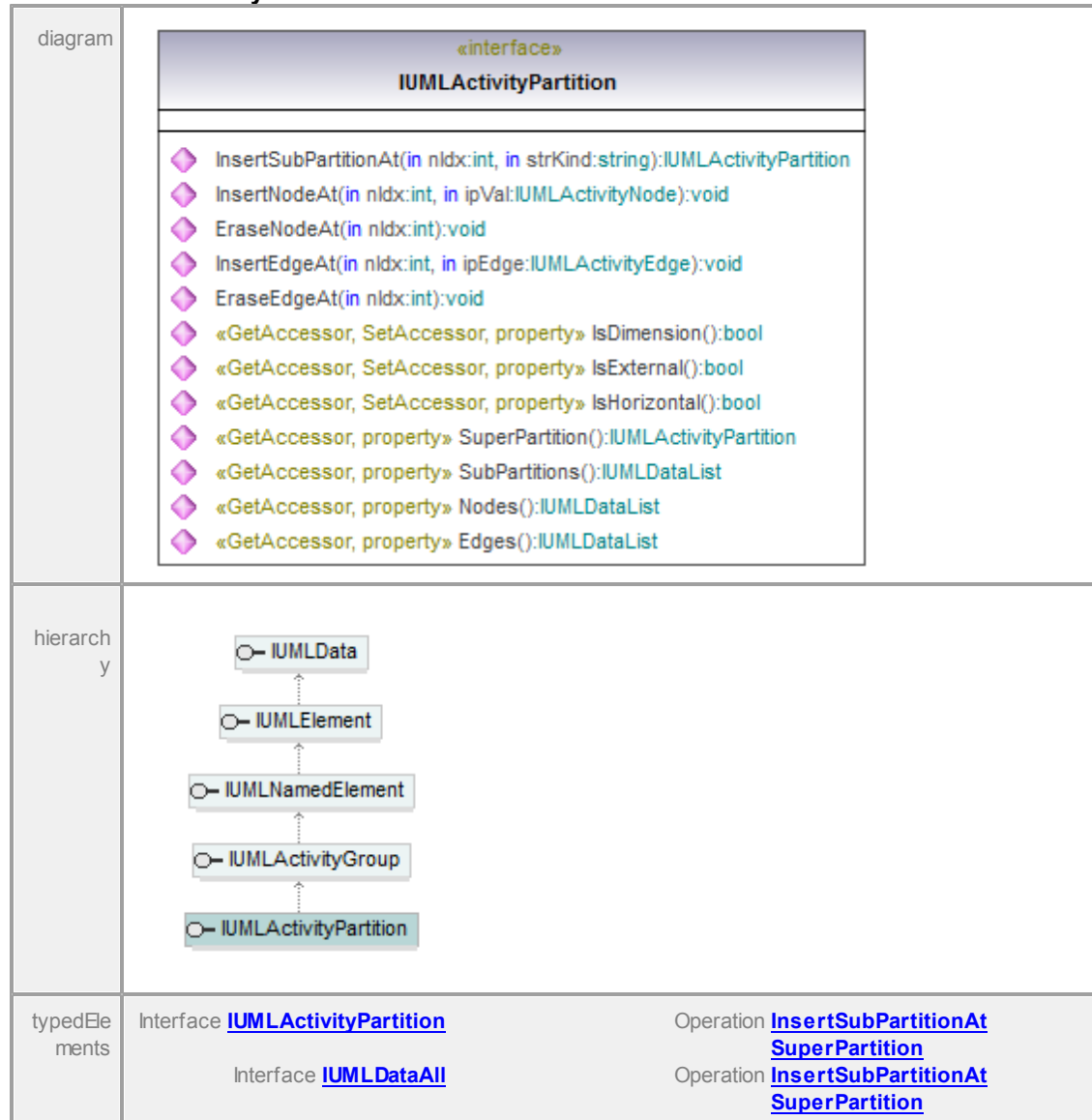
Interface **IUMLActivityParameterNode**Operation **IUMLActivityParameterNode::Parameter**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLParameter			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLActivityPartition

Interface **IUMLActivityPartition**Operation **IUMLActivityPartition::Edges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLActivityEdge .					

Operation **IUMLActivityPartition::EraseEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default

r	nldx return	in return	int void			

Operation **IUMLActivityPartition::EraseNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nldx return	in return	int void			

Operation **IUMLActivityPartition::InsertEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nldx ipEdge return	in in return	int IUMLActivityEdge void			

Operation **IUMLActivityPartition::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nldx ipVal return	in in return	int IUMLActivityNode void			

Operation **IUMLActivityPartition::InsertSubPartitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nldx strKind return	in in return	int string IUMLActivityPartition			

Operation **IUMLActivityPartition::IsDimension**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLActivityPartition::IsExternal**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLActivityPartition::IsHorizontal**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLActivityPartition::Nodes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLActivityNode .					

Operation **IUMLActivityPartition::SubPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLActivityPartition .					

Operation **IUMLActivityPartition::SuperPartition**

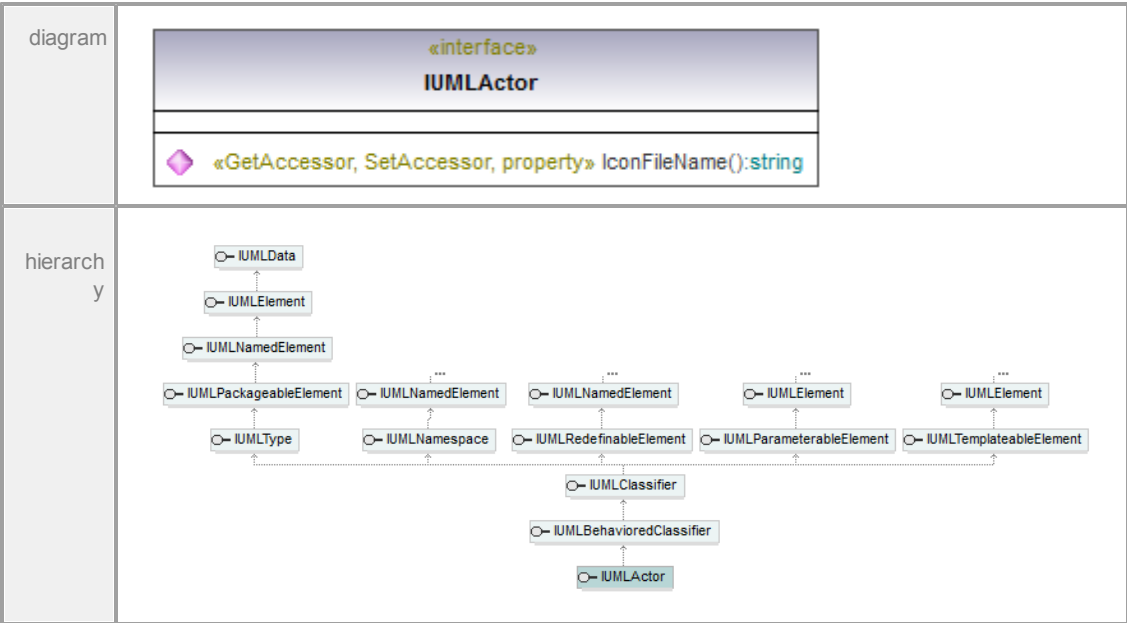
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLActivityPartition			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLActor

Interface **IUMLActor**



Operation **IUMLActor::IconFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

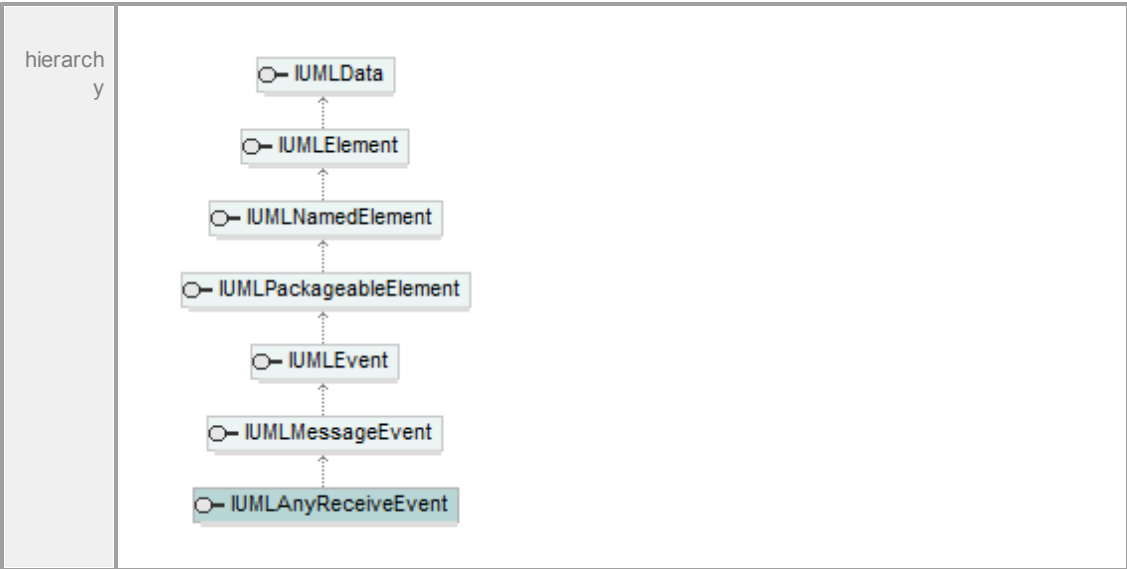
UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLAnyReceiveEvent

Interface **IUMLAnyReceiveEvent**





UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLArtifact

Interface **IUMLArtifact**

diagram	«interface» IUMLArtifact ◆ InsertManifestationAt(in nldx:int, in ipUtilizedElement:IUMLPackageableElement):IUMLManifestation ◆ InsertNestedArtifactAt(in nldx:int):IUMLArtifact ◆ InsertOwnedOperationAt(in nldx:int):IUMLOperation ◆ InsertOwnedAttributeAt(in nldx:int):IUMLProperty ◆ «GetAccessor, SetAccessor, property» FileName():string ◆ «GetAccessor, property» Manifestations():IUMLDataList ◆ «GetAccessor, property» NestedArtifacts():IUMLDataList ◆ «GetAccessor, property» OwnedAttributes():IUMLDataList ◆ «GetAccessor, property» OwnedOperations():IUMLDataList
hierarchy	
typedElements	Interface IUMLArtifact Interface IUMLDataAll Operation InsertNestedArtifactAt Operation InsertNestedArtifactAt

Operation **IUMLArtifact::FileName**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLArtifact::InsertManifestationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	ipUtilizedElement	in	IUMLPackageableElement			
	return	return	IUMLManifestation			

Operation **IUMLArtifact::InsertNestedArtifactAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	IUMLArtifact			

Operation **IUMLArtifact::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	IUMLProperty			

Operation **IUMLArtifact::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	IUMLOperation			

Operation **IUMLArtifact::Manifestations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLManifestation .					

Operation **IUMLArtifact::NestedArtifacts**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLArtifact .					

Operation **IUMLArtifact::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLProperty .					

Operation **IUMLArtifact::OwnedOperations**

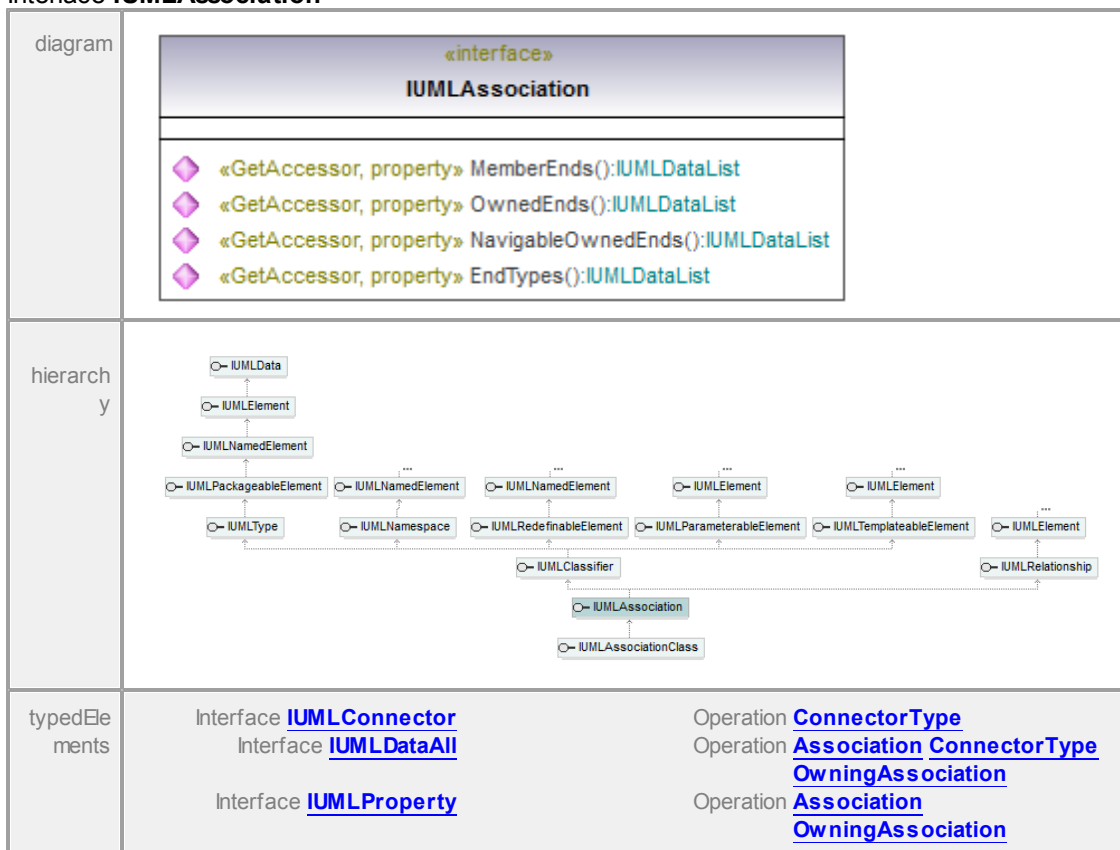
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLOperation .					

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLAssociation

Interface IUMLAssociation



Operation IUMLAssociation::EndTypes

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLType .					

Operation IUMLAssociation::MemberEnds

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLProperty .					

Operation **IUMLAssociation::NavigableOwnedEnds**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLProperty .					

Operation **IUMLAssociation::OwnedEnds**

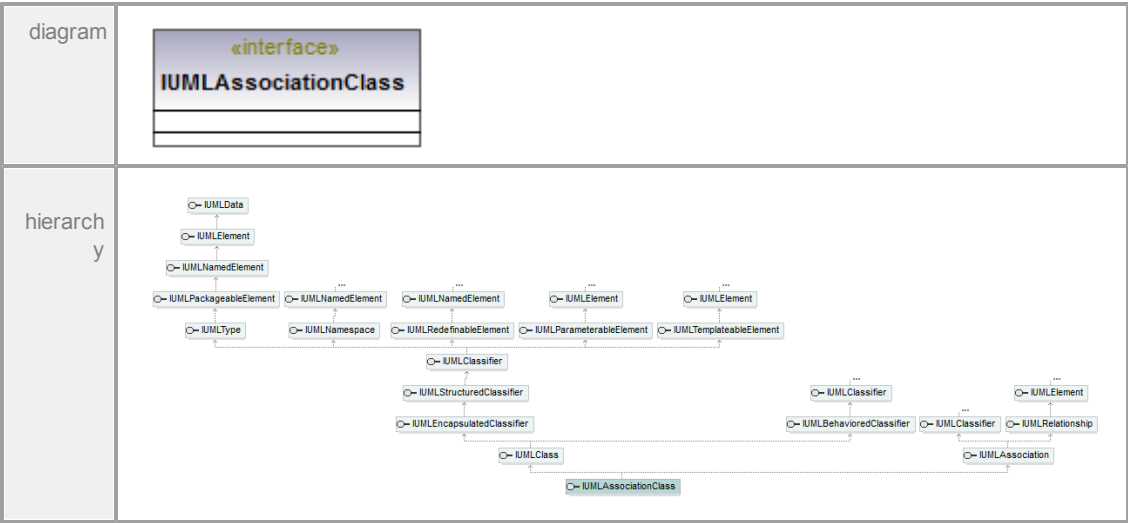
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLProperty .					

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLAssociationClass

Interface **IUMLAssociationClass**



UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLBehavior

Interface **IUMLBehavior**

diagram	<pre> «interface» IUMLBehavior +InsertPreconditionAt(in nldx:int):IUMLConstraint +InsertPostconditionAt(in nldx:int):IUMLConstraint +InsertOwnedParameterAt(in nldx:int):IUMLParameter +«GetAccessor, SetAccessor, property» IsReentrant():bool +«GetAccessor, SetAccessor, property» BehaviorSpecification():IUMLBehavioralFeature +«GetAccessor, property» Preconditions():IUMLDataList +«GetAccessor, property» Postconditions():IUMLDataList +«GetAccessor, property» OwnedParameters():IUMLDataList </pre>	
hierarchy	<pre> graph TD IUMLData --> IUMLElement IUMLElement --> IUMLNamedElement IUMLNamedElement --> IUMLPackageableElement IUMLPackageableElement --> IUMLType IUMLPackageableElement --> IUMLNamespace IUMLPackageableElement --> IUMLRedefinableElement IUMLPackageableElement --> IUMLParameterableElement IUMLPackageableElement --> IUMLTemplateableElement IUMLType --> IUMLClassifier IUMLNamespace --> IUMLClassifier IUMLRedefinableElement --> IUMLClassifier IUMLParameterableElement --> IUMLClassifier IUMLTemplateableElement --> IUMLClassifier IUMLClassifier --> IUMLStructuredClassifier IUMLClassifier --> IUMLEncapsulatedClassifier IUMLClassifier --> IUMLClass IUMLStructuredClassifier --> IUMLBehavior IUMLEncapsulatedClassifier --> IUMLBehavior IUMLClass --> IUMLBehavior IUMLBehavior --> IUMLActivity IUMLBehavior --> IUMLInteraction IUMLBehavior --> IUMLOpaqueBehavior IUMLBehavior --> IUMLStateMachine </pre>	
typedElements	Interface IUMLBehavioredClassifier Interface IUMLBehaviorExecutionSpecification Interface IUMLCallBehaviorAction Interface IUMLDataAll Interface IUMLDecisionNode Interface IUMLObjectFlow Interface IUMLObjectNode Interface IUMLState Interface IUMLTransition	Operation InsertOwnedBehaviorAt Operation BehaviorExecution Operation Behavior Operation Behavior BehaviorExecution DecisionInput DoActivity Effect Entry Exit InsertOwnedBehaviorAt Selection SetNew DoActivity SetNew Effect SetNew Entry SetNew Exit Transformation Operation DecisionInput Operation Transformation Operation Selection Operation DoActivity Entry Exit SetNew DoActivity SetNew Entry SetNew Exit Effect SetNew Effect

Operation **IUMLBehavior::BehaviorSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior IFeature			

Operation **IUMLBehavior::InsertOwnedParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int IUMLParameter <u>r</u>			

Operation **IUMLBehavior::InsertPostconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int IUMLConstraint <u>t</u>			

Operation **IUMLBehavior::InsertPreconditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int IUMLConstraint <u>t</u>			

Operation **IUMLBehavior::IsReentrant**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLBehavior::OwnedParameters**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLParameter .					

Operation **IUMLBehavior::Postconditions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLConstraint .					

Operation **IUMLBehavior::Preconditions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLConstraint .					

UModelAPI - IUMLBehavioralFeature

Interface **IUMLBehavioralFeature**

diagram	«interface» IUMLBehavioralFeature ◆ InsertOwnedParameterAt(in nldx:int):IUMLParameter ◆ InsertRaisedExceptionAt(in nldx:int, in ipVal:IUMLType):void ◆ EraseRaisedExceptionAt(in nldx:int):void ◆ «GetAccessor, property» OwnedParameters():IUMLDataList ◆ «GetAccessor, property» RaisedExceptions():IUMLDataList ◆ «GetAccessor, SetAccessor, property» IsAbstract():bool ◆ «GetAccessor, property» Methods():IUMLDataList ◆ «GetAccessor, SetAccessor, property» Concurrency():ENUMUMLCallConcurrencyKind
hierarch y	IUMLData ⋮ IUMLElement IUMLNamedElement ⋮ IUMLRedefinableElement IUMLFeature ⋮ IUMLBehavioralFeature IUMLOperation ⋮ IUMLBehavioralFeature IUMLNamedElement ⋮ IUMLNamespace IUMLBehavioralFeature ⋮ IUMLBehavioralFeature
typedEle ments	Interface IUMLBehavior Interface IUMLDataAll Operation BehaviorSpecification Operation BehaviorSpecification

Operation **IUMLBehavioralFeature::Concurrency**

paramete r	name return	direction return	type ENUMUMLCallC oncurrencyKin d	type modifier	multiplicity	default

Operation **IUMLBehavioralFeature::EraseRaisedExceptionAt**

paramete r	name nldx return	direction in return	type int void	type modifier	multiplicity	default

Operation **IUMLBehavioralFeature::InsertOwnedParameterAt**

paramete	name	direction	type	type modifier	multiplicity	default

r	nldx return	in return	int IUMLParameter r

Operation **IUMLBehavioralFeature::InsertRaisedExceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nldx ipVal return	in in return	int IUMLType void			

Operation **IUMLBehavioralFeature::IsAbstract**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLBehavioralFeature::Methods**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLBehavior .					

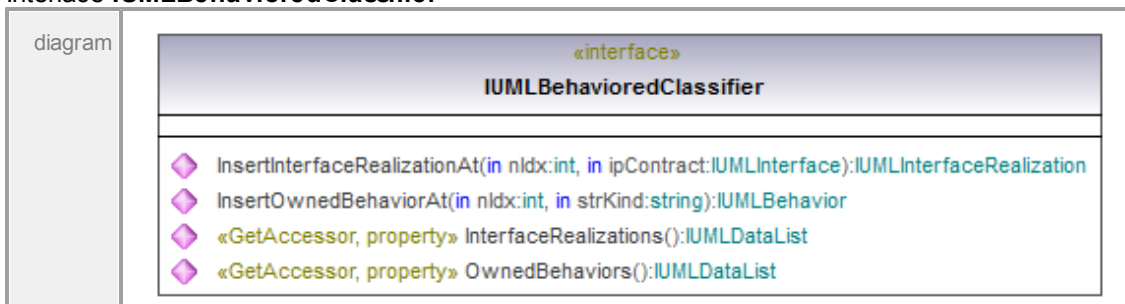
Operation **IUMLBehavioralFeature::OwnedParameters**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLParameter .					

Operation **IUMLBehavioralFeature::RaisedExceptions**

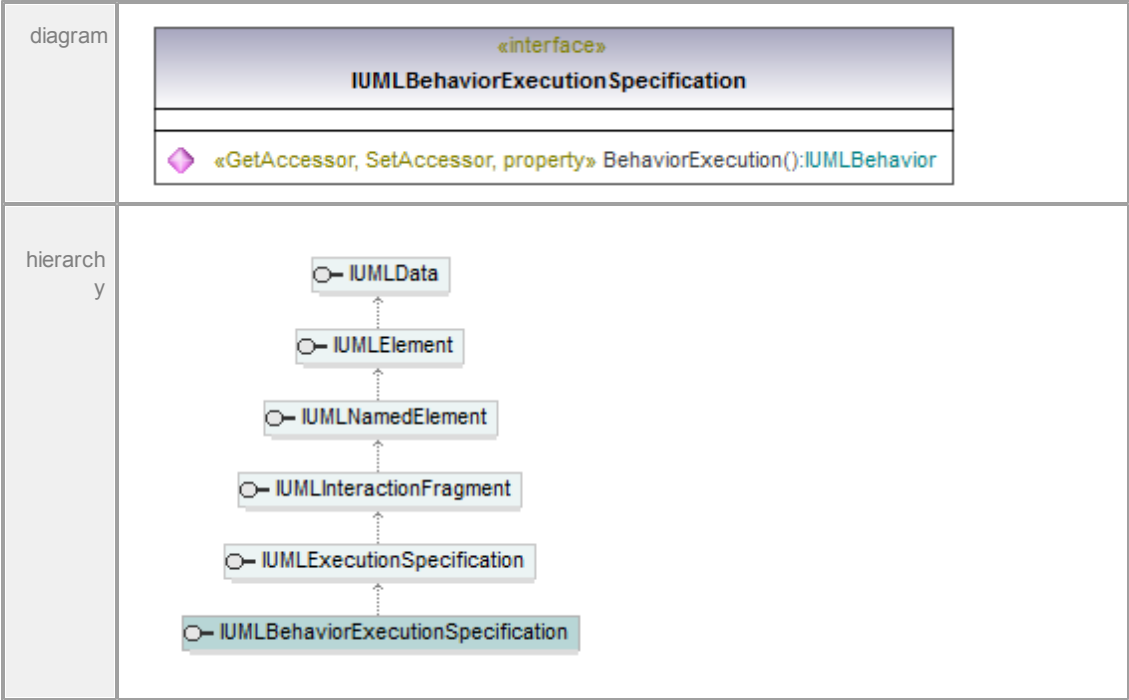
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLType .					

UModelAPI - IUMLBehavoredClassifier

Interface **IUMLBehavoredClassifier**

UModelAPI - IUMLBehaviorExecutionSpecification

Interface IUMLBehaviorExecutionSpecification

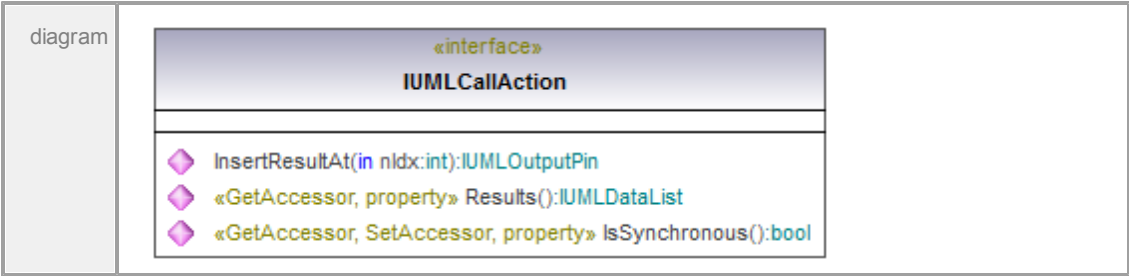


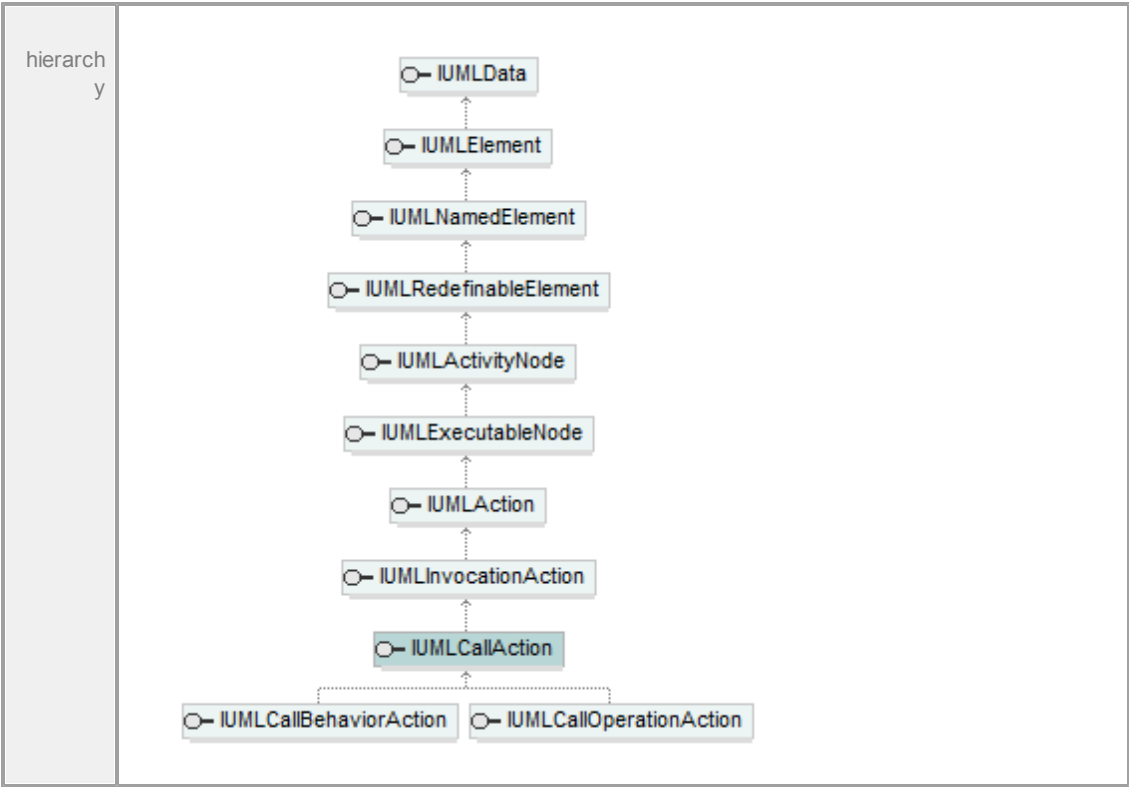
Operation IUMLBehaviorExecutionSpecification::BehaviorExecution

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLBehavior			

UModelAPI - IUMLCallAction

Interface IUMLCallAction





Operation **IUMLCallAction::InsertResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	return	return	IUMLOutputPin			

Operation **IUMLCallAction::IsSynchronous**

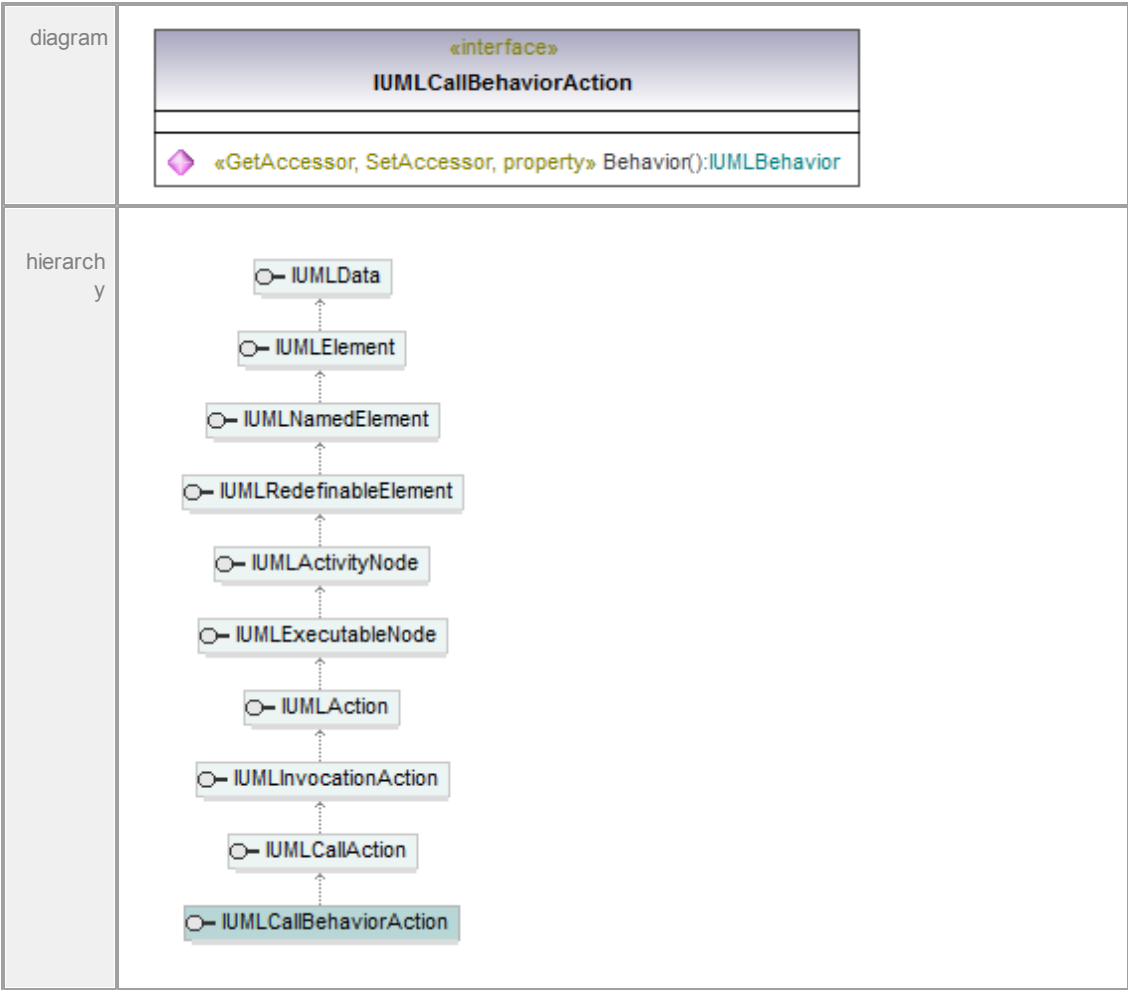
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLCallAction::Results**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLOutputPin .					

UModelAPI - IUMLCallBehaviorAction

Interface **IUMLCallBehaviorAction**



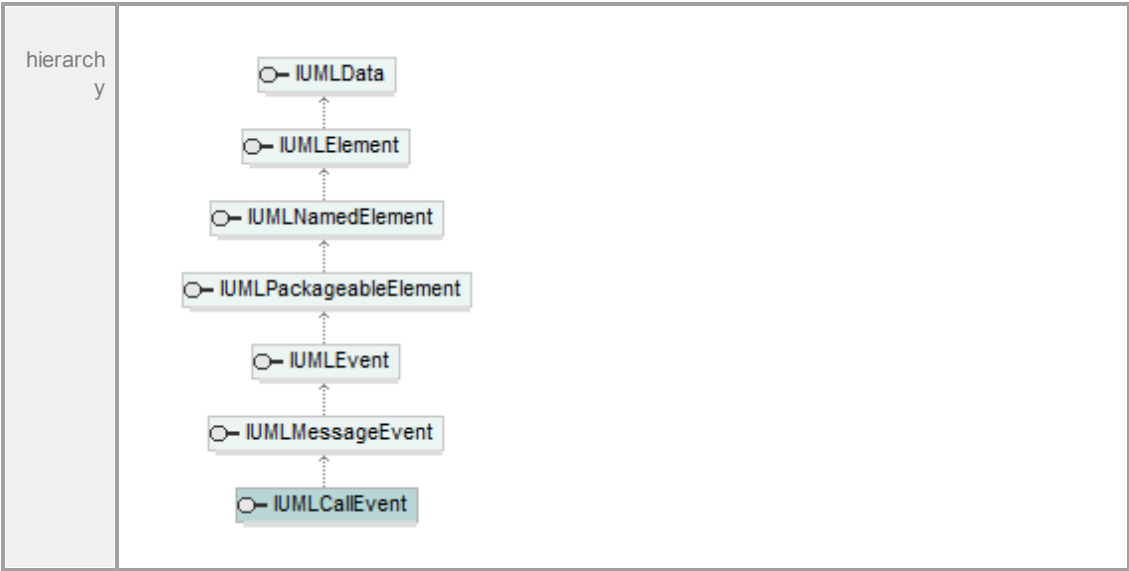
Operation **IUMLCallBehaviorAction::Behavior**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	<u>IUMLBehavior</u>			

UModelAPI - IUMLCallEvent

Interface **IUMLCallEvent**





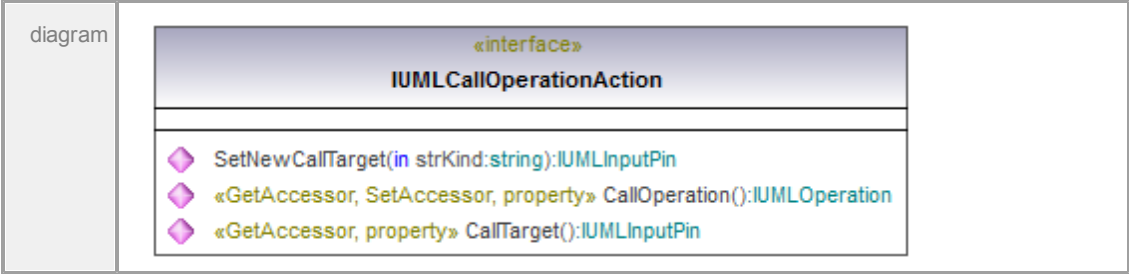
Operation **IUMLCallEvent::Operation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLOperation			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLCallOperationAction

Interface **IUMLCallOperationAction**





Operation **IUMLCallOperationAction::CallOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLOperation			

Operation **IUMLCallOperationAction::CallTarget**

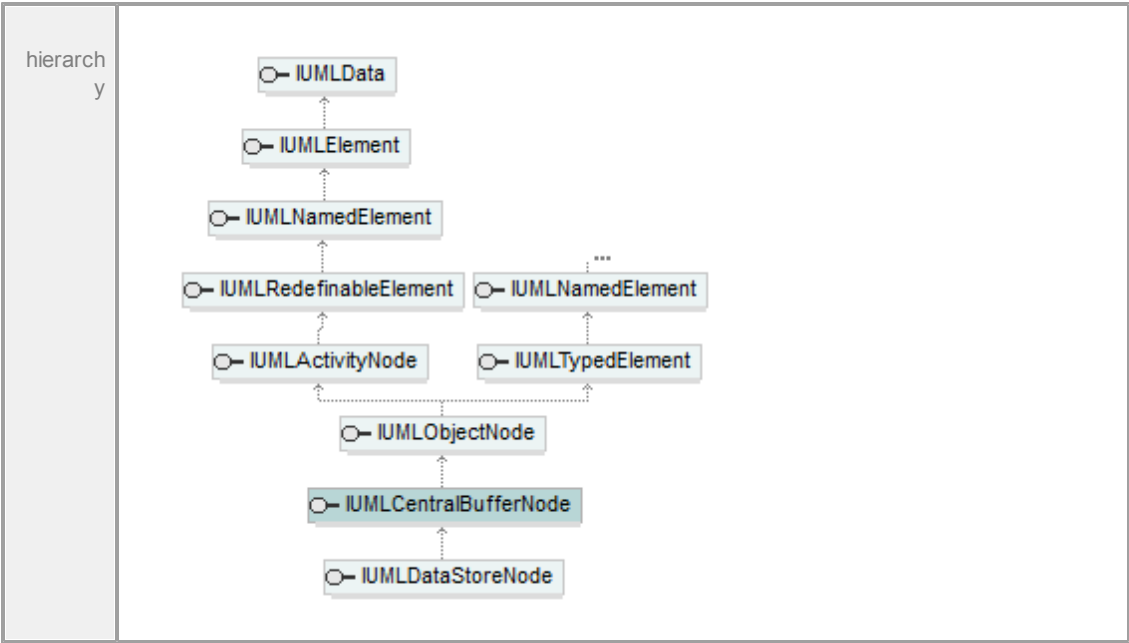
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLInputPin			

Operation **IUMLCallOperationAction::SetNewCallTarget**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind return	in return	string IUMLInputPin			

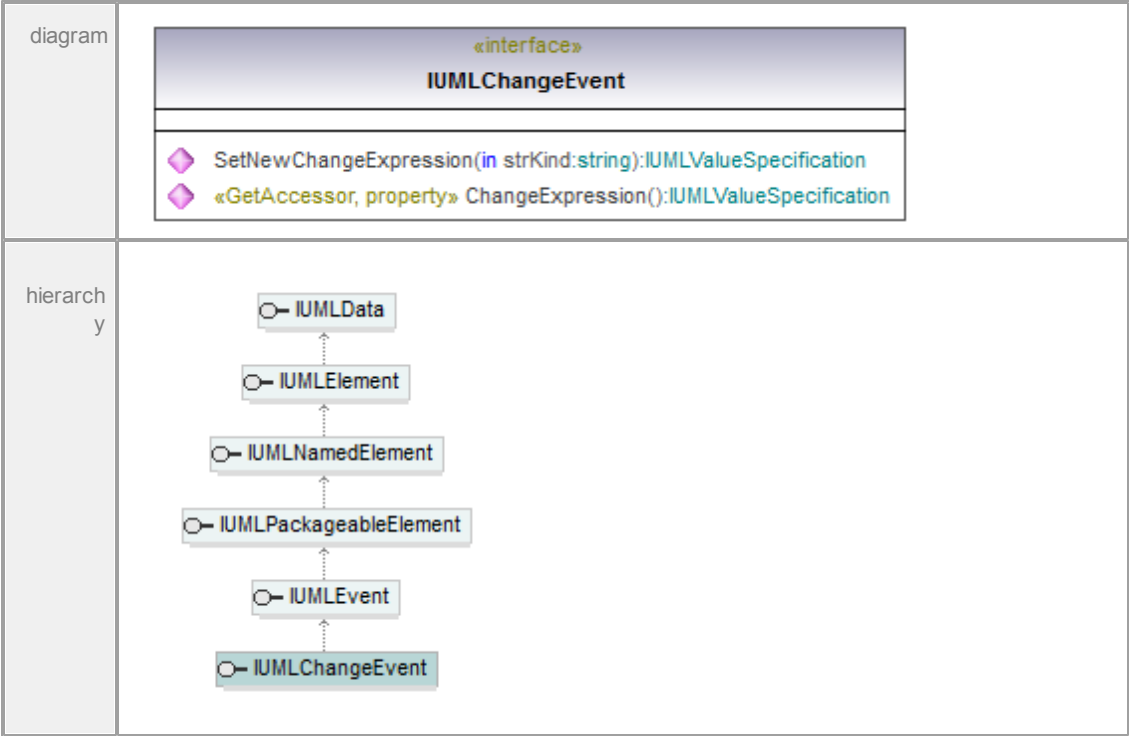
Interface **IUMLCentralBufferNode**





UModelAPI - IUMLChangeEvent

Interface IUMLChangeEvent



Operation IUMLChangeEvent::ChangeExpression

paramete	name	direction	type	type modifier	multiplicity	default

r	return	return	IUMLValueSpecification

Operation **IUMLChangeEvent::SetNewChangeExpression**

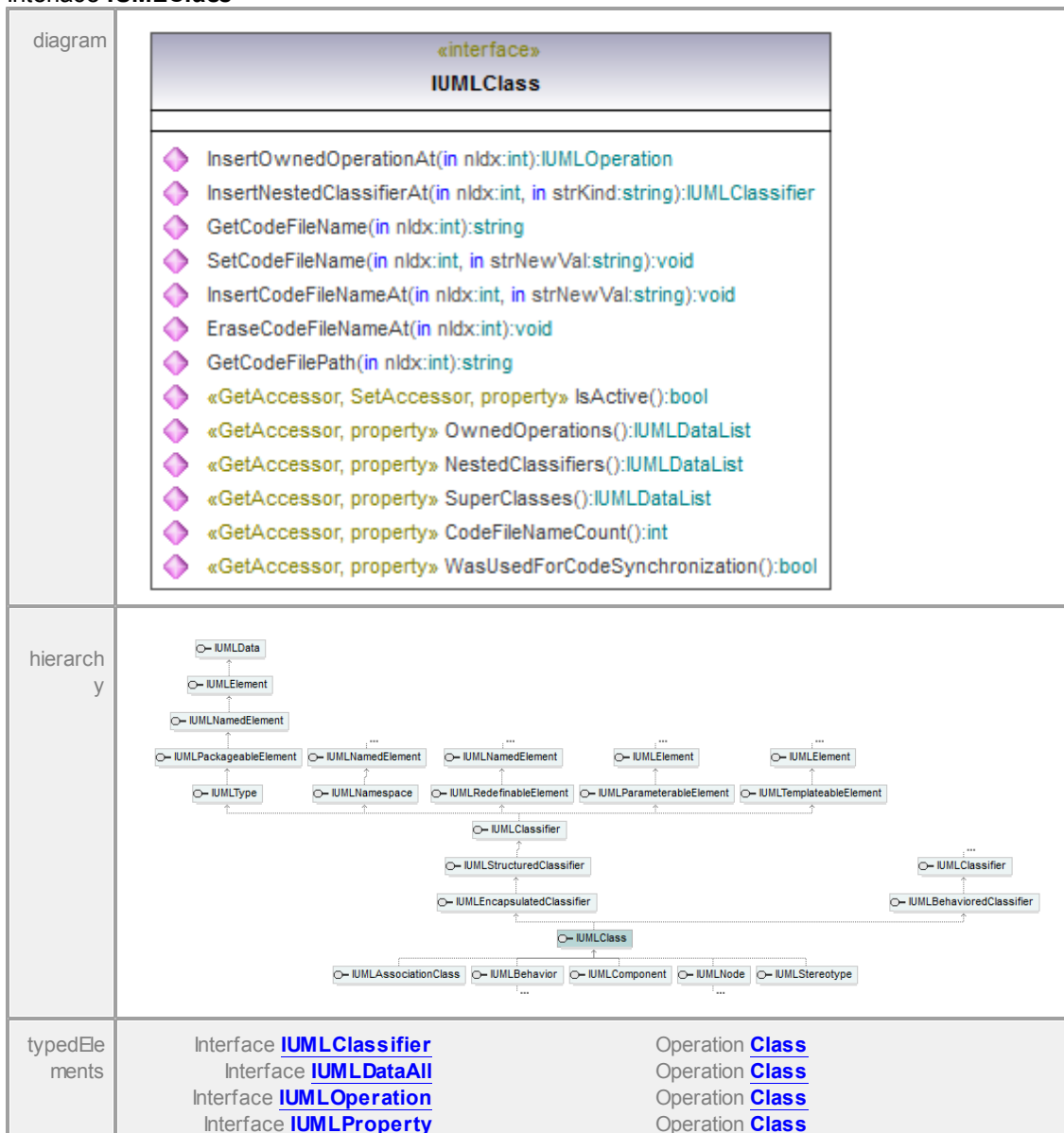
parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLValueSpecification			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLClass

Interface **IUMLClass**



Operation **IUMLClass::CodeFileNameCount**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLClass::EraseCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int void			

Operation **IUMLClass::GetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int string			

Operation **IUMLClass::GetCodeFilePath**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int string			
documentation	get the full code file path					

Operation **IUMLClass::InsertCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx strNewVal return	in in return	int string void			

Operation **IUMLClass::InsertNestedClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx strKind return	in in return	int string IUMLClassifier			

Operation **IUMLClass::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int IUMLOperation			

Operation **IUMLClass::IsActive**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLClass::NestedClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

document tation	A list of elements of type IUMLClassifier .

Operation **IUMLClass::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
document tation	A list of elements of type IUMLOperation .					

Operation **IUMLClass::SetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	strNewVal	in	string			
	return	return	void			

Operation **IUMLClass::SuperClasses**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
document tation	A list of elements of type IUMLClass .					

Operation **IUMLClass::WasUsedForCodeSynchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

UModelAPI - IUMLClassifier

Interface **IUMLClassifier**

diagram		
hierarchy		
typedElements	Interface IUMLAction Interface IUMLClass Interface IUMLClassifier Interface IUMLClassifierTemplateParameter Interface IUMLComponent Interface IUMLComponentRealization Interface IUMLDataAll	Operation ActionContext Operation InsertNestedClassifierAt Operation InsertGeneralizationAt Operation InsertConstrainingClassifierAt Operation InsertRealizationAt Operation RealizingClassifier Operation ActionContext Classifier General InsertConstrainingClassifierAt InsertConveyedAt InsertExceptionTypeAt InsertGeneralizationAt InsertNestedClassifierAt InsertRealizationAt InsertSubjectAt

	Interface	Operation	RealizingClassifier Specific
	IUMLExceptionHandler	Operation	InsertExceptionTypeAt
	Interface IUMLGeneralization	Operation	General Specific
	Interface IUMLInformationFlow	Operation	InsertConveyedAt
	Interface	Operation	Classifier
	IUMLInstanceSpecification	Operation	InsertNestedClassifierAt
	Interface IUMLInterface	Operation	Classifier
	Interface IUMLProperty	Operation	InsertSubjectAt
	Interface IUMLUseCase		

Operation **IUMLClassifier::Attributes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLProperty .					

Operation **IUMLClassifier::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLClass			

Operation **IUMLClassifier::CollaborationUses**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLCollaborationUse .					

Operation **IUMLClassifier::Features**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLFeature .					

Operation **IUMLClassifier::Generalizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLGeneralization .					

Operation **IUMLClassifier::Generals**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLClassifier .					

Operation **IUMLClassifier::InheritedMembers**

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documentation	A list of elements of type IUMLNamedElement .					

Operation **IUMLClassifier::InsertCollaborationUseAt**

parameter	name nidx return	direction in return	type int IUMLCollaborationUse	type modifier	multiplicity	default

Operation **IUMLClassifier::InsertGeneralizationAt**

parameter	name nidx ipGeneral return	direction in in return	type int IUMLClassifier IUMLGeneralization	type modifier	multiplicity	default

Operation **IUMLClassifier::InsertOwnedUseCaseAt**

parameter	name nidx return	direction in return	type int IUMLUseCase	type modifier	multiplicity	default

Operation **IUMLClassifier::IsAbstract**

parameter	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IUMLClassifier::IsFinalSpecialization**

parameter	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IUMLClassifier::NestingInterface**

parameter	name return	direction return	type IUMLInterface	type modifier	multiplicity	default

Operation **IUMLClassifier::OwnedUseCases**

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documentation	A list of elements of type IUMLUseCase .					

Operation **IUMLClassifier::Specifics**

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documentation	A list of elements of type IUMLClassifier .					

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLClassifierTemplateParameter::InsertConstrainingClassifierAt**

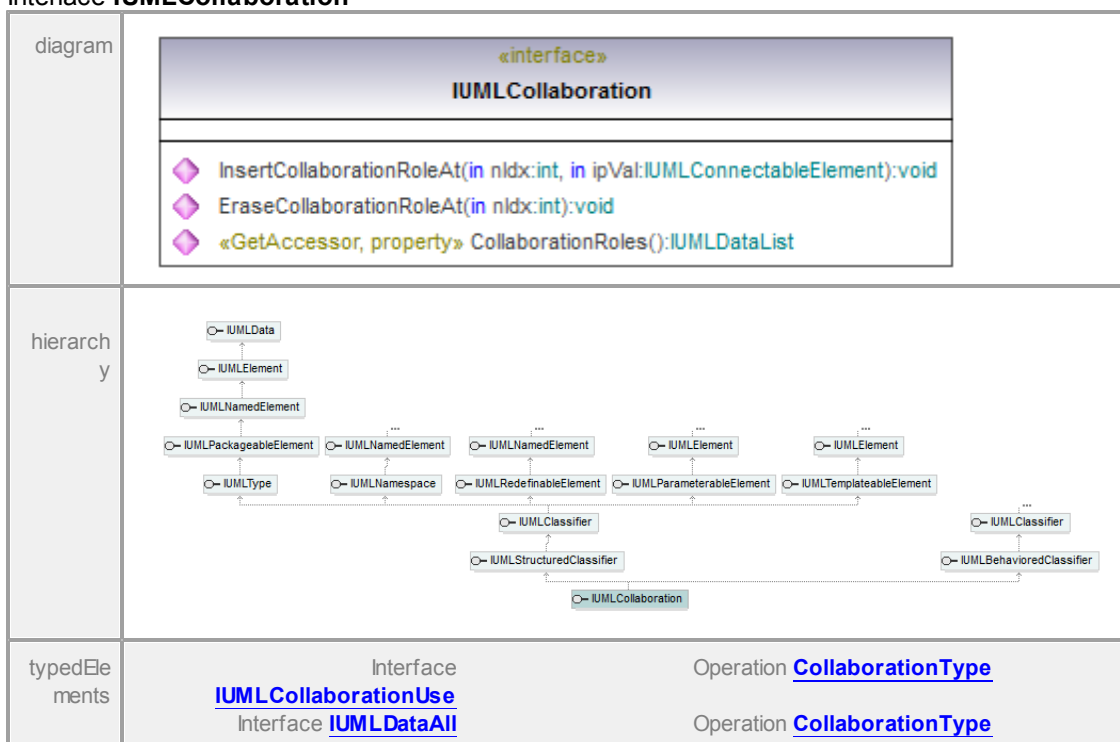
parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	ipVal	in	IUMLClassifier			
	return	return	void			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLCollaboration

Interface **IUMLCollaboration**



Operation **IUMLCollaboration::CollaborationRoles**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLCollaboration .					

Operation **IUMLCollaboration::EraseCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLCollaboration::InsertCollaborationRoleAt**

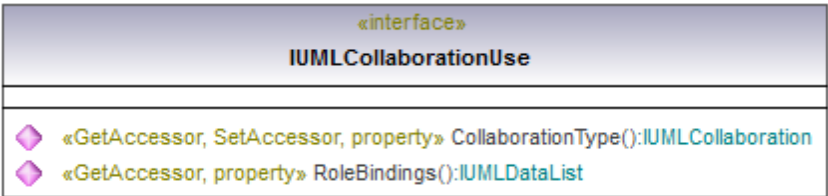
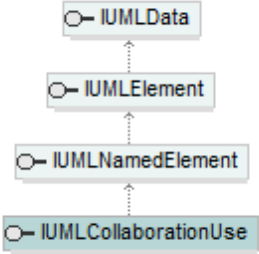
parameter	name	direction	type	type modifier	multiplicity	default
	idx	in	int			
	ipVal	in	IUMLConnectableElement			
	return	return	void			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLCollaborationUse

Interface **IUMLCollaborationUse**

diagram		
hierarchy		
typedElements	Interface IUMLClassifier Interface IUMLDataAll	Operation InsertCollaborationUseAt Operation InsertCollaborationUseAt

Operation **IUMLCollaborationUse::CollaborationType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLCollaboration			

Operation **IUMLCollaborationUse::RoleBindings**

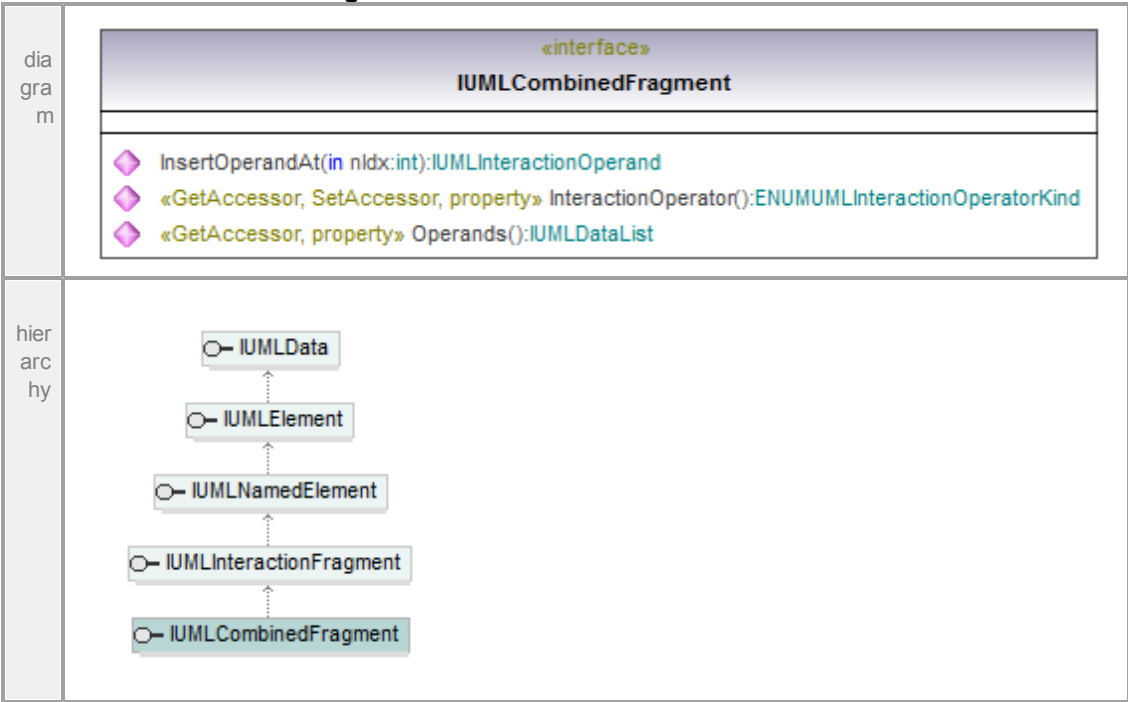
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLDependency .					

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLCombinedFragment

Interface **IUMLCombinedFragment**



Operation **IUMLCombinedFragment::InsertOperandAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	IUMLInteractionOperand			

Operation **IUMLCombinedFragment::InteractionOperator**

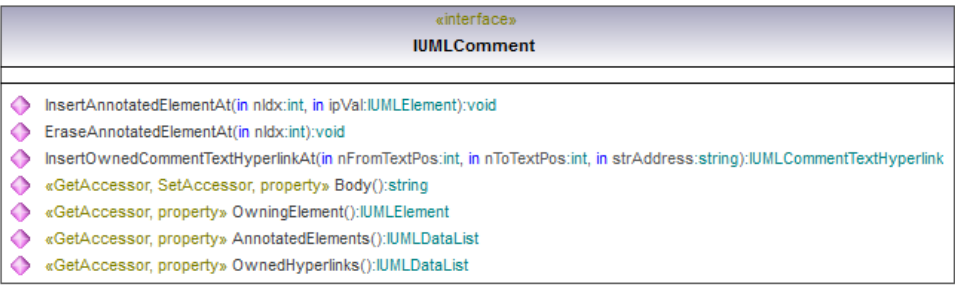
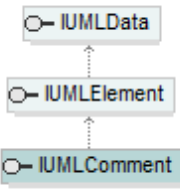
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ENUMUMLInteractionOperatorKind			

Operation **IUMLCombinedFragment::Operands**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLInteractionOperand .					

UModelAPI - IUMLComment

Interface **IUMLComment**

diagram		
hierarch y		
typedEle ments	Interface IUMLDataAll Interface IUMLElement	Operation InsertOwnedCommentAt Operation OwnedDocComment Operation InsertOwnedCommentAt Operation OwnedDocComment

Operation **IUMLComment::AnnotatedElements**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documen tation	A list of elements of type IUMLElement .					

Operation **IUMLComment::Body**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLComment::EraseAnnotatedElementAt**

paramete r	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	void			

Operation **IUMLComment::InsertAnnotatedElementAt**

paramete r	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLElement			
	return	return	void			

Operation **IUMLComment::InsertOwnedCommentTextHyperlinkAt**

paramete	name	direction	type	type modifier	multiplicity	default

r	nFromTextPos	in	int			
	nToTextPos	in	int			
	strAddress	in	string			
	return	return	IUMLComment			
			TextHyperlink			

Operation **IUMLComment::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLComment::OwningElement**

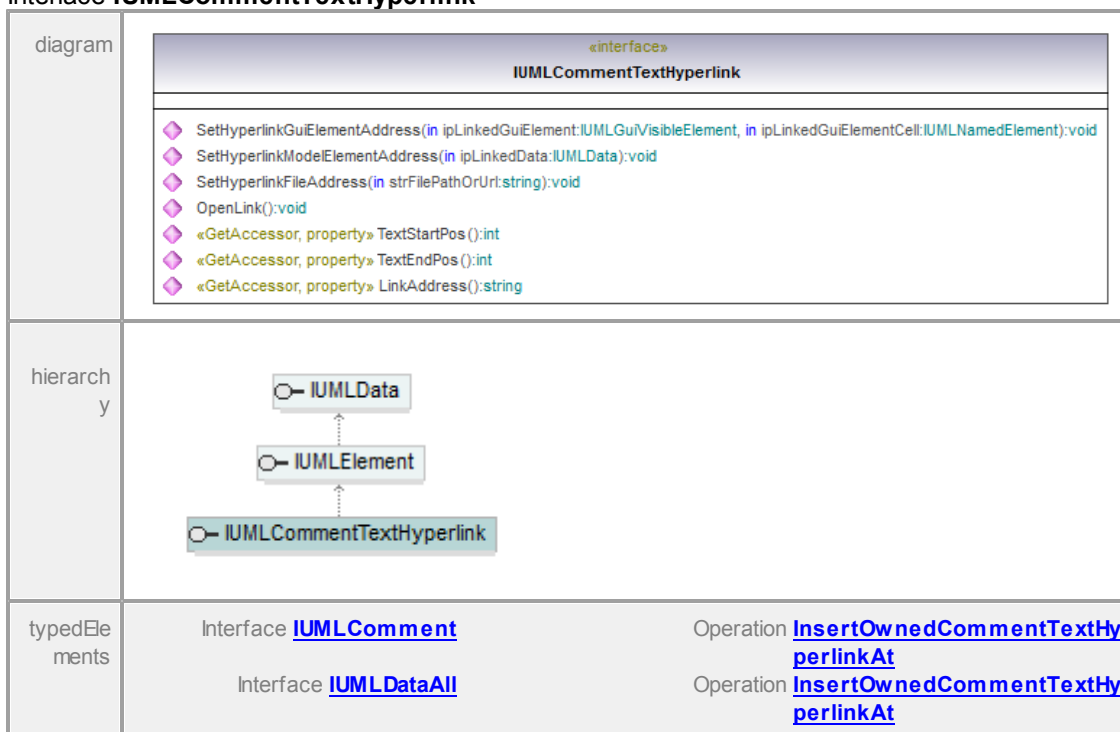
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLElement			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLCommentTextHyperlink

Interface **IUMLCommentTextHyperlink**



Operation **IUMLCommentTextHyperlink::LinkAddress**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLCommentTextHyperlink::OpenLink**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	void

Operation **IUMLCommentTextHyperlink::SetHyperlinkFileAddress**

parameter	name	direction	type	type modifier	multiplicity	default
r	strFilePathOrUrl	in	string			
	return	return	void			

Operation **IUMLCommentTextHyperlink::SetHyperlinkGuiElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipLinkedGuiElement	in	IUMLGuiVisibleElement			
	ipLinkedGuiElementCell	in	IUMLNamedElement			
	return	return	void			

Operation **IUMLCommentTextHyperlink::SetHyperlinkModelElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipLinkedData	in	IUMLData			
	return	return	void			

Operation **IUMLCommentTextHyperlink::TextEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

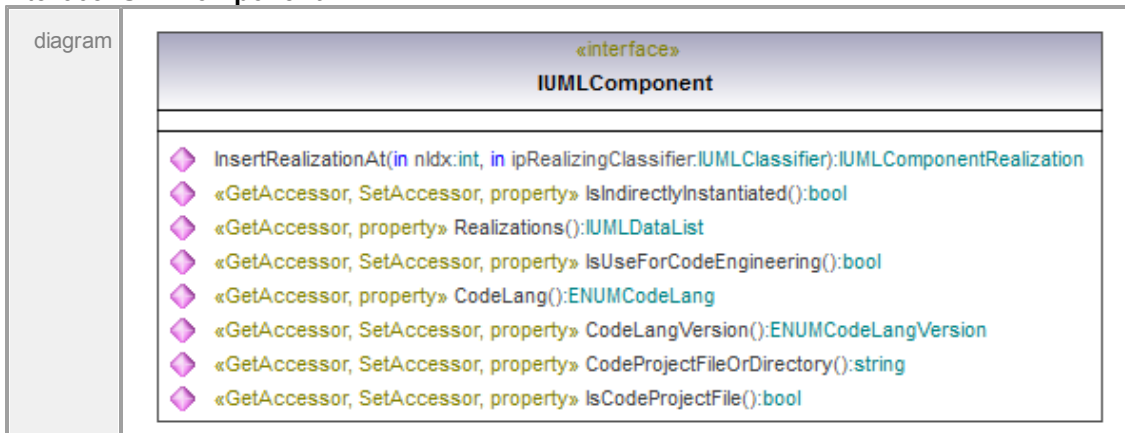
Operation **IUMLCommentTextHyperlink::TextStartPos**

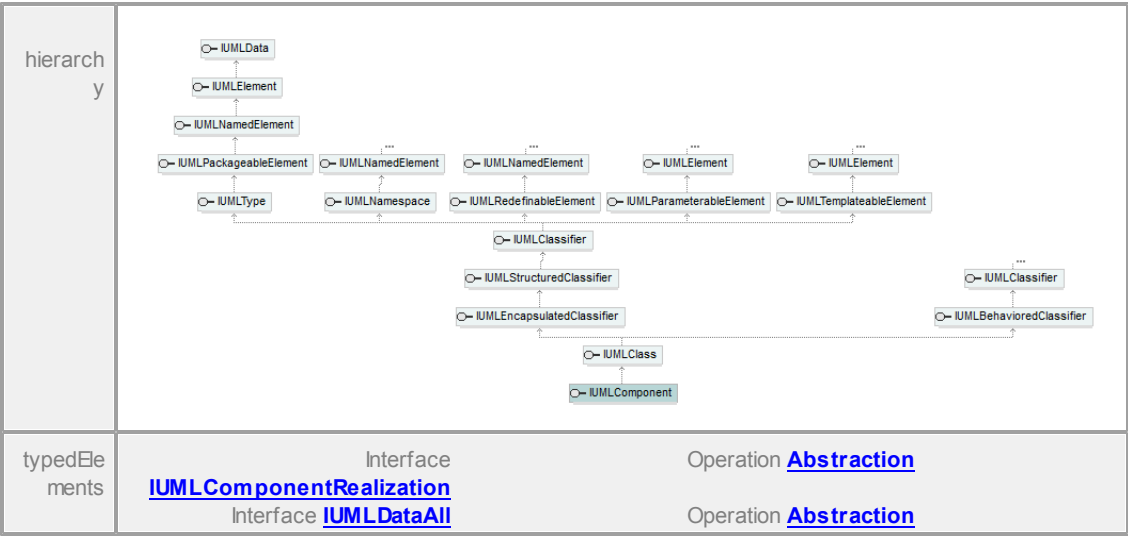
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLComponent

Interface **IUMLComponent**



Operation **IModelComponent::CodeLang**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ENUMCodeLang			

Operation **IModelComponent::CodeLangVersion**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ENUMCodeLangVersion			

Operation **IModelComponent::CodeProjectFileOrDirectory**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IModelComponent::InsertRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipRealizingClassifier	in	IModelClassifier			
	return	return	IModelComponentRealization			

Operation **IModelComponent::IsCodeProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IModelComponent::IsIndirectlyInstantiated**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IModelComponent::IsUseForCodeEngineering**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

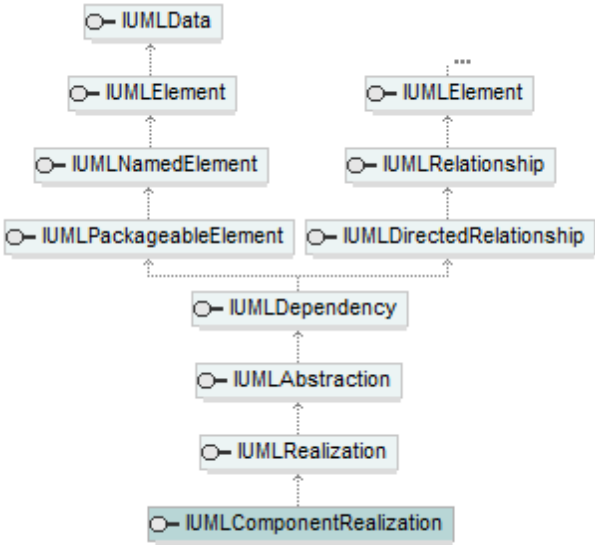
Operation **IUMLComponent::Realizations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLComponentRealization .					

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLComponentRealization

Interface **IUMLComponentRealization**

diagram		
hierarchy		
typedElements	Interface IUMLComponent Interface IUMLDataAll	Operation InsertRealizationAt Operation InsertRealizationAt

Operation **IUMLComponentRealization::Abstraction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLComponent			

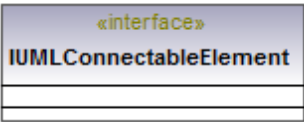
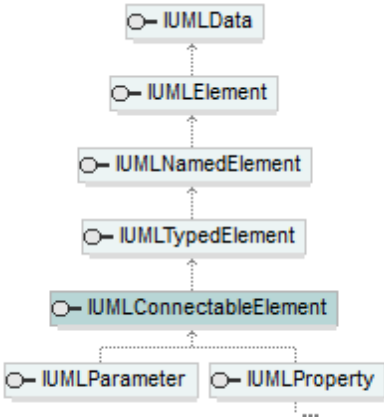
Operation **IUMLComponentRealization::RealizingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLClassifier			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLConnectableElement

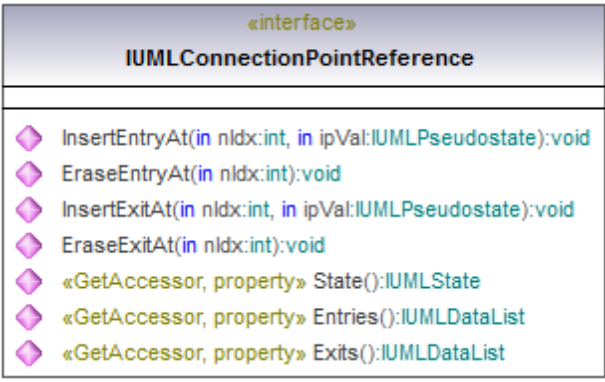
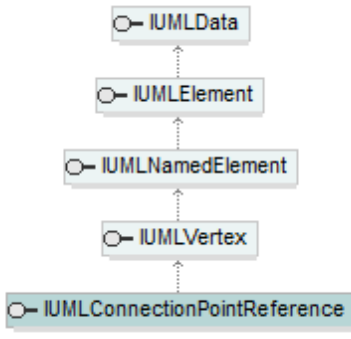
Interface **IUMLConnectableElement**

diagram			
hierarchy			
typedElements	Interface IUMLCollaboration Interface IUMLConnectorEnd Interface IUMLDataAll Interface IUMLLifeLine Interface IUMLStructuredClassifier	Operation InsertCollaborationRoleAt Operation Role Operation InsertCollaborationRoleAt InsertOwnedConnectorAt Represents Role Operation Represents Operation InsertOwnedConnectorAt	

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLConnectionPointReference

Interface **IUMLConnectionPointReference**

diagram		
hierarchy		
typedElements	Interface IUMLDataAll Interface IUMLState	Operation InsertConnectionAt Operation InsertConnectionAt

Operation **IUMLConnectionPointReference::Entries**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLPseudostate .					

Operation **IUMLConnectionPointReference::EraseEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	void			

Operation **IUMLConnectionPointReference::EraseExitAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	void			

Operation **IUMLConnectionPointReference::Exits**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	IUMLDataList
documentation	A list of elements of type IUMLPseudostate .		

Operation **IUMLConnectionPointReference::InsertEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLPseudostate			
	return	return	void			

Operation **IUMLConnectionPointReference::InsertExitAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLPseudostate			
	return	return	void			

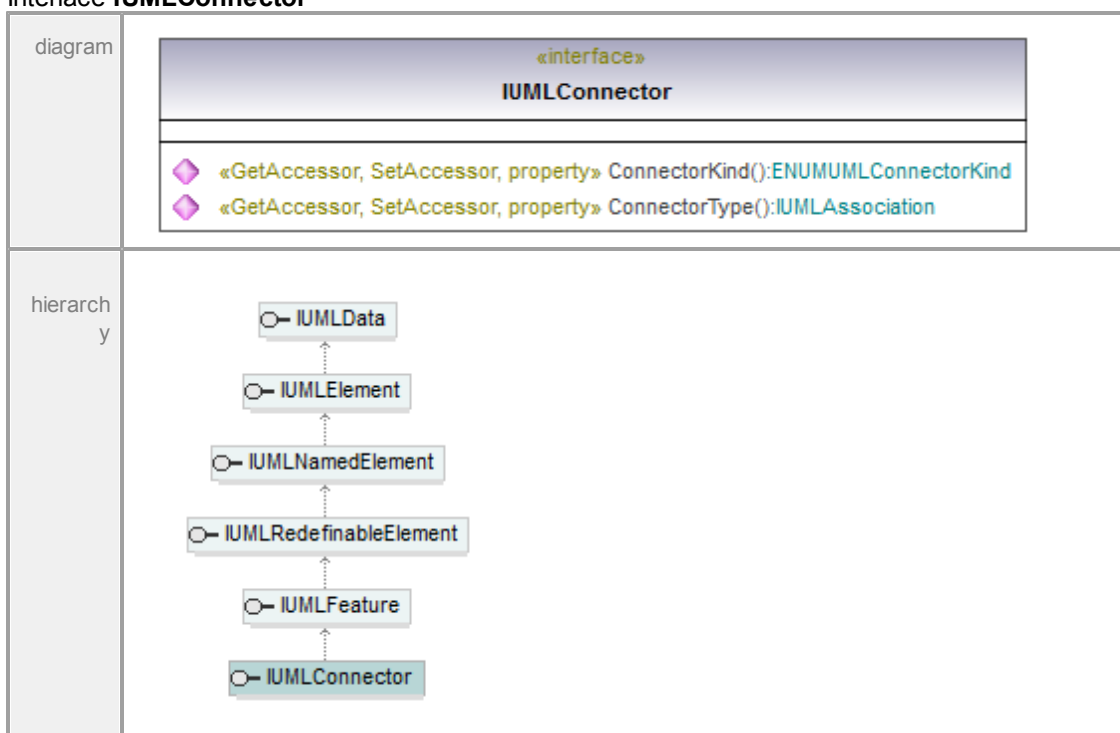
Operation **IUMLConnectionPointReference::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLState			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLConnector

Interface **IUMLConnector**

typedElements	Interface IUMLDataAll	Operation InsertOwnedConnectorAt
	Interface IUMLInformationFlow	Operation InsertRealizingConnectorAt
	Interface IUMLStructuredClassifier	Operation InsertOwnedConnectorAt

Operation **IUMLConnector::ConnectorKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ENUMUMLConnectorKind			
documentation	Deprecated: Since UML2.3 (UModel2010r2) 'ConnectorKind' is derived and cannot be set anymore.					

Operation **IUMLConnector::ConnectorType**

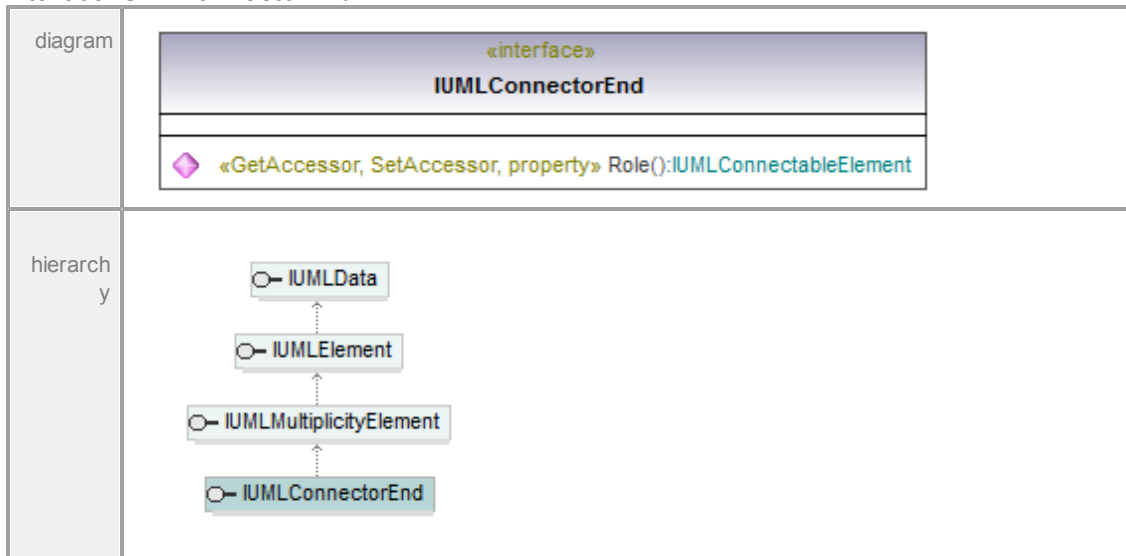
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLAssociation			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLConnectorEnd

Interface **IUMLConnectorEnd**



Operation **IUMLConnectorEnd::Role**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLConnectableElement			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLConstraint

Interface **IUMLConstraint**

diag ram	«interface» IUMLConstraint InsertConstrainedElementAt(in nIdx:int, in ipVal:IUMLElement):void EraseConstrainedElementAt(in nIdx:int):void SetNewSpecification(in strKind:string):IUMLValueSpecification SetNewSpecificationLiteralString(in strNewVal:string):IUMLLiteralString SetNewSpecificationInstanceValue(in ipInstance:IUMLInstanceSpecification):IUMLInstanceValue «GetAccessor, property» Context():IUMLNamespace «GetAccessor, property» ConstrainedElements():IUMLDataList «GetAccessor, property» Specification():IUMLValueSpecification «GetAccessor, property» OwningTransition():IUMLProtocolTransition «GetAccessor, property» OwningState():IUMLState	
hier arc hy	<pre> classDiagram IUMLData < -- IUMLElement IUMLElement < -- IUMLNamedElement IUMLNamedElement < -- IUMLPackageableElement IUMLPackageableElement < -- IUMLConstraint IUMLConstraint < -- IUMLInteractionConstraint IUMLConstraint < -- IUMLIntervalConstraint </pre>	
typ ed E me nts	Interface IUMLAction Interface IUMLBehavior Interface IUMLDataAll Interface IUMLNamespace Interface IUMLProtocolTransition	Operation InsertLocalPostConditionAt Operation InsertLocalPreConditionAt Operation InsertPostconditionAt Operation InsertPreconditionAt Operation InsertLocalPostConditionAt Operation InsertLocalPreConditionAt Operation InsertOwnedRuleAt Operation InsertPostconditionAt Operation InsertPreconditionAt Invariant Operation OwningConstraint Operation PostCondition PreCondition Operation SetNewInvariant Operation SetNewPostCondition Operation SetNewPreCondition Operation SetNewStateInvariant Operation SetNewTransitionGuard Operation StateInvariant TransitionGuard Operation InsertOwnedRuleAt Operation PostCondition PreCondition Operation SetNewPostCondition

	Interface IUMLState	Operation SetNew PreCondition
	Interface IUMLStateInvariant	Operation SetNew StateInvariant
	Interface IUMLTransition	Operation Invariant SetNew Invariant
		Operation SetNew TransitionGuard
	Interface IUMLValueSpecification	Operation OwningConstraint

Operation **IUMLConstraint::ConstrainedElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLElement .					

Operation **IUMLConstraint::Context**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLNamespace			

Operation **IUMLConstraint::EraseConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	void			

Operation **IUMLConstraint::InsertConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLElement			
	return	return	void			

Operation **IUMLConstraint::OwningState**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLState			

Operation **IUMLConstraint::OwningTransition**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLProtocolTransition			

Operation **IUMLConstraint::SetNewSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	in	string			
	return	return	IUMLValueSpecification			

Operation **IUMLConstraint::SetNewSpecificationInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default

r	ipInstance	in	IUMLInstanceSpecification
	return	return	IUMLInstanceValue

Operation **IUMLConstraint::SetNewSpecificationLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewValue	in	string			
	return	return	IUMLLiteralString			

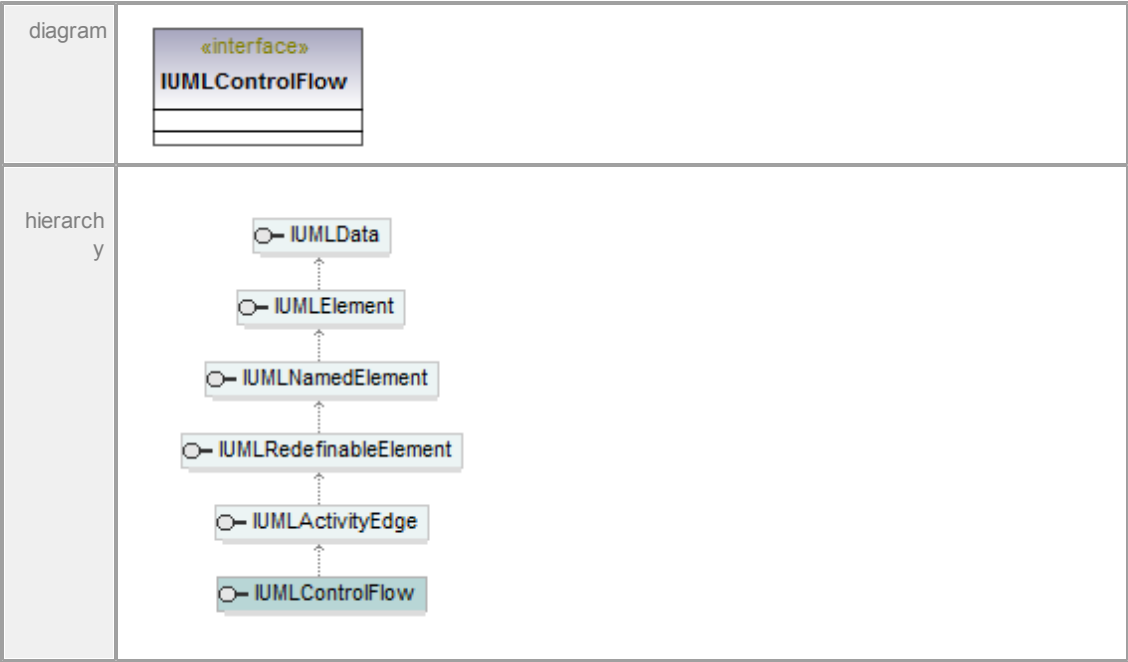
Operation **IUMLConstraint::Specification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLValueSpecification			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLControlFlow

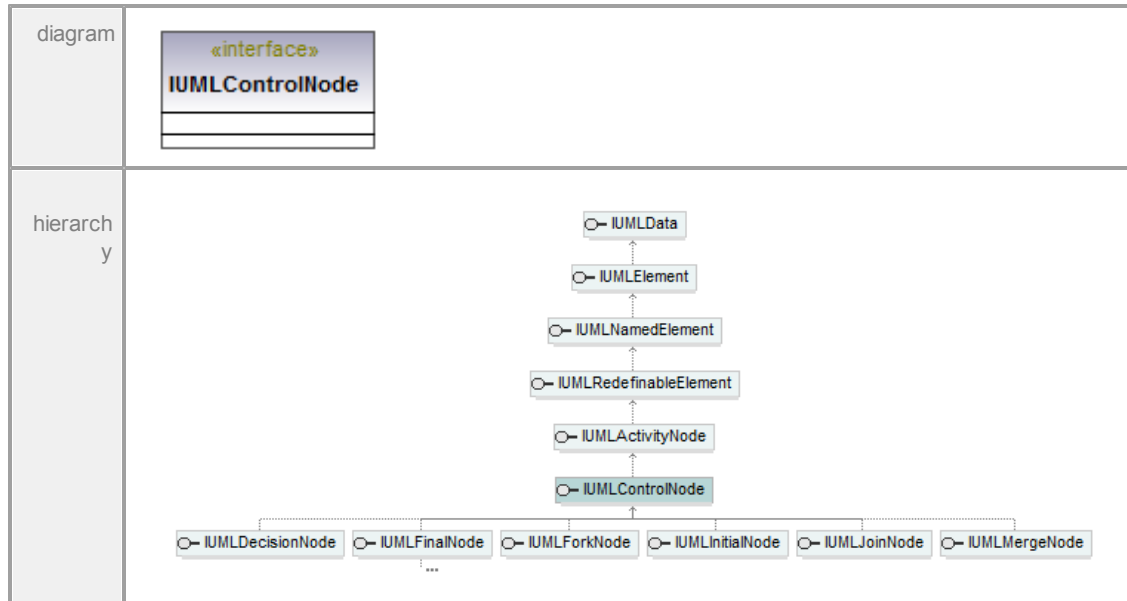
Interface **IUMLControlFlow**



UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLControlNode

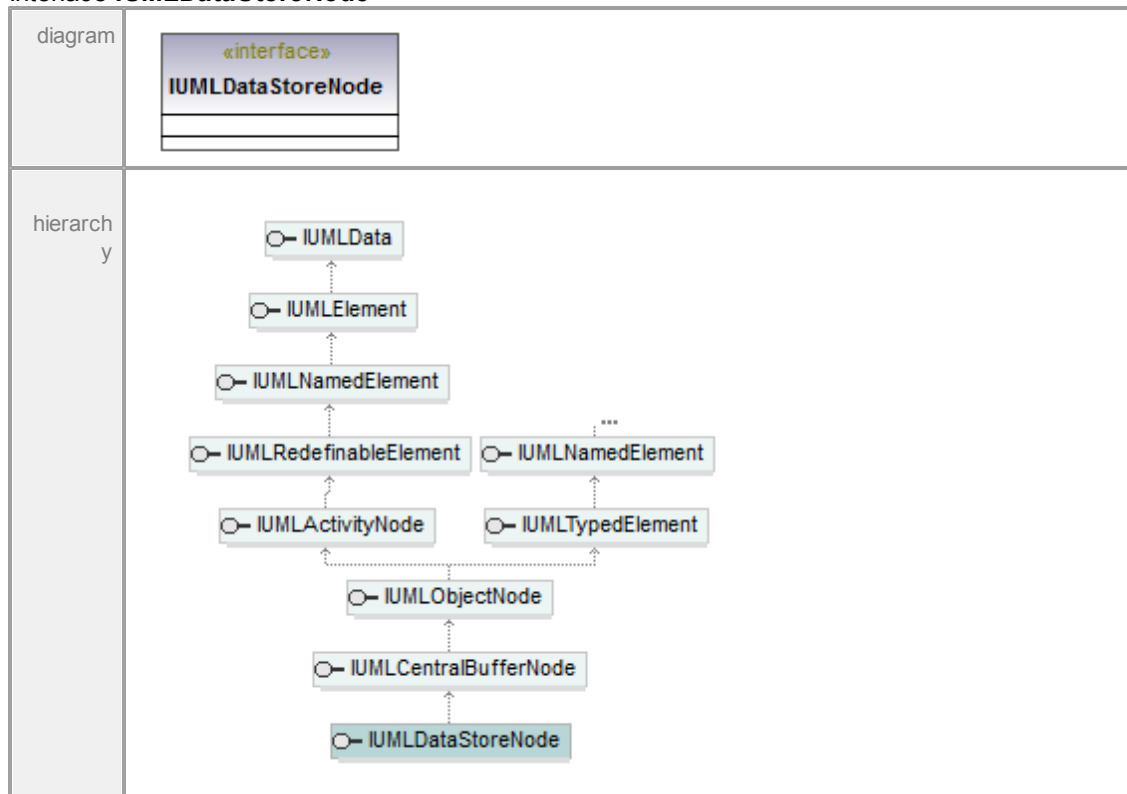
Interface IUMLControlNode

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLDataStoreNode

Interface IUMLDataStoreNode



UModelAPI - IUMLDataType

Interface **IUMLDataType**

diagram	«interface» IUMLDataType ◆ InsertOwnedOperationAt(in nidx:int):IUMLOperation ◆ InsertOwnedAttributeAt(in nidx:int):IUMLProperty ◆ «GetAccessor, property» OwnedAttributes():IUMLDataList ◆ «GetAccessor, property» OwnedOperations():IUMLDataList ◆ «GetAccessor, property» WasUsedForCodeSynchronization():bool
hierarch y	
typedEle ments	Interface IUMLDataAll Interface IUMLOperation Interface IUMLProperty Operation Datatype Operation Datatype Operation Datatype

Operation **IUMLDataType::InsertOwnedAttributeAt**

paramete r	name nidx return	direction in return	type int IUMLProperty	type modifier	multiplicity	default

Operation **IUMLDataType::InsertOwnedOperationAt**

paramete r	name nidx return	direction in return	type int IUMLOperation	type modifier	multiplicity	default

Operation **IUMLDataType::OwnedAttributes**

paramete r	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documen tation	A list of elements of type IUMLProperty .					

Operation **IUMLDataType::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLOperation .					

Operation **IUMLDataType::WasUsedForCodeSynchronization**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

UModelAPI - IUMLDecisionNode

Interface **IUMLDecisionNode**

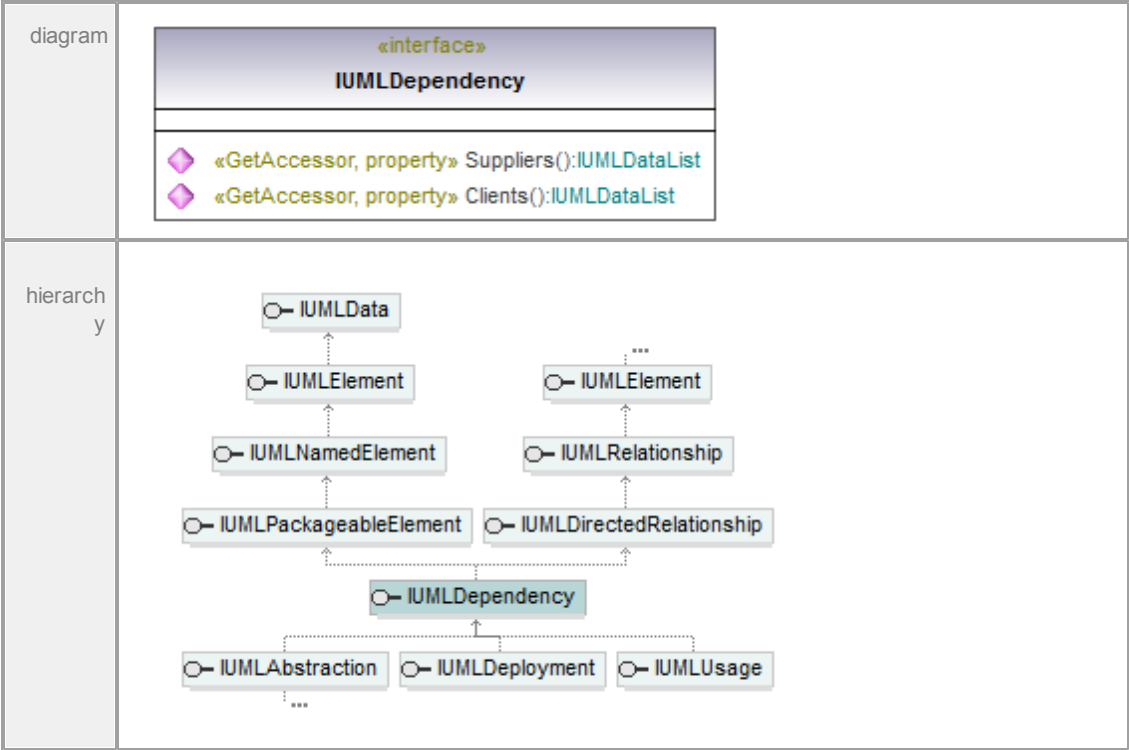


Operation **IUMLDecisionNode::DecisionInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLBehavior			

UModelAPI - IUMLDependency

Interface **IUMLDependency**



Operation **IUMLDependency::Clients**

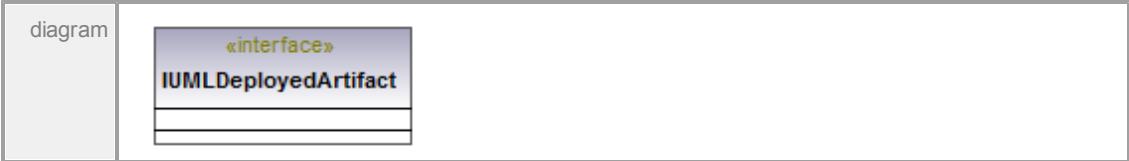
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLNamedElement .					

Operation **IUMLDependency::Suppliers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLNamedElement .					

UModelAPI - IUMLDeployedArtifact

Interface **IUMLDeployedArtifact**



hierarch y	<pre>graph BT IUMLElement --> IUMLNamedElement IUMLElement -.-> IUMLData IUMLNamedElement --> IUMLDeployedArtifact IUMLDeployedArtifact --> IUMLArtifact IUMLDeployedArtifact --> IUMLInstanceSpecification IUMLInstanceSpecification -.-> Ellipsis[...]</pre>	
typedEle ments	Interface IUMLDataAll Interface IUMLDeploymentTarget	Operation InsertDeploymentAt Operation InsertDeploymentAt

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLDeployment

Interface IUMLDeployment

diagram	<pre>classDiagram class IUMLDeployment { <<interface>> «GetAccessor, property» Location() IUMLDeploymentTarget }</pre>	
hierarch y	<pre>graph BT IUMLDeployment --> IUMLDependency IUMLDependency --> IUMLPackageableElement IUMLDependency --> IUMLDirectedRelationship IUMLPackageableElement --> IUMLNamedElement IUMLNamedElement --> IUMLElement IUMLDirectedRelationship --> IUMLRelationship IUMLRelationship -.-> IUMLElement IUMLElement -.-> Ellipsis[...]</pre>	
typedEle ments	Interface IUMLDataAll Interface IUMLDeploymentTarget	Operation InsertDeploymentAt Operation InsertDeploymentAt

Operation IUMLDeployment::Location

paramete	name	direction	type	type modifier	multiplicity	default

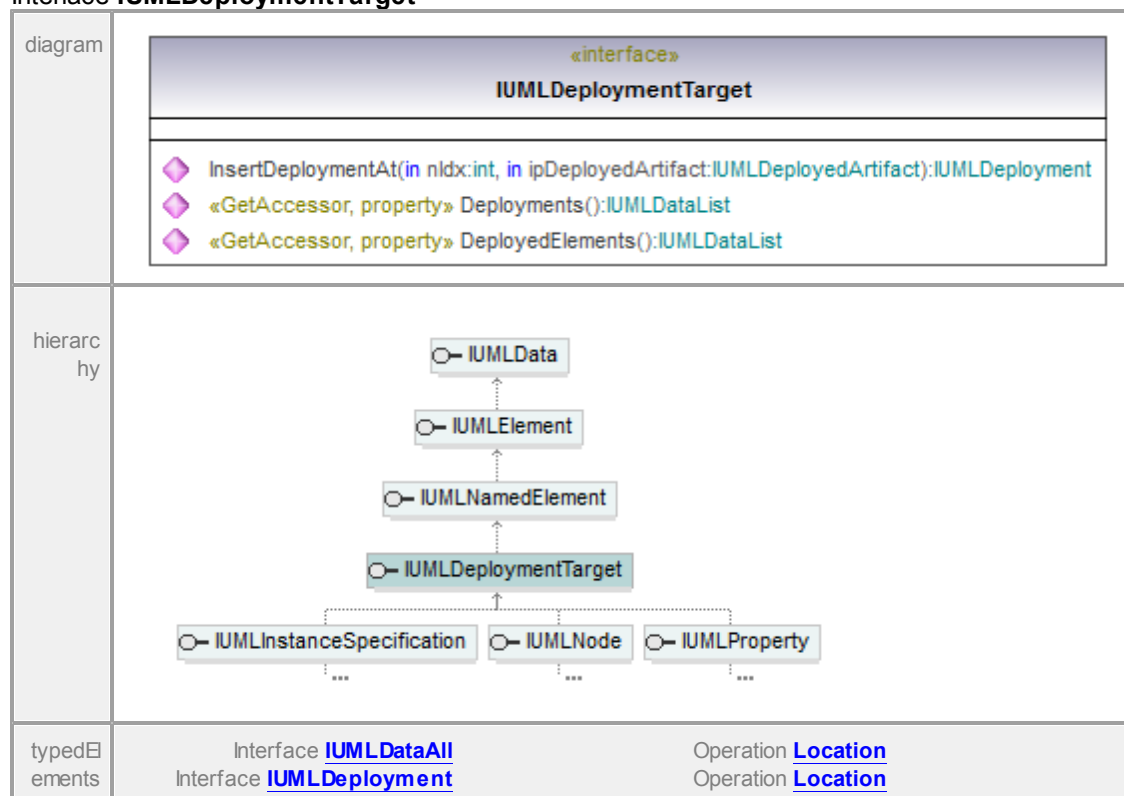
r	return	return	IUMLDeploymentTarget

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLDeploymentTarget

Interface IUMLDeploymentTarget



Operation IUMLDeploymentTarget::DeployedElements

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documen tation	A list of elements of type IUMLPackageableElement .					

Operation IUMLDeploymentTarget::Deployments

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documen tation	A list of elements of type IUMLDeployment .					

Operation IUMLDeploymentTarget::InsertDeploymentAt

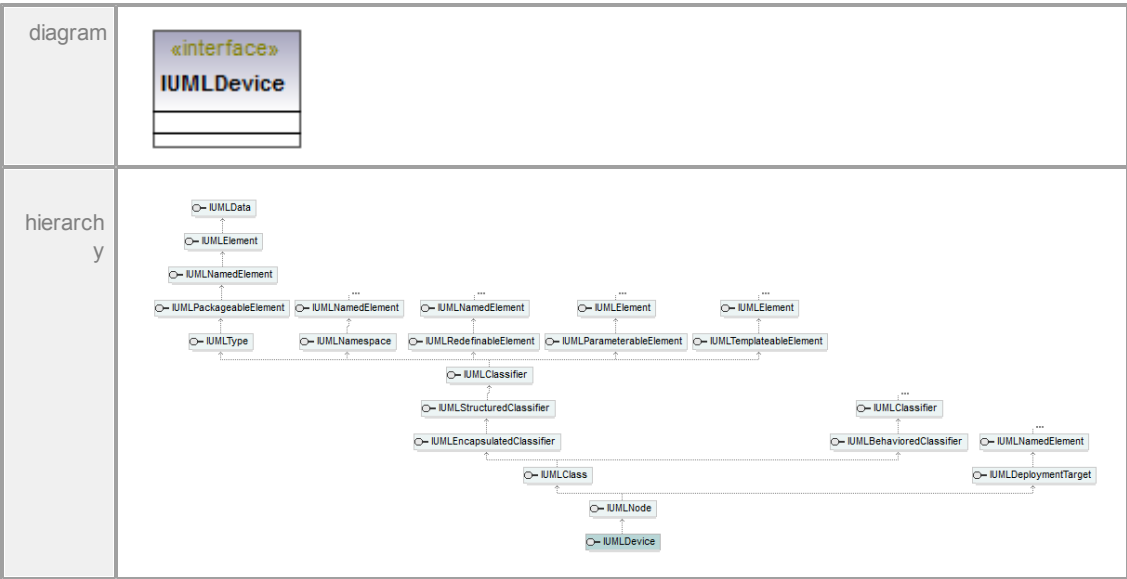
paramete r	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			

	ipDeployedArti in fact return return	IUMLDeployed Artifact IUMLDeployme nt

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLDevice

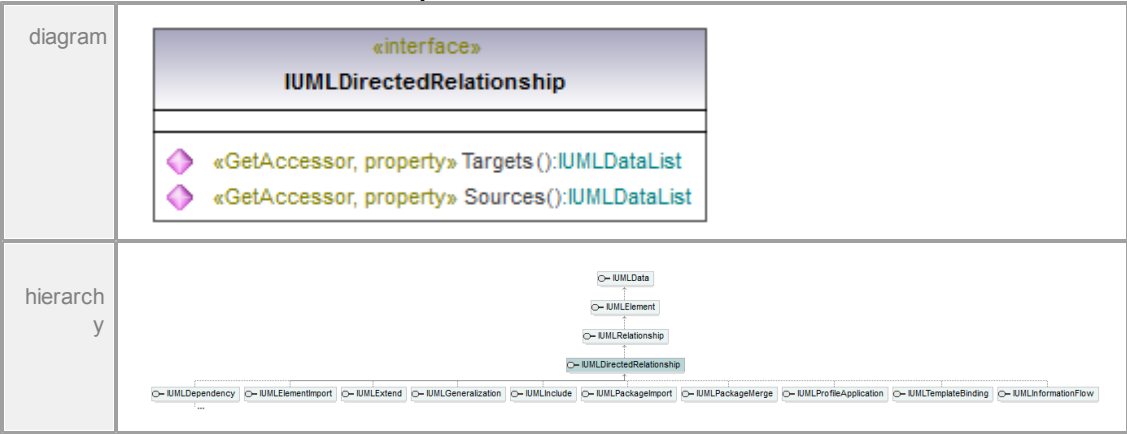
Interface **IUMLDevice**



UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLDirectedRelationship

Interface **IUMLDirectedRelationship**



Operation **IUMLDirectedRelationship::Sources**

paramete	name	direction	type	type modifier	multiplicity	default

r	return	return	IUMLDataList
documentation	A list of elements of type IUMLElement .		

Operation **IUMLDirectedRelationship::Targets**

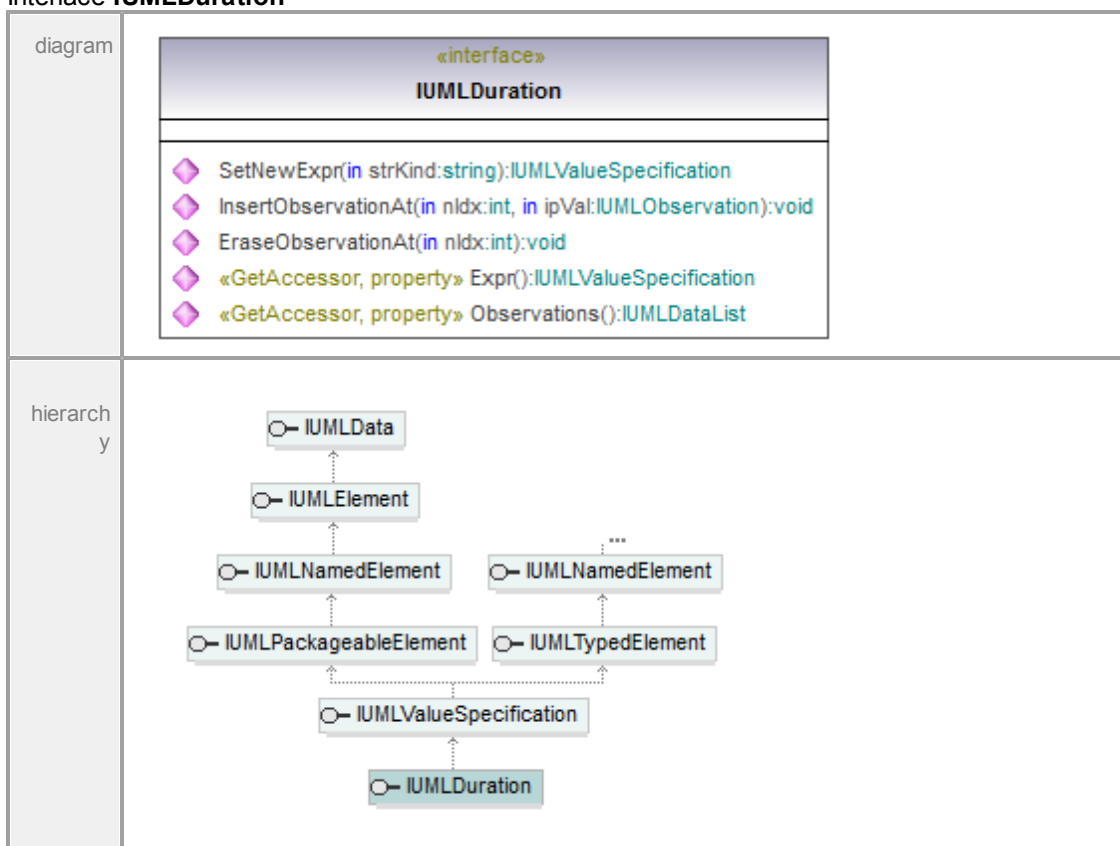
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLElement .					

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLDuration

Interface **IUMLDuration**



Operation **IUMLDuration::EraseObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	ndx	in	int			
	return	return	void			

Operation **IUMLDuration::Expr**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLValueSpecification			

Operation **IUMLDuration::InsertObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipVal	in	IUMLObservation			
	return	return	void			

Operation **IUMLDuration::Observations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLObservation .					

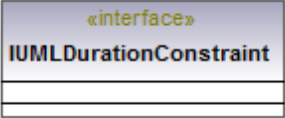
Operation **IUMLDuration::SetNewExpr**

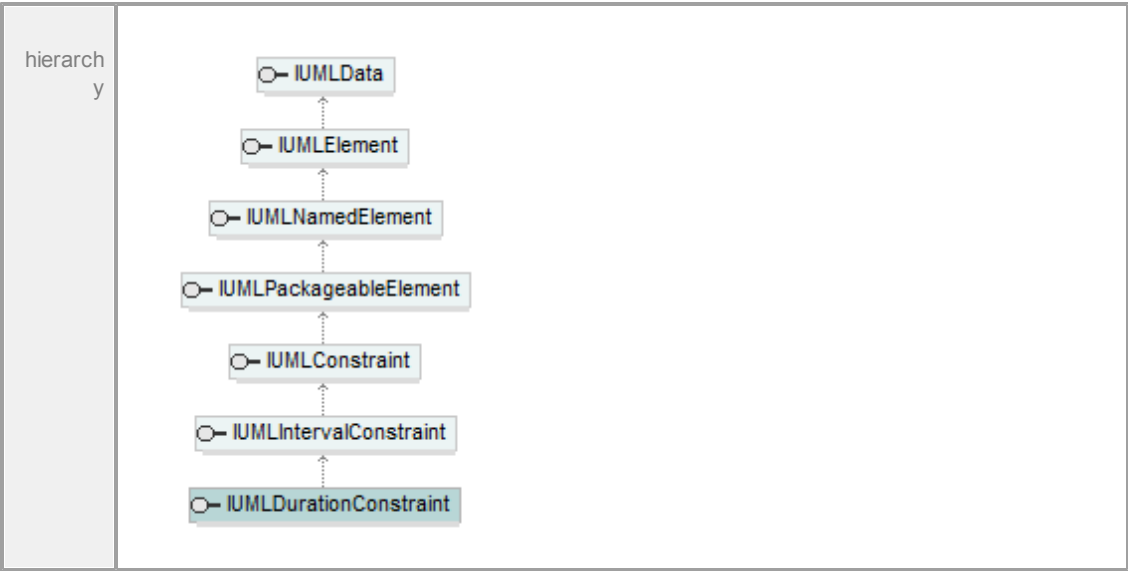
parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	return	return	IUMLValueSpecification			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLDurationConstraint

Interface **IUMLDurationConstraint**

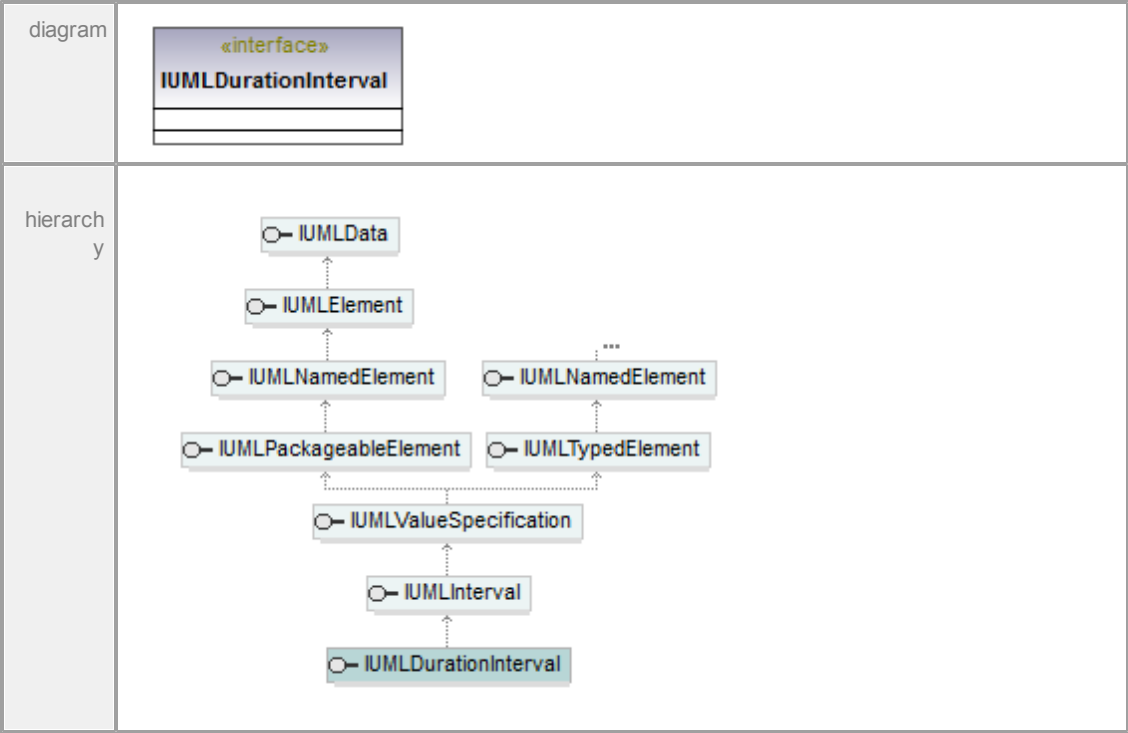
diagram	



UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLDurationInterval

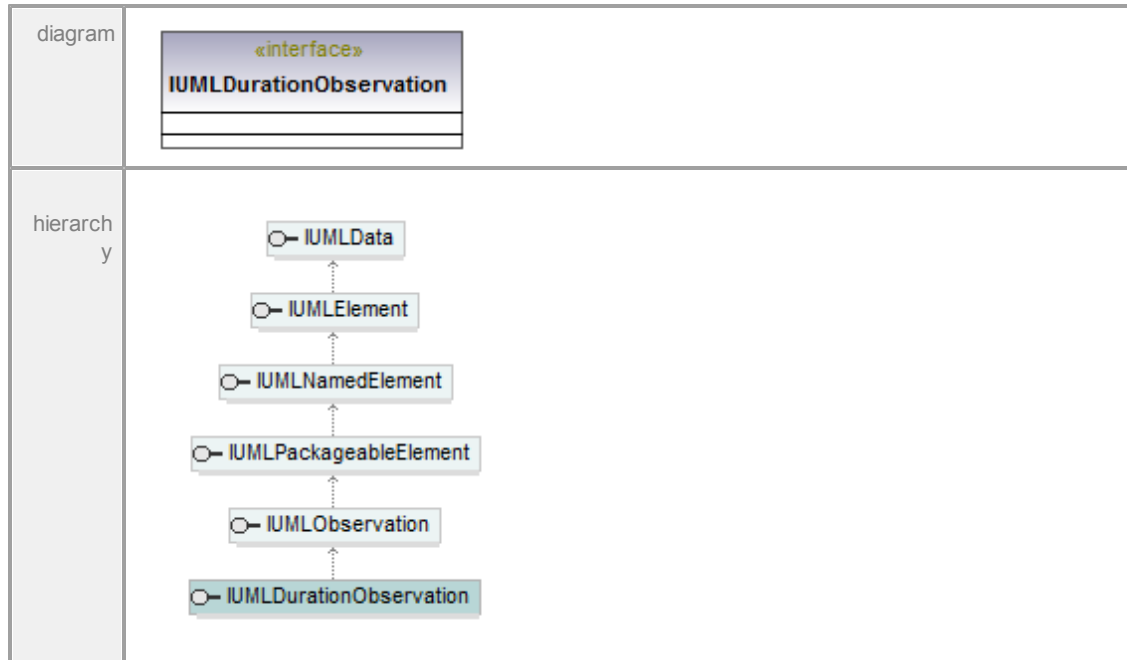
Interface IUMLDurationInterval



UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLDurationObservation

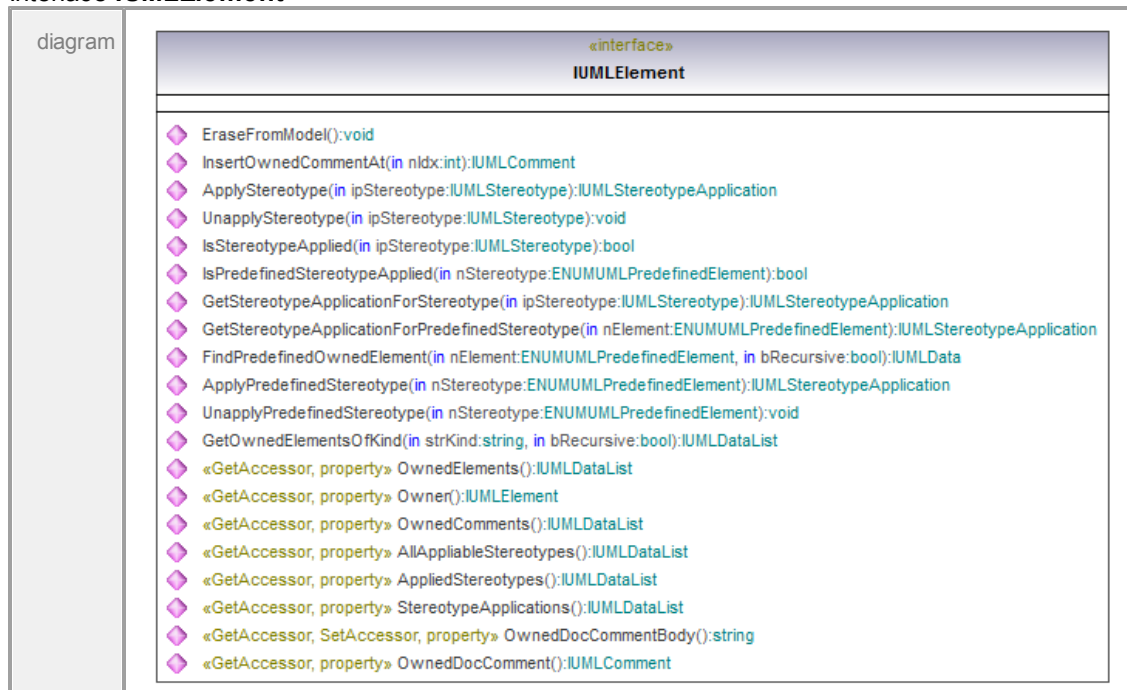
Interface IUMLDurationObservation

UML documentation generated by UModel UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLElement

Interface IUMLElement



hierarchy	<pre> graph TD UMLData --> UMLComment UMLData --> UMLExceptionHandler UMLData --> UMLHyperlink UMLData --> UMLMultiplicityElement UMLData --> UMLNamedElement UMLData --> UMLParameterableElement UMLData --> UMLRelationship UMLData --> UMLSet UMLData --> UMLTemplateableElement UMLData --> UMLTemplateParameter UMLData --> UMLTemplateParameterSubstitution UMLData --> UMLTemplateSignature UMLData --> UMLCommentFixedHyperlink </pre>	
typedElements	Interface <u>IGenerateSequenceDiagramDiagramOwner</u> Interface <u>IUMLElement</u> Interface <u>IUMLGuiDiagram</u> Interface <u>IUMLGuiLink</u> Interface <u>IUMLGuiNodeLink</u> Interface <u>IUMLGuiTextLabel</u> Interface <u>IUMLPackage</u> Interface <u>IUMLStereotype</u>	Operation <u>InsertAnnotatedElementAtOwningElement</u> Operation <u>InsertConstrainedElementAtAppliedElement</u> Operation <u>InsertAnnotatedElementAtInsertConstrainedElementAtInsertPackagedElementRelationshipAtIsElementVisibleLinkedOwnerOwnerOwningElementSetElementVisibleTextLabelElementOwner</u> Operation <u>LinkedOwner</u> Operation <u>Element</u> Operation <u>IsElementVisibleSetElementVisibleTextLabelElementInsertPackagedElementRelationshipAtAppliedElement</u>

Operation IUMLElement::AllApplicableStereotypes

parameter	name	direction	type	type modifier	multiplicity	default
return	return	return	IUMLDataList			
documentation	A list of elements of type IUMLStereotype .					

Operation **IUMLElement::AppliedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
return	return	return	IUMLDataList			
documentation	A list of elements of type IUMLStereoType .					

Operation IUMLElement::ApplyPredefinedStereotype

parameter	name	direction	type	type modifier	multiplicity	default
r	nStereotype	in	<u>ENUMUMLPred</u>			
	return	return	<u>efinedElement</u> <u>IUMLStereotyp</u> <u>eApplication</u>			

Operation IUMLElement::ApplyStereotype

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	in	<u>IUMLStereotype</u>			
	return	return	<u>IUMLStereotype</u>			

	eApplication					

Operation **IUMLElement::EraseFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	void			
documentation	Use this function to erase the element from the model and all diagrams. Use IUMLGuiDiagram::EraseFromDiagram to erase from diagram only.					

Operation **IUMLElement::FindPredefinedOwnedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	in	ENUMUMLPredefinedElement			
	bRecursive	in	bool			
	return	return	IUMLData			

Operation **IUMLElement::GetOwnedElementsOfKind**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	in	string			
	bRecursive	in	bool			
	return	return	IUMLDataList			
documentation	get all owned elements of the specified kind (<i>strKind</i>)					

Operation **IUMLElement::GetStereotypeApplicationForPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	nElement	in	ENUMUMLPredefinedElement			
	return	return	IUMLStereotypeApplication			

Operation **IUMLElement::GetStereotypeApplicationForStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
	ipStereotype	in	IUMLStereotype			
	return	return	IUMLStereotypeApplication			

Operation **IUMLElement::InsertOwnedCommentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	IUMLComment			

Operation **IUMLElement::IsPredefinedStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
	nStereotype	in	ENUMUMLPredefinedElement			
	return	return	bool			

Operation **IUMLElement::IsStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
r	isStereotype	in	IUMLStereotype			
	return	return	bool			

Operation **IUMLElement::OwnedComments**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLComment .					

Operation **IUMLElement::OwnedDocComment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLComment			

Operation **IUMLElement::OwnedDocCommentBody**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLElement::OwnedElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLElement .					

Operation **IUMLElement::Owner**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLElement			

Operation **IUMLElement::StereotypeApplications**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLStereotypeApplication .					

Operation **IUMLElement::UnapplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
r	nStereotype	in	ENUMUMLPredefinedElement			
	return	return	void			

Operation **IUMLElement::UnapplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default

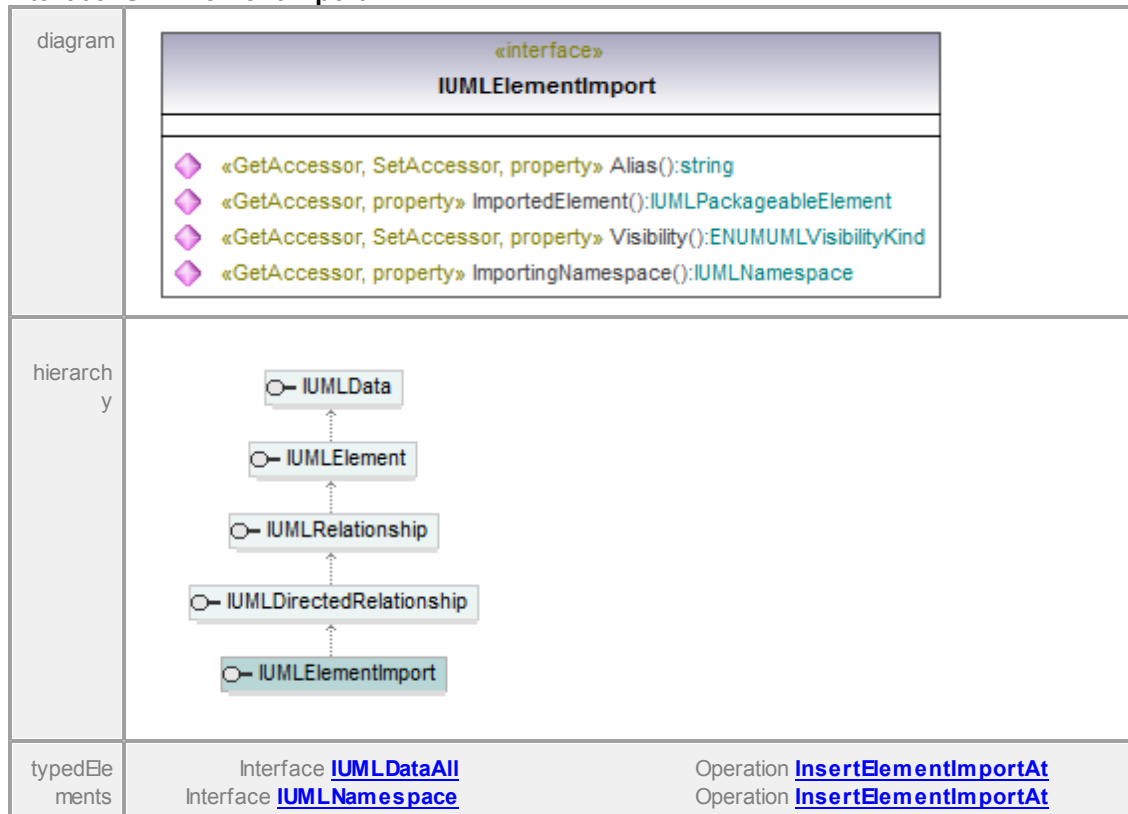
r	ipStereotype	in	IUMLStereotype
	return	return	void

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLElementImport

Interface IUMLElementImport



Operation IUMLElementImport::Alias

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation IUMLElementImport::ImportedElement

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLPackageableElement			

Operation IUMLElementImport::ImportingNamespace

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUM_NAMESPACE			

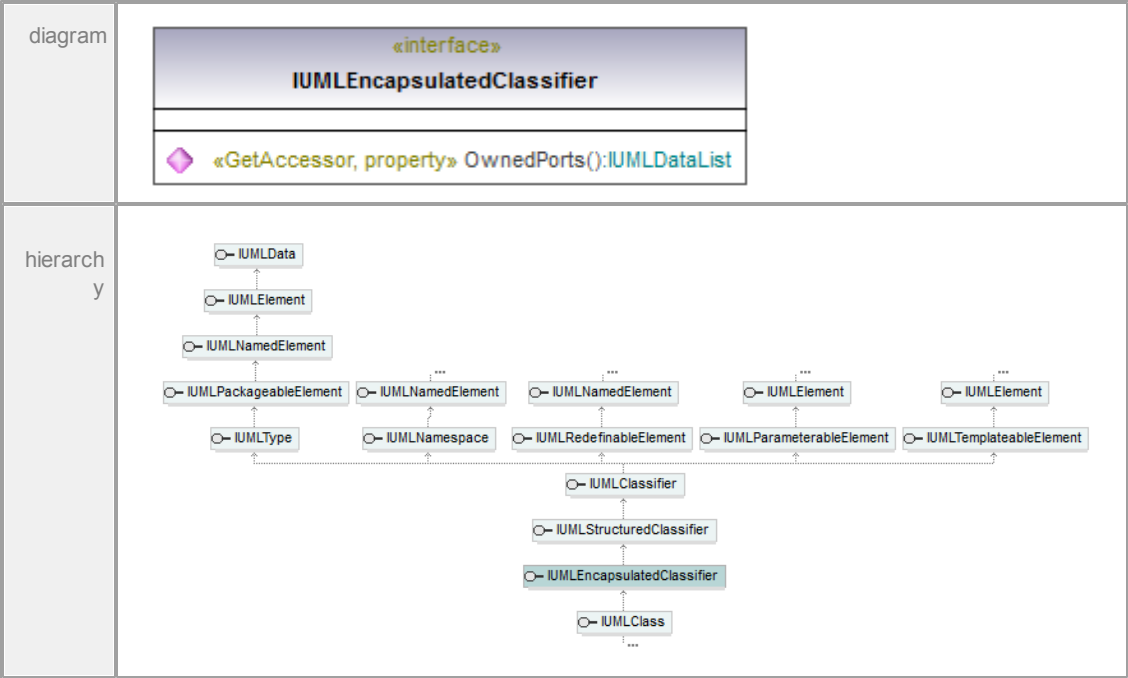
Operation IUMLElementImport::Visibility

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ENUMUMLVisibilityKind			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLEncapsulatedClassifier

Interface IUMLEncapsulatedClassifier



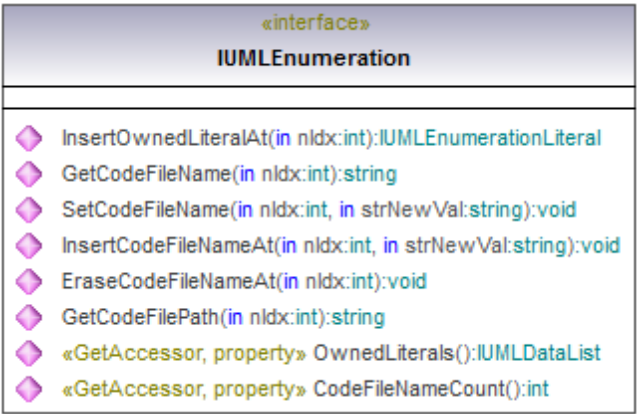
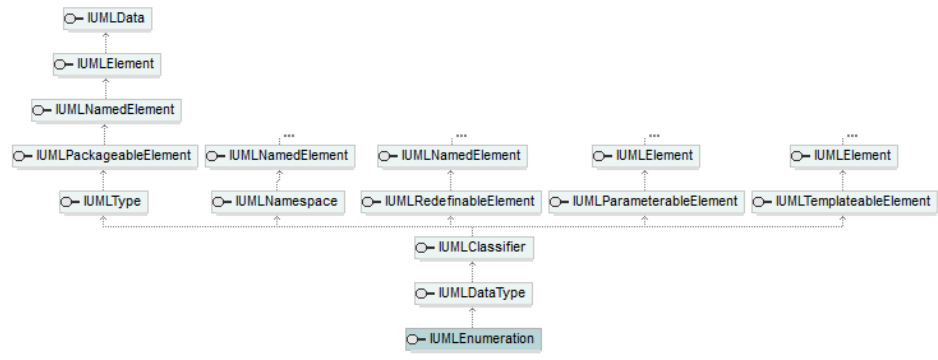
Operation IUMLEncapsulatedClassifier::OwnedPorts

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLEnumeration

Interface **IUMLEnumeration**

diagram		
hierarchy		
typedElements	Interface IUMLDataAll Interface IUMLEnumerationLiteral	Operation Enumeration Operation Enumeration

Operation **IUMLEnumeration::CodeFileNameCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLEnumeration::EraseCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	return	return	void			

Operation **IUMLEnumeration::GetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	return	return	string			

Operation **IUMLEnumeration::GetCodeFilePath**

parameter	name	direction	type	type modifier	multiplicity	default

r	nidx return	in return	int string	
documenta tion	get the full code file path			

Operation **IUMLEnumeration::InsertCodeFileNameAt**

paramete r	name nidx strNewVal return	direction in in return	type int string void	type modifier	multiplicity	default

Operation **IUMLEnumeration::InsertOwnedLiteralAt**

paramete r	name nidx return	direction in return	type int IUMLEnumerationLiteral	type modifier	multiplicity	default

Operation **IUMLEnumeration::OwnedLiterals**

paramete r	name return	direction return	type IUMLEDataList	type modifier	multiplicity	default
documenta tion	A list of elements of type IUMLEnumerationLiteral .					

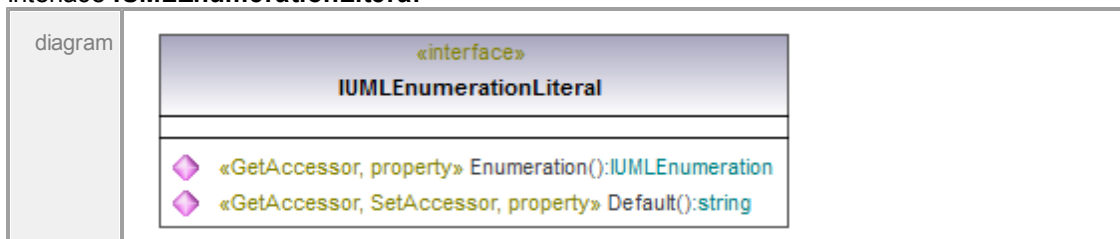
Operation **IUMLEnumeration::SetCodeFileName**

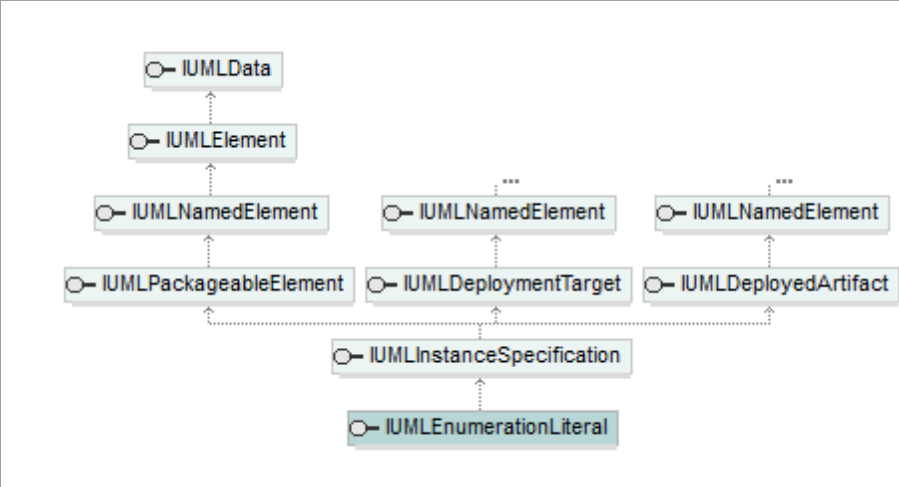
paramete r	name nidx strNewVal return	direction in in return	type int string void	type modifier	multiplicity	default

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLEnumerationLiteral

Interface **IUMLEnumerationLiteral**

hierarchy	 <pre>graph BT IUMLEnumerationLiteral --> IUMLInstanceSpecification IUMLInstanceSpecification --> IUMLPackageableElement IUMLInstanceSpecification --> IUMLDeploymentTarget IUMLInstanceSpecification --> IUMLDeployedArtifact IUMLPackageableElement --> IUMLNamedElement IUMLDeploymentTarget --> IUMLNamedElement IUMLDeployedArtifact --> IUMLNamedElement IUMLNamedElement --> IUMLElement IUMLElement --> IUMLData</pre>		
typedElements	Interface IUMLDataAll Interface IUMLEnumeration	Operation InsertOwnedLiteralAt Operation InsertOwnedLiteralAt	

Operation **IUMLEnumerationLiteral::Default**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

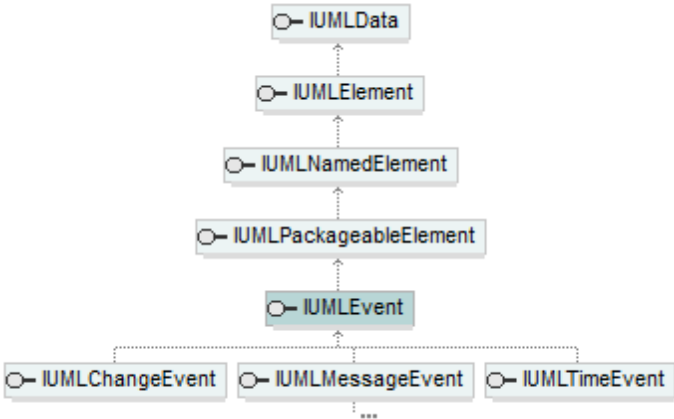
Operation **IUMLEnumerationLiteral::Enumeration**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLEnumeration			

UModelAPI - IUMLEvent

Interface **IUMLEvent**

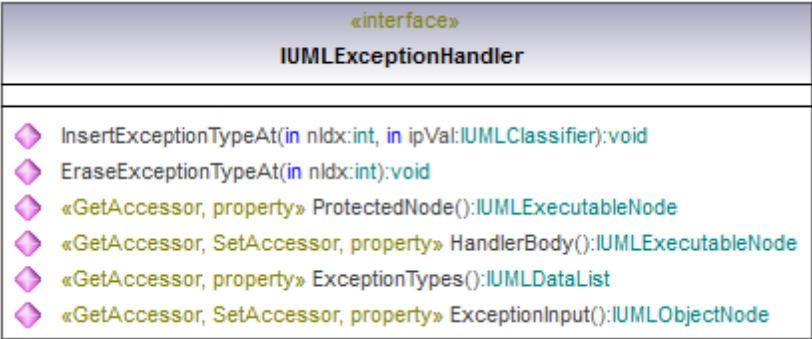
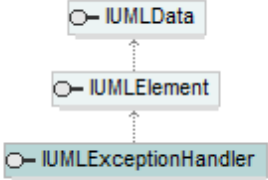
diagram	 <pre>classDiagram class IUMLEvent { <<interface>> }</pre>

hierarch y		
typedEle ments	Interface IUMLDataAll Interface IUMLOccurrenceSpecification Interface IUMLTrigger	Operation Event OccurringEvent Operation OccurringEvent Operation Event

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLEExceptionHandler

Interface **IUMLEExceptionHandler**

diagram		
hierarch y		
typedEle ments	Interface IUMLDataAll Interface IUMLExecutableNode	Operation InsertHandlerAt Operation InsertHandlerAt

Operation **IUMLEExceptionHandler::EraseExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	void			

Operation **IUMLEExceptionHandler::ExceptionInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLObjectNode			

Operation **IUMLEExceptionHandler::ExceptionTypes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLClassifier .					

Operation **IUMLEExceptionHandler::HandlerBody**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLExecutableNode			

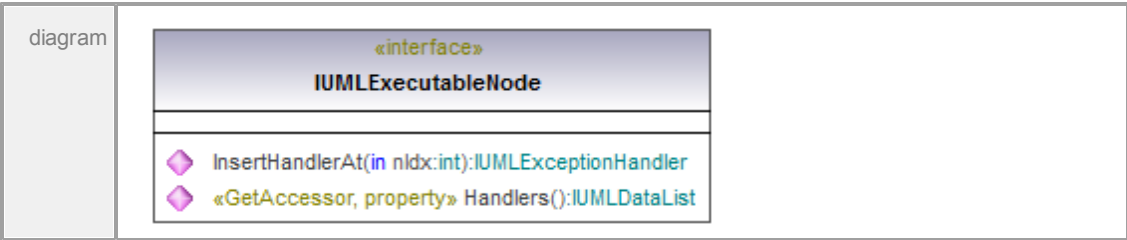
Operation **IUMLEExceptionHandler::InsertExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	ipVal	in	IUMLClassifier			
	return	return	void			

Operation **IUMLEExceptionHandler::ProtectedNode**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLExecutableNode			

Interface **IUMLExecutableNode**



hierarchy	<pre>graph BT; IUMLAction --> IUMLExecutableNode; IUMLExecutableNode --> IUMLActivityNode; IUMLActivityNode --> IUMLRedefinableElement; IUMLRedefinableElement --> IUMLNamedElement; IUMLNamedElement --> IUMLElement; IUMLElement --> IUMLData</pre>		
typedElements	Interface IUMLDataAll Interface IUMLExceptionHandler	Operation HandlerBody ProtectedNode Operation HandlerBody ProtectedNode	

Operation **IUMLExecutableNode::Handlers**

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documentation	A list of elements of type IUMLExceptionHandler .					

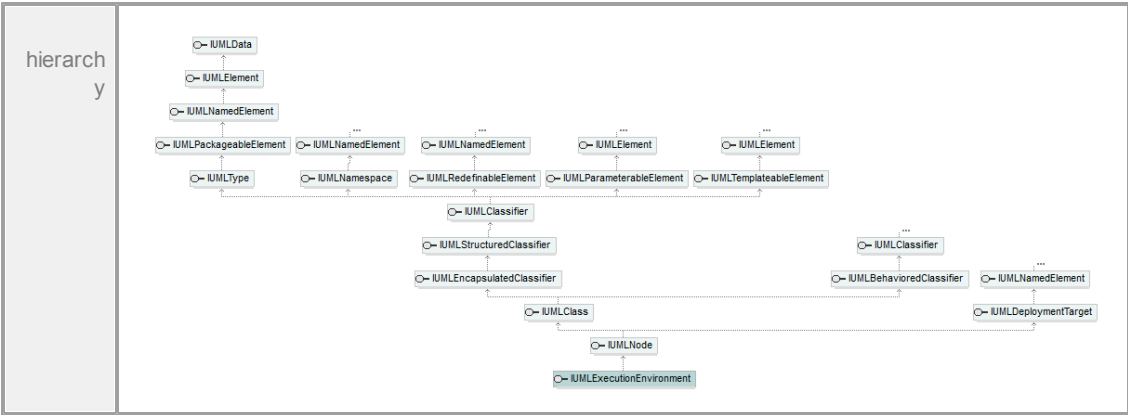
Operation **IUMLExecutableNode::InsertHandlerAt**

parameter	name ndx return	direction in return	type int IUMLExceptionHandler	type modifier	multiplicity	default

UModelAPI - IUMLExecutionEnvironment

Interface **IUMLExecutionEnvironment**

diagram	<pre>classDiagram class IUMLExecutionEnvironment { <<interface>> }</pre>



UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLExecutionSpecification

Interface **IUMLExecutionSpecification**

diagram	<pre>classDiagram class IUMLExecutionSpecification { «GetAccessor, SetAccessor, property» Start():IUMLOccurrenceSpecification «GetAccessor, SetAccessor, property» Finish():IUMLOccurrenceSpecification }</pre>	
hierarchy	<pre>classDiagram IUMLExecutionSpecification < .. IUMLActionExecutionSpecification IUMLExecutionSpecification < .. IUMLBehaviorExecutionSpecification IUMLExecutionSpecification .. IUMLInteractionFragment IUMLInteractionFragment .. IUMLNamedElement IUMLNamedElement .. IUMLElement IUMLElement .. IUMLData</pre>	
typedElements	Interface IUMLDataAll	Operation ExecutionSpecificationFinish
	Interface IUMLOccurrenceSpecification	Operation ExecutionSpecificationStart

Operation **IUMLExecutionSpecification::Finish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLOccurrenceSpecification			

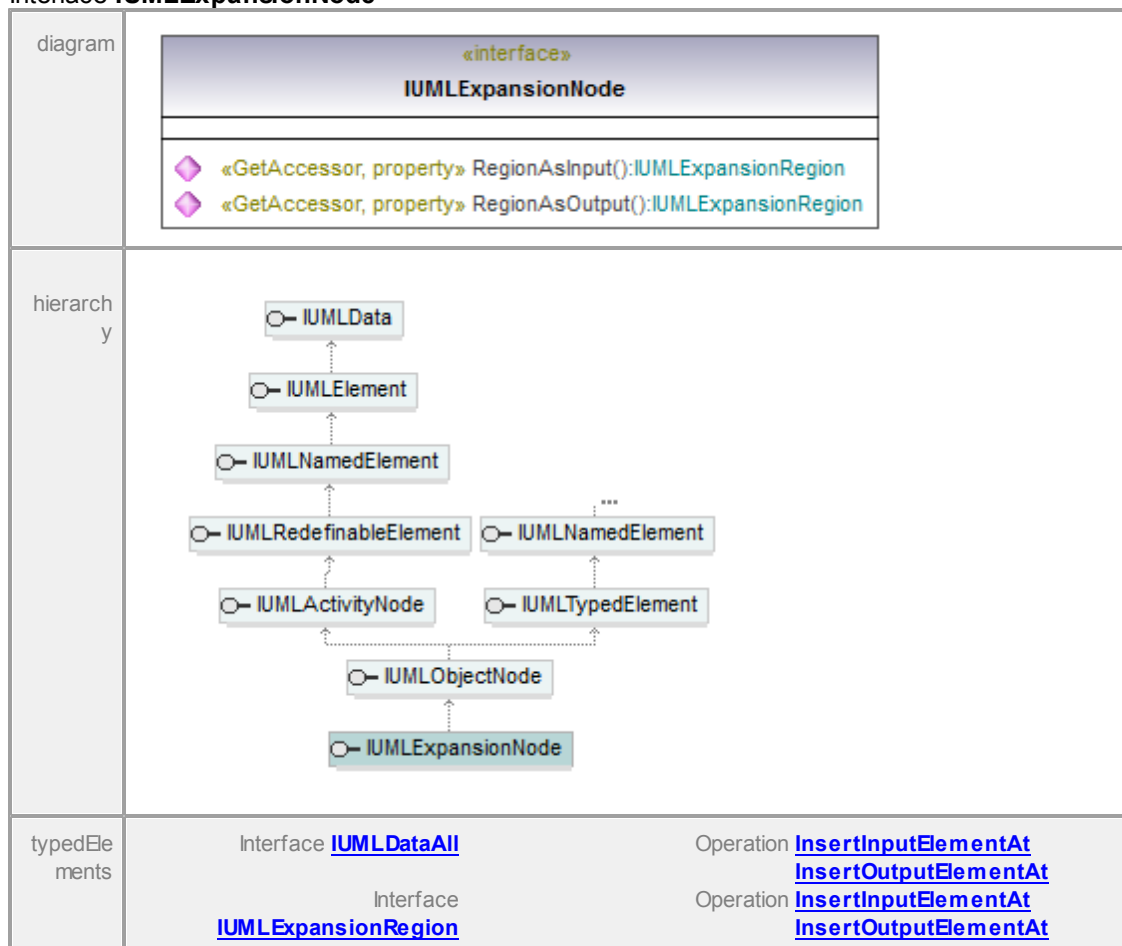
Operation **IUMLExecutionSpecification::Start**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLOccurrenceSpecification			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLExpansionNode

Interface **IUMLExpansionNode**Operation **IUMLExpansionNode::RegionAsInput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLExpansionRegion			

Operation **IUMLExpansionNode::RegionAsOutput**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLExpansionRegion			

	Region

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLExpansionRegion

Interface **IUMLExpansionRegion**

diagram		
typedElements	Interface IUMLDataAll	Operation RegionAs Input RegionAs Output
	Interface IUMLExpansionNode	Operation RegionAs Input RegionAs Output

Operation **IUMLExpansionRegion::EraseInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	return	return	void			

Operation **IUMLExpansionRegion::EraseOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default

r	nidx return	in return	int void

Operation **IUMLExpansionRegion::InputElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEDataList			
documentation	A list of elements of type IUMLExpansionNode .					

Operation **IUMLExpansionRegion::InsertInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx ipNode	in in	int IUMLExpansionNode			
	return	return	void			

Operation **IUMLExpansionRegion::InsertOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx ipNode	in in	int IUMLExpansionNode			
	return	return	void			

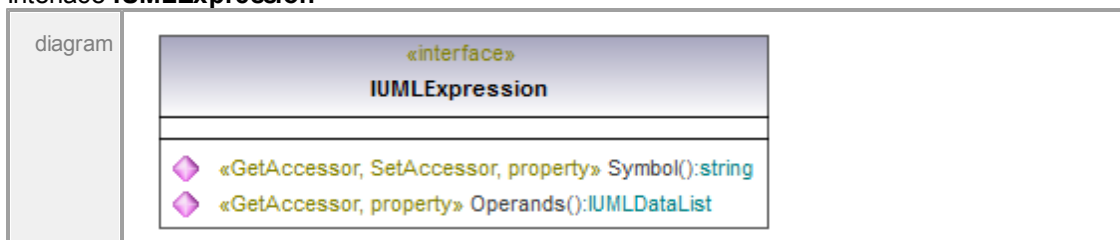
Operation **IUMLExpansionRegion::Mode**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLExpansionKind			

Operation **IUMLExpansionRegion::OutputElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEDataList			
documentation	A list of elements of type IUMLExpansionNode .					

UModelAPI - IUMLExpansion

Interface **IUMLExpansion**

hierarchy	<pre>graph BT IUMLExpression -- > IUMLValueSpecification IUMLValueSpecification -- > IUMLPackageableElement IUMLValueSpecification -- > IUMLTypedElement IUMLPackageableElement -- > IUMLNamedElement IUMLTypedElement -- > IUMLNamedElement IUMLNamedElement -- > IUMLElement IUMLElement -- > IUMLData</pre>
typedElements	Interface IUMLDataAll Interface IUMLValueSpecification Operation Expression Operation Expression

Operation **IUMLExpression::Operands**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLValueSpecification .					

Operation **IUMLExpression::Symbol**

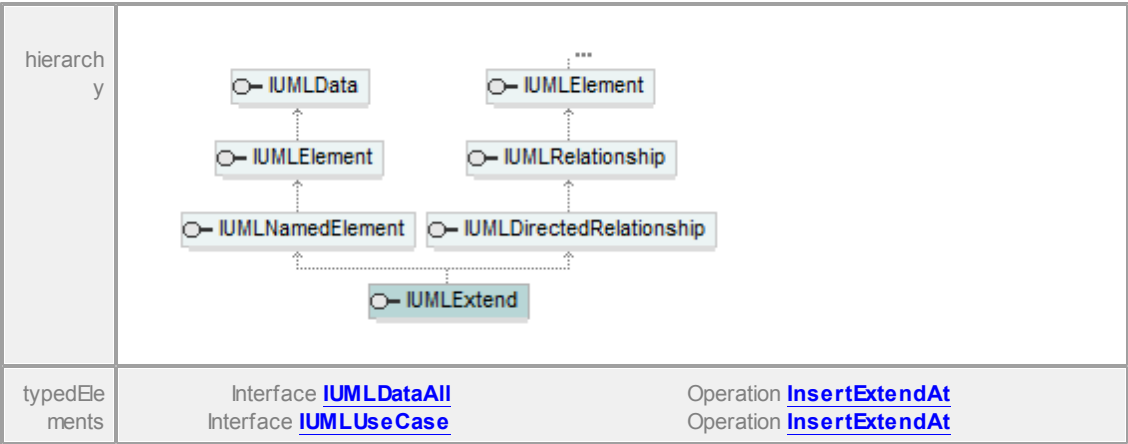
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLExtend

Interface **IUMLExtend**

diagram	<pre>classDiagram class IUMLExtend { <<interface>> InsertExtensionLocationAt(in nIdx:int, in pExtensionLocation:IUMLExtensionPoint):void EraseExtensionLocationAt(in nIdx:int):void «GetAccessor, property» Extension():IUMLUseCase «GetAccessor, property» ExtendedCase():IUMLUseCase «GetAccessor, property» ExtensionLocations():IUMLDataList }</pre>



Operation **IUMLExtend::EraseExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	return	return	void			

Operation **IUMLExtend::ExtendedCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLUseCase			

Operation **IUMLExtend::Extension**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLUseCase			

Operation **IUMLExtend::ExtensionLocations**

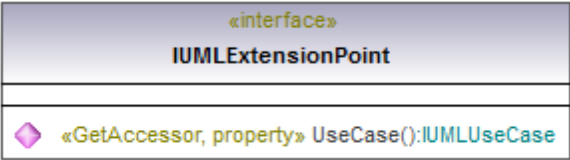
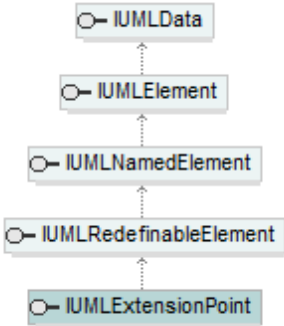
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLExtensionPoint .					

Operation **IUMLExtend::InsertExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	pExtensionLocation	in	IUMLExtensionPoint			
	return	return	void			

UModelAPI - IUMLExtensionPoint

Interface **IUMLExtensionPoint**

diagram			
hierarch y			
typedEle ments	Interface IUMLDataAll	Operation InsertExtensionLocationAt	Operation InsertExtensionPointAt
	Interface IUMLExtend	Operation InsertExtensionLocationAt	Operation InsertExtensionPointAt
	Interface IUMLUseCase	Operation InsertExtensionPointAt	

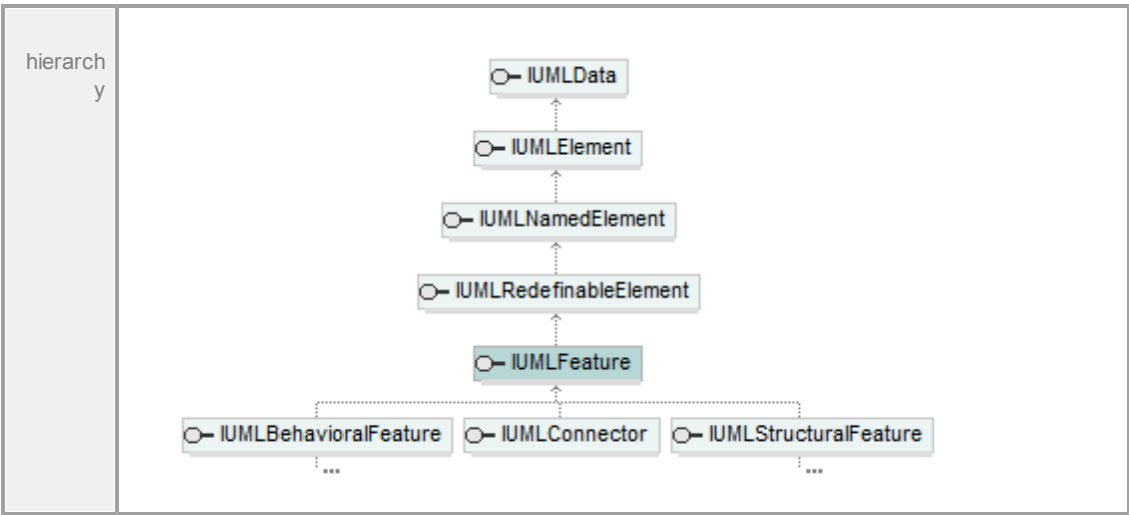
Operation **IUMLExtensionPoint::UseCase**

paramete r	name return	direction return	type IUMLUseCase	type modifier	multiplicity	default

UModelAPI - IUMLFeature

Interface **IUMLFeature**

diagram			



Operation **IUMLFeature::FeaturingClassifiers**

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documentation	A list of elements of type IUMLClassifier .					

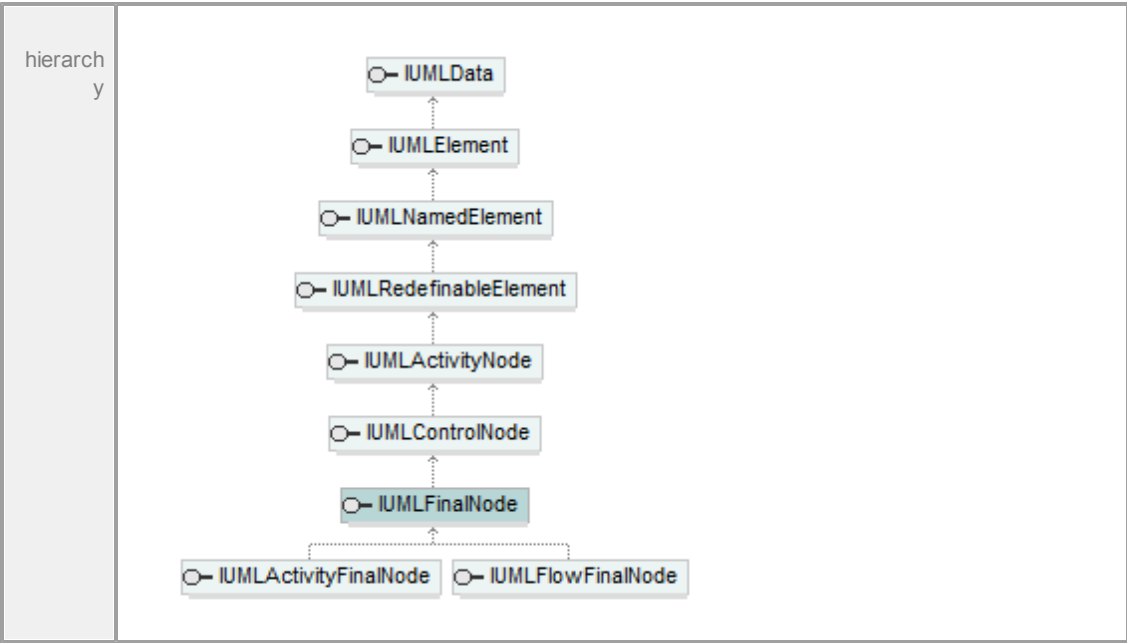
Operation **IUMLFeature::IsStatic**

parameter	name return	direction return	type bool	type modifier	multiplicity	default

UModelAPI - IUMLFinalNode

Interface **IUMLFinalNode**

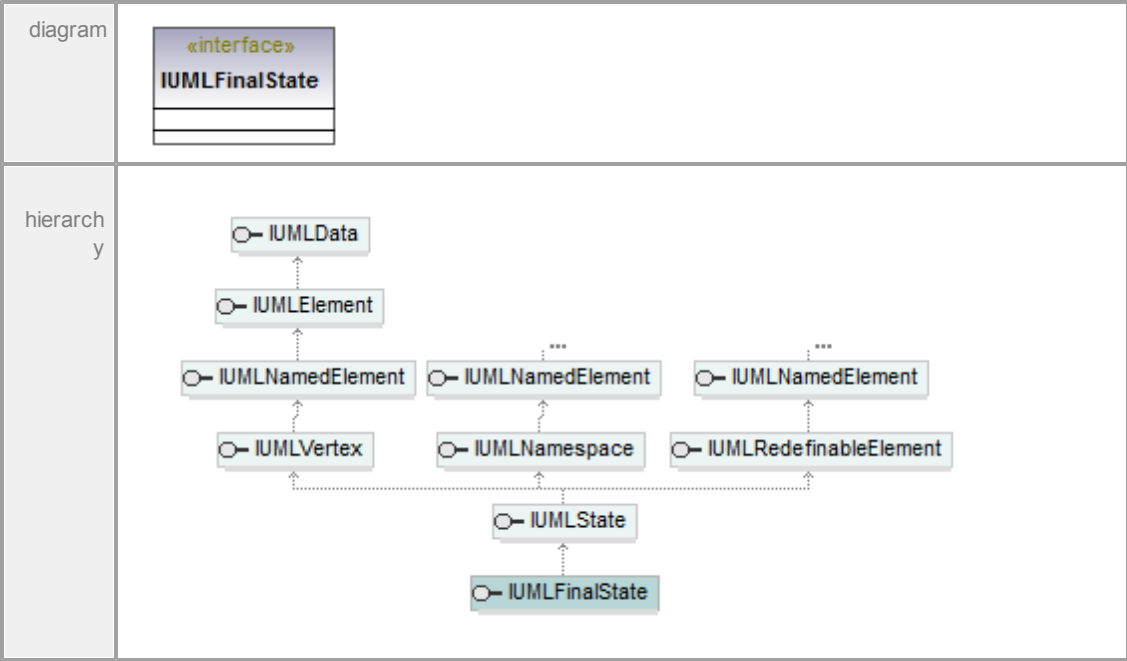




UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLFinalState

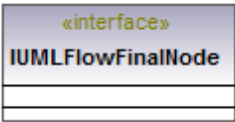
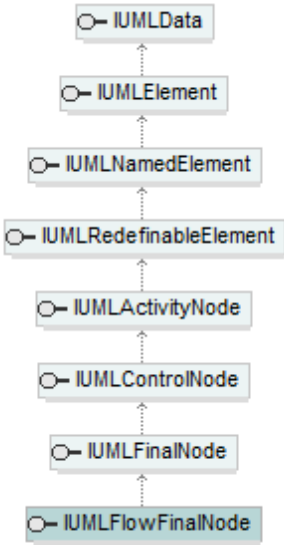
Interface IUMLFinalState



UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

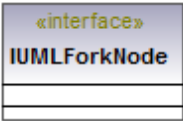
UModelAPI - IUMLFlowFinalNode

Interface IUMLFlowFinalNode

diagram	 The diagram shows the UML interface IUMLFlowFinalNode. It is represented as a rectangle with a light blue header containing the text «interface» in green. Below the header, the name IUMLFlowFinalNode is written in bold black text. There are two empty compartments below the name.
hierarchy	 The hierarchy diagram shows the inheritance structure of IUMLFlowFinalNode. It is a vertical stack of boxes connected by dashed arrows pointing upwards. The boxes, from bottom to top, are: IUMLFlowFinalNode, IUMLFinalNode, IUMLControlNode, IUMLActivityNode, IUMLRedefinableElement, IUMLNamedElement, IUMLElement, and IUMLData. Each box has a small circle icon to its left.

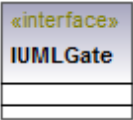
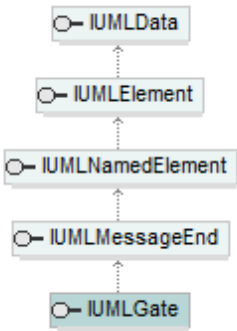
UModelAPI - IUMLForkNode

Interface IUMLForkNode

diagram	 The diagram shows the UML interface IUMLForkNode. It is represented as a rectangle with a light blue header containing the text «interface» in green. Below the header, the name IUMLForkNode is written in bold black text. There are two empty compartments below the name.

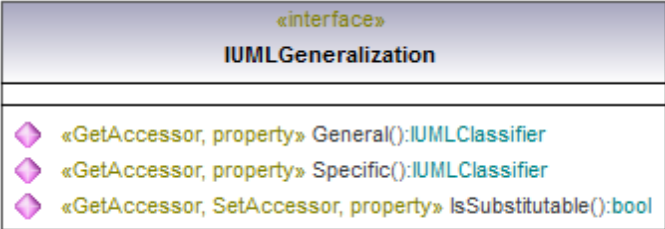
UModelAPI - IUMLGate

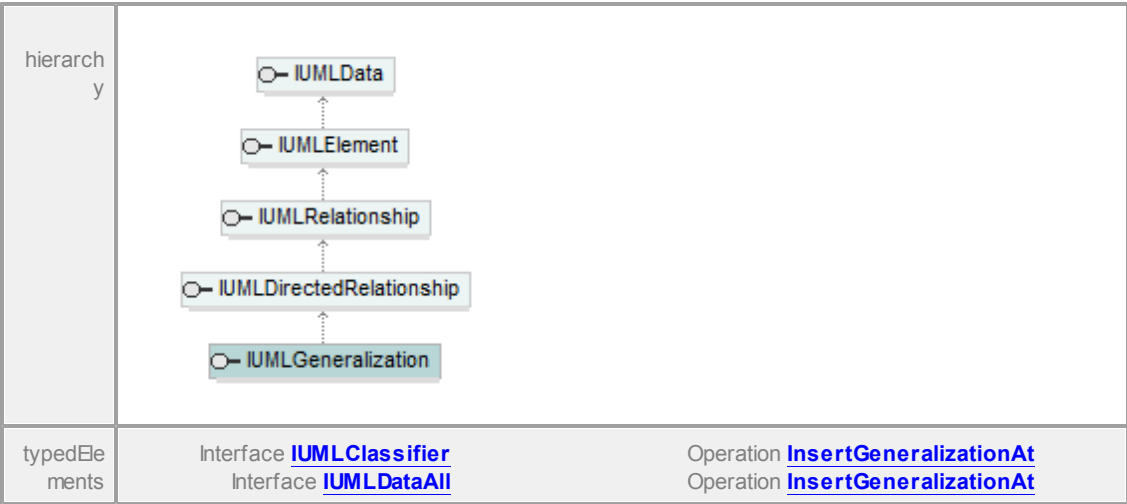
Interface IUMLGate

diagram		
hierarch y		
typedEle ments	Interface IUMLDataAll Interface IUMLInteraction Interface IUMLInteractionUse	Operation InsertActualGateAt InsertFormalGateAt Operation InsertFormalGateAt Operation InsertActualGateAt

UModelAPI - IUMLGeneralization

Interface IUMLGeneralization

diagram	



Operation **IUMLGeneralization::General**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLClassifier			

Operation **IUMLGeneralization::IsSubstitutable**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

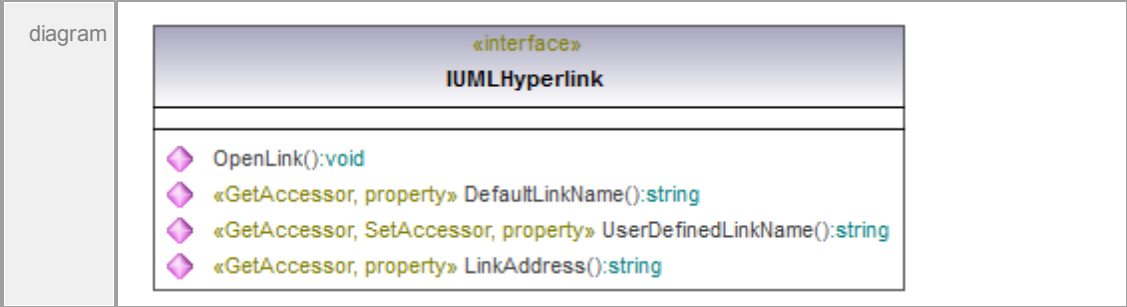
Operation **IUMLGeneralization::Specific**

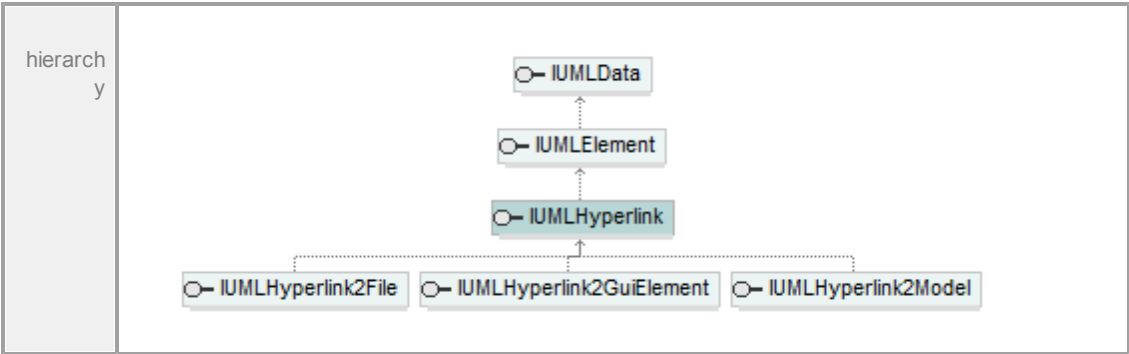
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLClassifier			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLHyperlink

Interface **IUMLHyperlink**





Operation **IUMLHyperlink::DefaultLinkName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLHyperlink::LinkAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLHyperlink::OpenLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	void			

Operation **IUMLHyperlink::UserDefinedLinkName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLHyperlink2File

Interface **IUMLHyperlink2File**



hierarchy			
typedElements	Interface IUMLDataAll Interface IUMLNamedElement	Operation InsertOwnedHyperlink2FileAt Operation InsertOwnedHyperlink2FileAt	

Operation **IUMLHyperlink2File::SetHyperlinkFileAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	in	string			
	return	return	void			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLHyperlink2GuiElement

Interface **IUMLHyperlink2GuiElement**

diagram			
hierarchy			
typedElements	Interface IUMLDataAll Interface IUMLNamedElement	Operation InsertOwnedHyperlink2GuiElementAt Operation InsertOwnedHyperlink2GuiElementAt	

Operation **IUMLHyperlink2GuiElement::LinkedGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLGuiVisible			

	Element

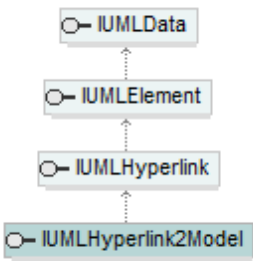
Operation **IUMLHyperlink2GuiElement::LinkedGuiElementCell**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLNamedElement			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLHyperlink2Model

Interface **IUMLHyperlink2Model**

diagram			
hierarchy			
typedElements	Interface IUMLDataAll	Operation InsertOwnedHyperlink2ModelAt	
	Interface IUMLNamedElement	Operation InsertOwnedHyperlink2ModelAt	

Operation **IUMLHyperlink2Model::LinkedModelElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLData			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLInclude

Interface **IUMLInclude**

diagram	«interface» IUMLInclude «GetAccessor, property» Addition():IUMLUseCase «GetAccessor, property» IncludingCase():IUMLUseCase	
hierarchy	<pre>graph BT IUMLData -- > IUMLElement IUMLElement -- > IUMLNamedElement IUMLElement -- > IUMLRelationship IUMLRelationship -- > IUMLDirectedRelationship IUMLInclude -- > IUMLNamedElement IUMLInclude -- > IUMLDirectedRelationship</pre>	
typedElements	Interface IUMLDataAll Interface IUMLUseCase	Operation InsertIncludeAt Operation InsertIncludeAt

Operation **IUMLInclude::Addition**

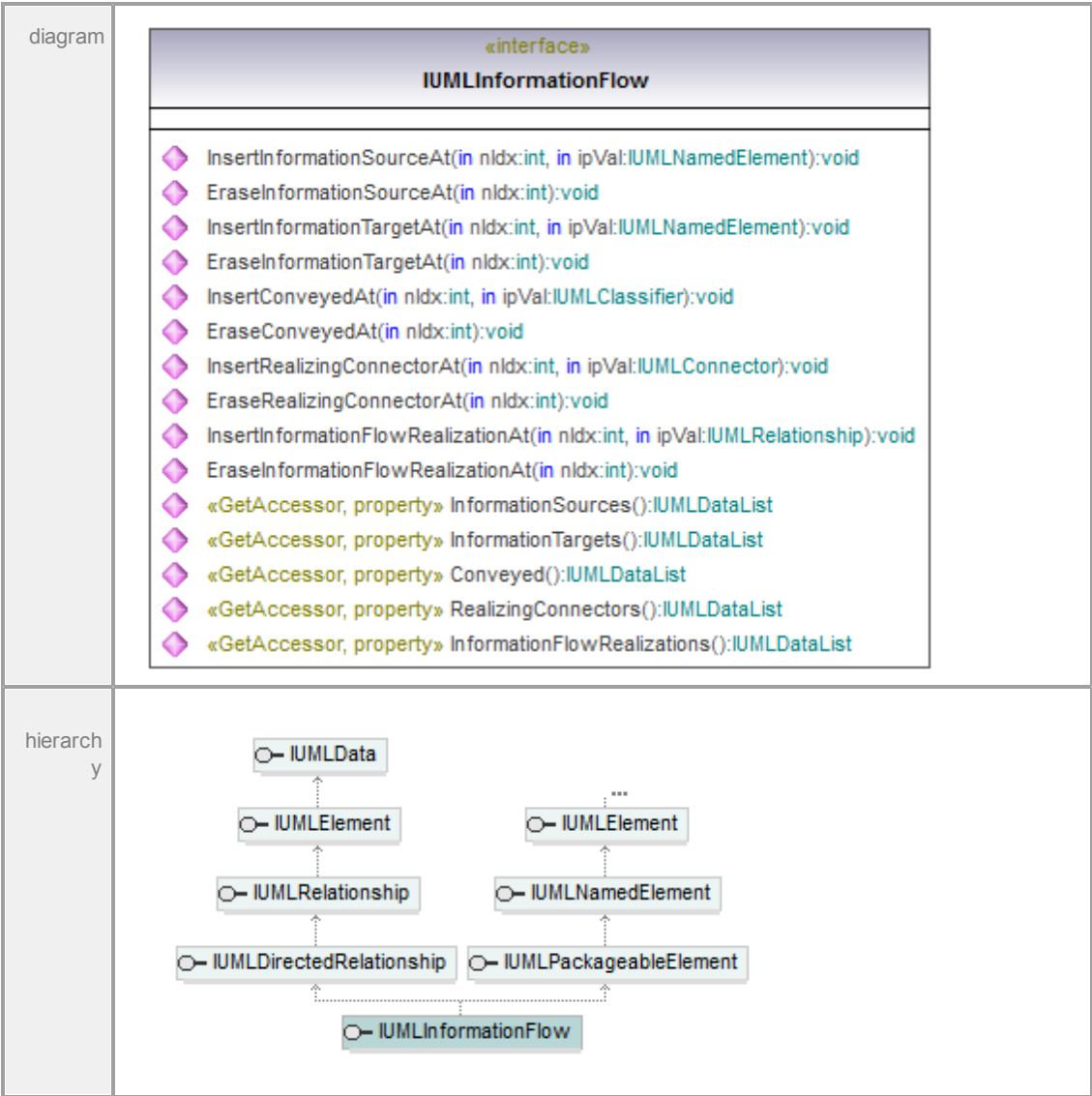
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLUseCase			

Operation **IUMLInclude::IncludingCase**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLUseCase			

UModelAPI - IUMLInformationFlow

Interface **IUMLInformationFlow**



Operation **IUMLInformationFlow::Conveyed**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

Operation **IUMLInformationFlow::EraseConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	void			

Operation **IUMLInformationFlow::EraseInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	void			

Operation **IUMLInformationFlow::EraseInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	void			

Operation **IUMLInformationFlow::EraseInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	void			

Operation **IUMLInformationFlow::EraseRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	void			

Operation **IUMLInformationFlow::InformationFlowRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLInformationFlow::InformationSources**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLInformationFlow::InformationTargets**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLInformationFlow::InsertConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipVal	in	IUMLClassifier			
	return	return	void			

Operation **IUMLInformationFlow::InsertInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipVal	in	IUMLRelationship			
	return	return	void			

Operation **IUMLInformationFlow::InsertInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipVal	in	IUMLNamedElement			
	return	return	void			

Operation **IUMLInformationFlow::InsertInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLNamedElement			
	return	return	void			

Operation **IUMLInformationFlow::InsertRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLConnector			
	return	return	void			

Operation **IUMLInformationFlow::RealizingConnectors**

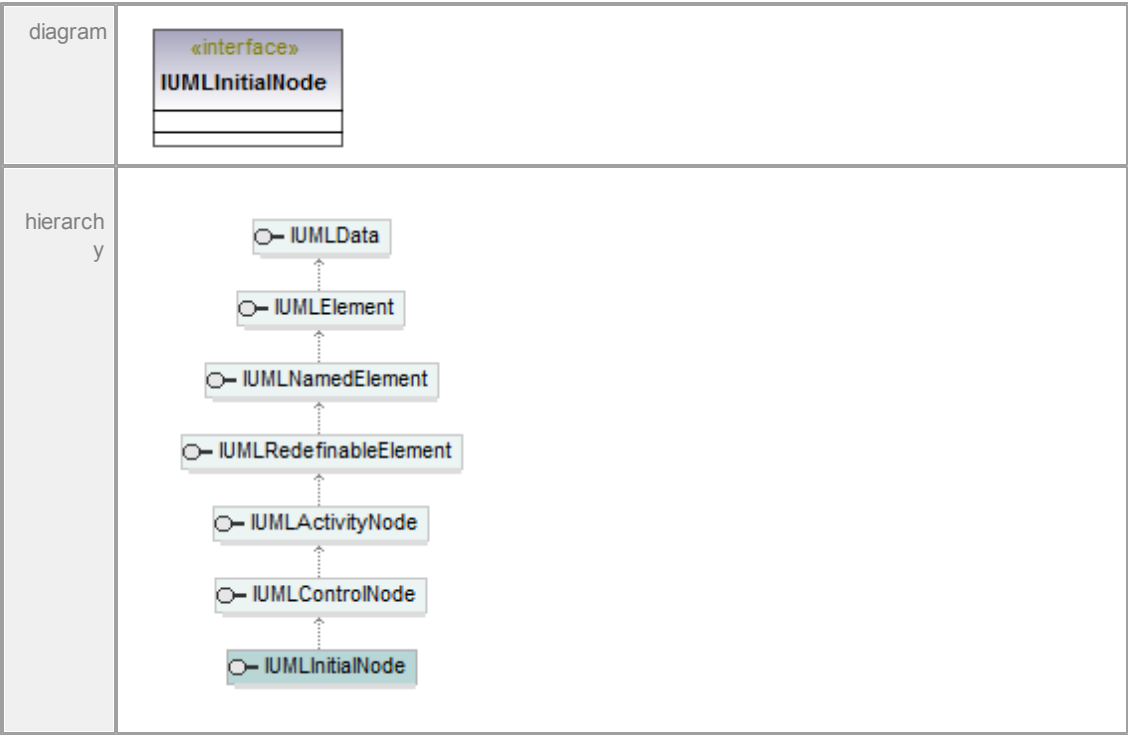
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLInitialNode

Interface **IUMLInitialNode**

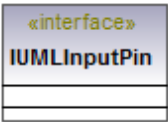
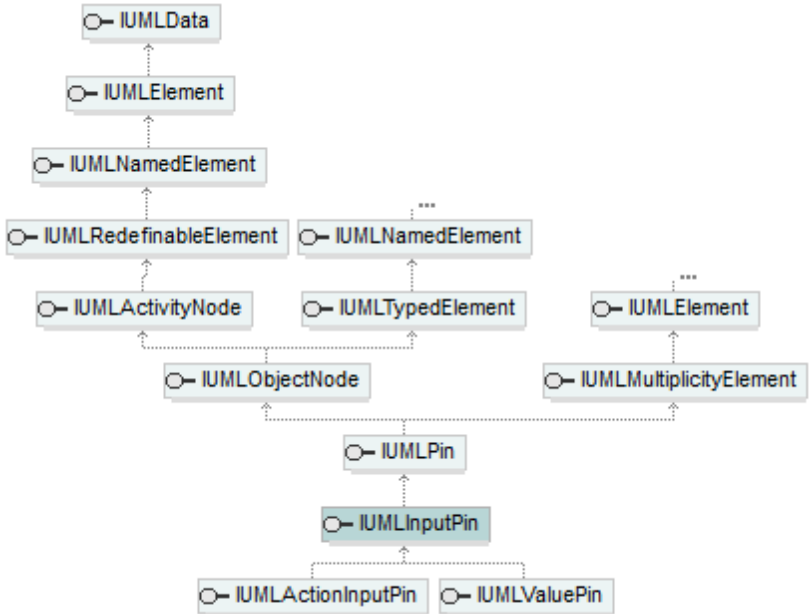


UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLInputPin

Interface IUMLInputPin

diagram		
hierarchy		
typedElements	Interface IUMLCallOperationAction Interface IUMLDataAll Interface IUMLInvocationAction Interface IUMLOpaqueAction Interface IUMLSendSignalAction	Operation CallTarget SetNewCallTarget Operation CallTarget InsertArgumentAt InsertInputValueAt SetNewCallTarget SetNewSignalTarget SignalTarget Operation InsertArgumentAt Operation InsertInputValueAt Operation SetNewSignalTarget SignalTarget

UModelAPI - IUMLInstanceSpecification

Interface **IUMLInstanceSpecification**

diagram	«interface» IUMLInstanceSpecification ◆ InsertSlotAt(in nIdx:int, in ipDefiningFeature:IUMLStructuralFeature):IUMLSlot ◆ SetSlotValueAt(in nIdx:int, in ipForDefiningFeature:IUMLStructuralFeature, in strNewValue:string):IUMLValueSpecification ◆ SetSlotInstanceValueAt(in nIdx:int, in ipForDefiningFeature:IUMLStructuralFeature, in ipInstance:IUMLInstanceSpecification):IUMLInstanceValue ◆ SetNewSpecification(in strKind:string):IUMLValueSpecification ◆ SetNewSpecificationLiteralString(in strNewVal:string):IUMLLiteralString ◆ SetNewSpecificationInstanceValue(in ipInstance:IUMLInstanceSpecification):IUMLInstanceValue ◆ «GetAccessor, property» Slots():IUMLDataList ◆ «GetAccessor, SetAccessor, property» Classifier():IUMLClassifier ◆ «GetAccessor, property» Specification():IUMLValueSpecification	
hierarchy		
typedElements	Interface IUMLConstraint Interface IUMLDataAll Interface IUMLInstanceSpecification Interface IUMLInstanceValue Interface IUMLParameter Interface IUMLProperty Interface IUMLSlot Interface IUMLValueSpecification	Operation SetNewSpecificationInstanceValue Operation InsertSlotInstanceValueAtInstanceOwningInstanceSpec SetNewDefaultValueInstanceValue SetNewSpecificationInstanceValueSetSlotInstanceValueAtInstance Operation SetNewDefaultValueInstanceValue Operation SetNewDefaultValueInstanceValue Operation InsertSlotInstanceValueAtOwningInstance Operation OwningInstanceSpec

Operation **IUMLInstanceSpecification::Classifier**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLClassifier			

Operation **IUMLInstanceSpecification::InsertSlotAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	ipDefiningFeature	in	IUMLStructuralFeature			
	return	return	IUMLSlot			

Operation **IUMLInstanceSpecification::SetNewSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	return	return	IUMLValueSpecification			

Operation **IUMLInstanceSpecification::SetNewSpecificationInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipInstance	in	IUMLInstanceSpecification			
	return	return	IUMLInstanceValue			

Operation **IUMLInstanceSpecification::SetNewSpecificationLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
r	strNewValue	in	string			
	return	return	IUMLLiteralString			

Operation **IUMLInstanceSpecification::SetSlotInstanceValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	ipForDefiningFeature	in	IUMLStructuralFeature			
	ipInstance	in	IUMLInstanceSpecification			
	return	return	IUMLInstanceValue			

Operation **IUMLInstanceSpecification::SetSlotValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	ipForDefiningFeature	in	IUMLStructuralFeature			
	strNewValue	in	string			
	return	return	IUMLValueSpecification			

Operation **IUMLInstanceSpecification::Slots**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLSlot .					

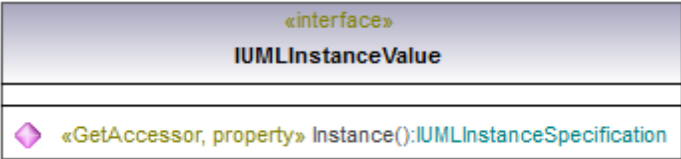
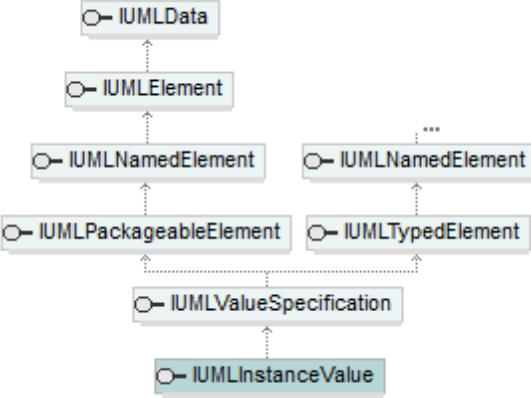
Operation **IUMLInstanceSpecification::Specification**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLValueSpecification			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLInstanceValue

Interface **IUMLInstanceValue**

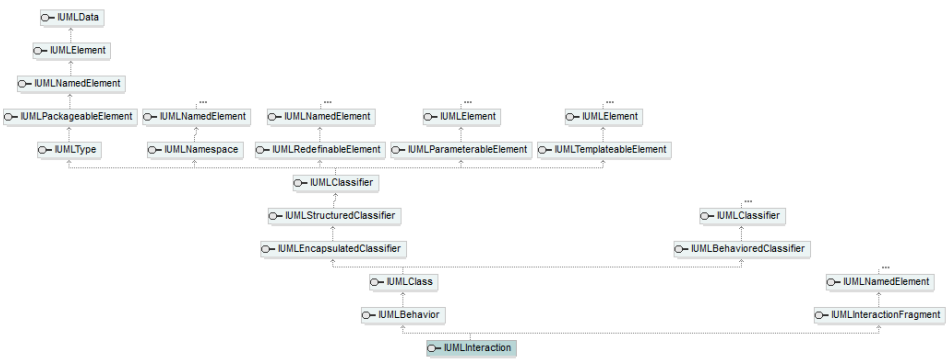
diagram		
hierarchy		
typedElements	Interface IUMLConstraint Interface IUMLDataAll Interface IUMLInstanceSpecification Interface IUMLParameter Interface IUMLProperty Interface IUMLSlot	Operation SetNewSpecificationInstanceValue Operation InsertSlotInstanceValueAt SetNewDefaultValueInstanceValue SetNewSpecificationInstanceValue SetSlotInstanceValueAt Operation SetNewSpecificationInstanceValue SetSlotInstanceValueAt Operation SetNewDefaultValueInstanceValue Operation SetNewDefaultValueInstanceValue Operation InsertSlotInstanceValueAt

Operation **IUMLInstanceValue::Instance**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLInstanceSpecification			

UModelAPI - IUMLInteraction

Interface IUMLInteraction

diagram	«interface» IUMLInteraction ◆ InsertLifelineAt(in nldx:int):IUMLLifeline ◆ InsertFragmentAt(in nldx:int, in strKind:string):IUMLInteractionFragment ◆ InsertMessageAt(in nldx:int):IUMLMessage ◆ InsertFormalGateAt(in nldx:int):IUMLGate ◆ «GetAccessor, property» Lifelines():IUMLDataList ◆ «GetAccessor, property» Fragments():IUMLDataList ◆ «GetAccessor, property» Messages():IUMLDataList ◆ «GetAccessor, property» FormalGates():IUMLDataList
hierarchy	
typedElements	Interface IUMLDataAll Interface IUMLInteractionUse Operation RefersTo Operation RefersTo

Operation IUMLInteraction::FormalGates

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documentation	A list of elements of type IUMLGate .					

Operation IUMLInteraction::Fragments

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documentation	A list of elements of type IUMLInteractionFragment .					

Operation IUMLInteraction::InsertFormalGateAt

parameter	name nldx	direction in	type int	type modifier	multiplicity	default

	return	return	IUMLGate

Operation **IUMLInteraction::InsertFragmentAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	strKind	in	string			
	return	return	IUMLInteractionFragment			

Operation **IUMLInteraction::InsertLifelineAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLLifeline			

Operation **IUMLInteraction::InsertMessageAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLMessage			

Operation **IUMLInteraction::Lifelines**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLLifeline .					

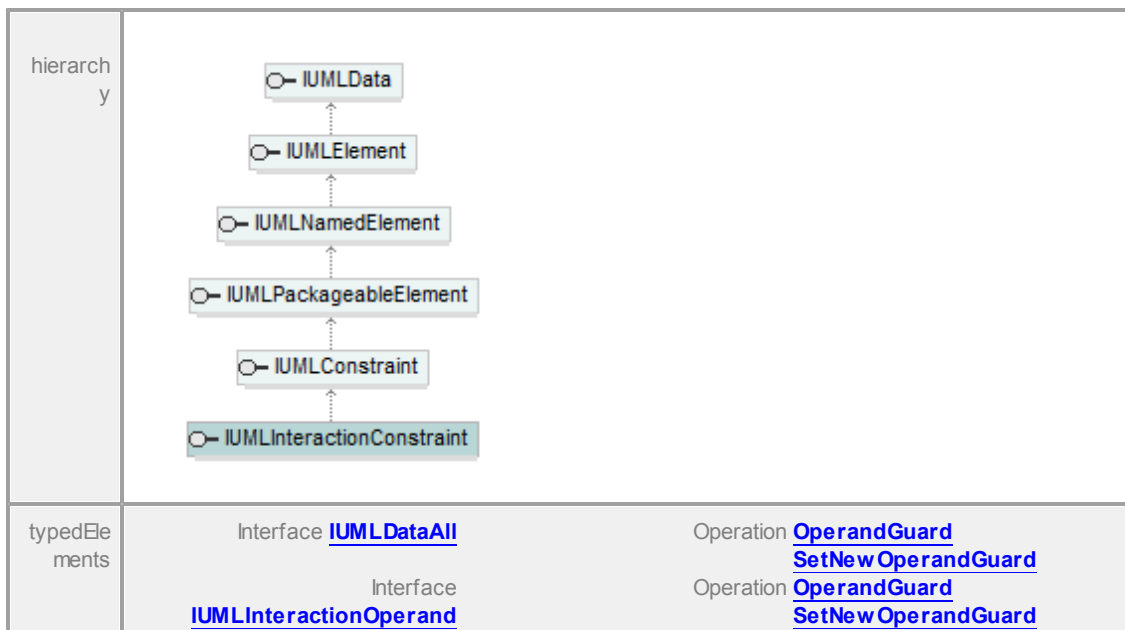
Operation **IUMLInteraction::Messages**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLMessage .					

UModelAPI - IUMLInteractionConstraint

Interface **IUMLInteractionConstraint**

diagram	<pre> classDiagram class IUMLInteractionConstraint { <<interface>> SetNewMaxInt(in strKind:string):IUMLValueSpecification SetNewMinInt(in strKind:string):IUMLValueSpecification «GetAccessor, property» MaxInt():IUMLValueSpecification «GetAccessor, property» MinInt():IUMLValueSpecification } </pre>



Operation **IUMLInteractionConstraint::MaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLValueSpecification			

Operation **IUMLInteractionConstraint::MinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLValueSpecification			

Operation **IUMLInteractionConstraint::SetNewMaxInt**

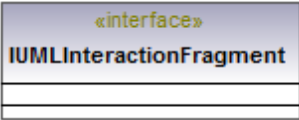
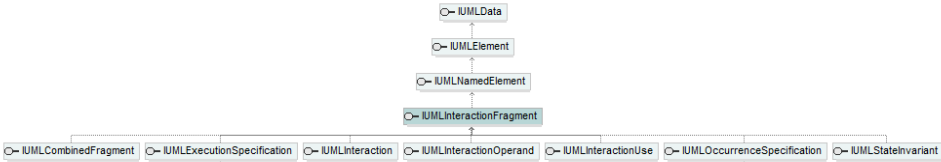
parameter	name	direction	type	type modifier	multiplicity	default
	strKind return	in return	string IUMLValueSpecification			

Operation **IUMLInteractionConstraint::SetNewMinInt**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind return	in return	string IUMLValueSpecification			

UModelAPI - IUMLInteractionFragment

Interface **IUMLInteractionFragment**

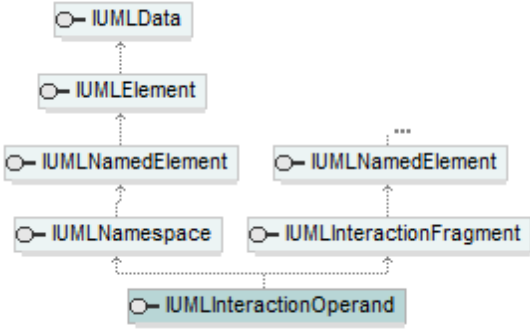
diagram		
hierarch y		
typedEle ments	Interface IUMLDataAll Interface IUMLInteraction Interface IUMLLifetime	Operation InsertCoveredByAt InsertFragmentAt Operation InsertFragmentAt Operation InsertCoveredByAt

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLInteractionOperand

Interface **IUMLInteractionOperand**

diagram		
hierarch y		
typedEle ments	Interface IUMLCombinedFragment Interface IUMLDataAll	Operation InsertOperandAt Operation InsertOperandAt

Operation **IUMLInteractionOperand::OperandGuard**

paramete	name	direction	type	type modifier	multiplicity	default

r	return	return	IUMLInteractionConstraint

Operation **IUMLInteractionOperand::SetNewOperandGuard**

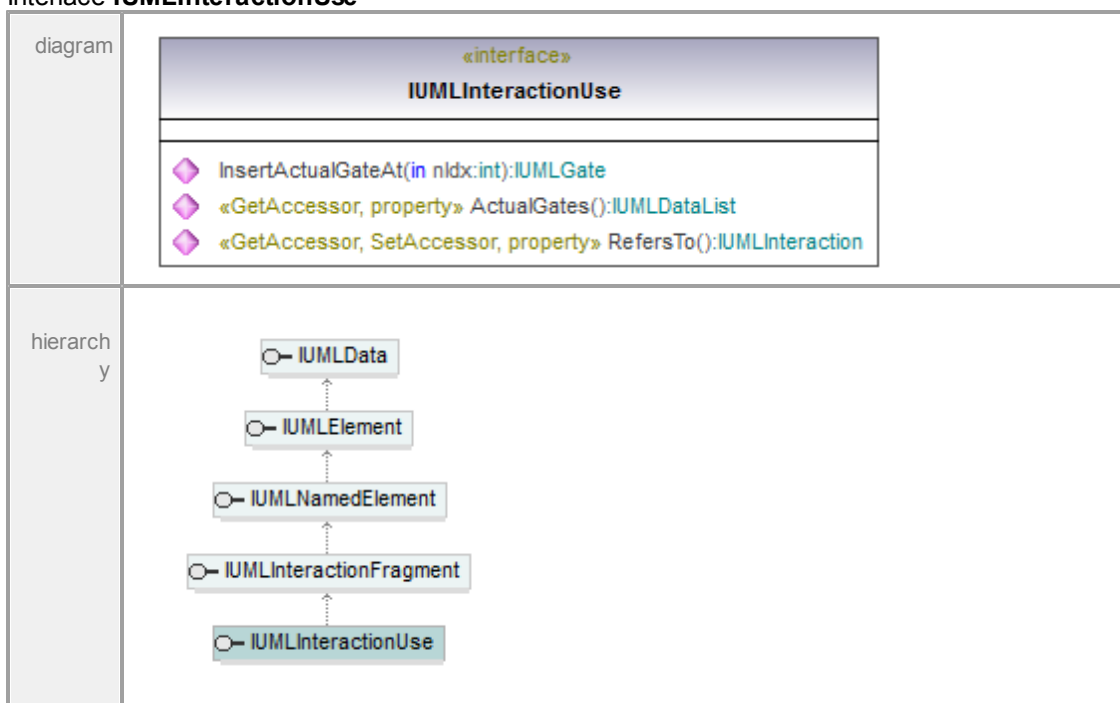
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLInteractionConstraint			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLInteractionUse

Interface **IUMLInteractionUse**



Operation **IUMLInteractionUse::ActualGates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLGate .					

Operation **IUMLInteractionUse::InsertActualGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx return	in return	int IUMLGate			

Operation **IUMLInteractionUse::RefersTo**

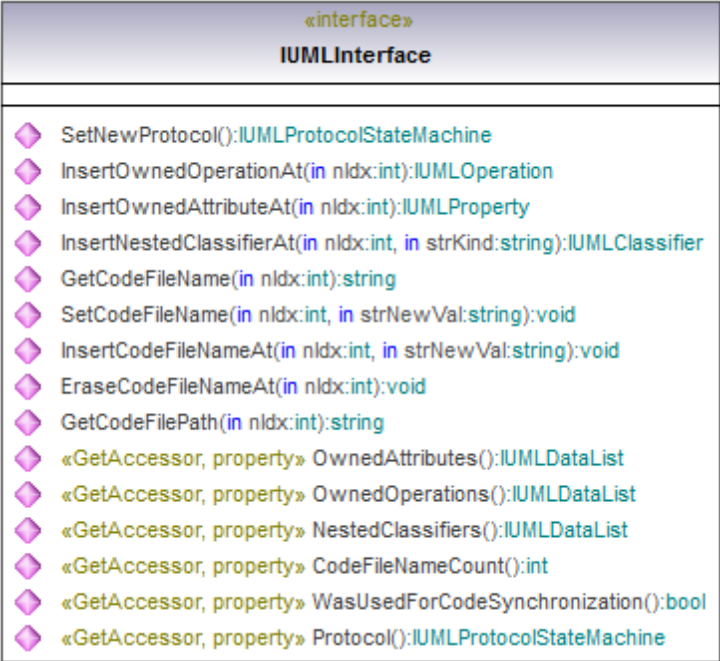
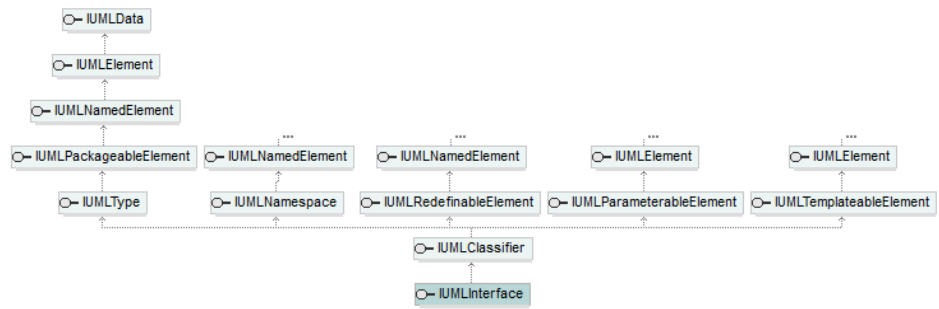
parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	IUMLInteraction

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLInterface

Interface **IUMLInterface**

diagram		
hierarchy		
typedElements	Interface IUMLBehavioeredClassifier Interface IUMLClassifier Interface IUMLDataAll Interface IUMLInterfaceRealization Interface IUMLOperation Interface IUMLProperty Interface	Operation InsertInterfaceRealizationAt Operation NestingInterface Operation Contract Operation InsertInterfaceRealizationAt Operation NestingInterface Operation Contract Operation Interface Operation Interface Operation Interface

IUMLProtocolStateMachine

Operation **IUMLInterface::CodeFileNameCount**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLInterface::EraseCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLInterface::GetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	string			

Operation **IUMLInterface::GetCodeFilePath**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	string			
documentation	get the full code file path					

Operation **IUMLInterface::InsertCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	strNewVal	in	string			
	return	return	void			

Operation **IUMLInterface::InsertNestedClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	strKind	in	string			
	return	return	IUMLClassifier			

Operation **IUMLInterface::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	IUMLProperty			

Operation **IUMLInterface::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	IUMLOperation			

Operation **IUMLInterface::NestedClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDatList			
documentation	A list of elements of type IUMLClassifier .					

Operation IUMLInterface::OwnedAttributes

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDatList			
documentation	A list of elements of type IUMLProperty .					

Operation IUMLInterface::OwnedOperations

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDatList			
documentation	A list of elements of type IUMLOperation .					

Operation IUMLInterface::Protocol

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLProtocolStateMachine			

Operation IUMLInterface::SetCodeFileName

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	strNewVal	in	string			
	return	return	void			

Operation IUMLInterface::SetNewProtocol

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLProtocolStateMachine			

Operation IUMLInterface::WasUsedForCodeSynchronization

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

UModelAPI - IUMLInterfaceRealization

Interface **IUMLInterfaceRealization**

diagram	«interface» IUMLInterfaceRealization «GetAccessor, property» ImplementingClassifier():IUMLBehavioredClassifier «GetAccessor, property» Contract():IUMLInterface	
hierarch y	<pre>graph BT IUMLInterfaceRealization --> IUMLRealization IUMLRealization --> IUMLAbstraction IUMLAbstraction --> IUMLDependency IUMLDependency --> IUMLPackageableElement IUMLPackageableElement --> IUMLNamedElement IUMLNamedElement --> IUMLData IUMLPackageableElement --> IUMLDirectedRelationship IUMLDirectedRelationship --> IUMLRelationship IUMLRelationship --> IUMLElement IUMLDirectedRelationship --> IUMLRealization</pre>	
typedEle ments	Interface IUMLBehavioredClassifier Interface IUMLDataAll	Operation InsertInterfaceRealizationAt Operation InsertInterfaceRealizationAt

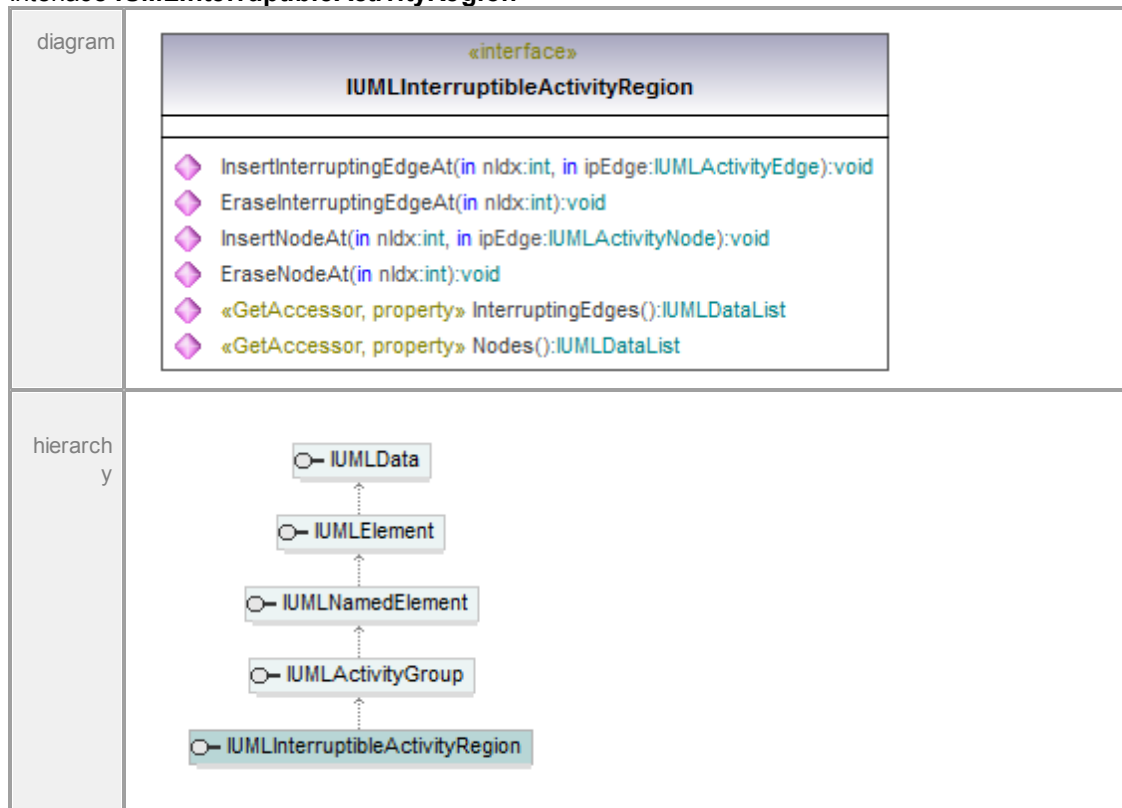
Operation **IUMLInterfaceRealization::Contract**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLInterface			

Operation **IUMLInterfaceRealization::ImplementingClassifier**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLBehavioredClassifier			

UModelAPI - IUMLInterruptibleActivityRegion

Interface **IUMLInterruptibleActivityRegion**Operation **IUMLInterruptibleActivityRegion::EraseInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	void			

Operation **IUMLInterruptibleActivityRegion::EraseNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	void			

Operation **IUMLInterruptibleActivityRegion::InsertInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	ipEdge	in	IUMLActivityEdge			
	return	return	void			

Operation **IUMLInterruptibleActivityRegion::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	ipEdge	in	IUMLActivityNode			
	return	return	void			

Operation **IUMLInterruptibleActivityRegion::InterruptingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLActivityEdge .					

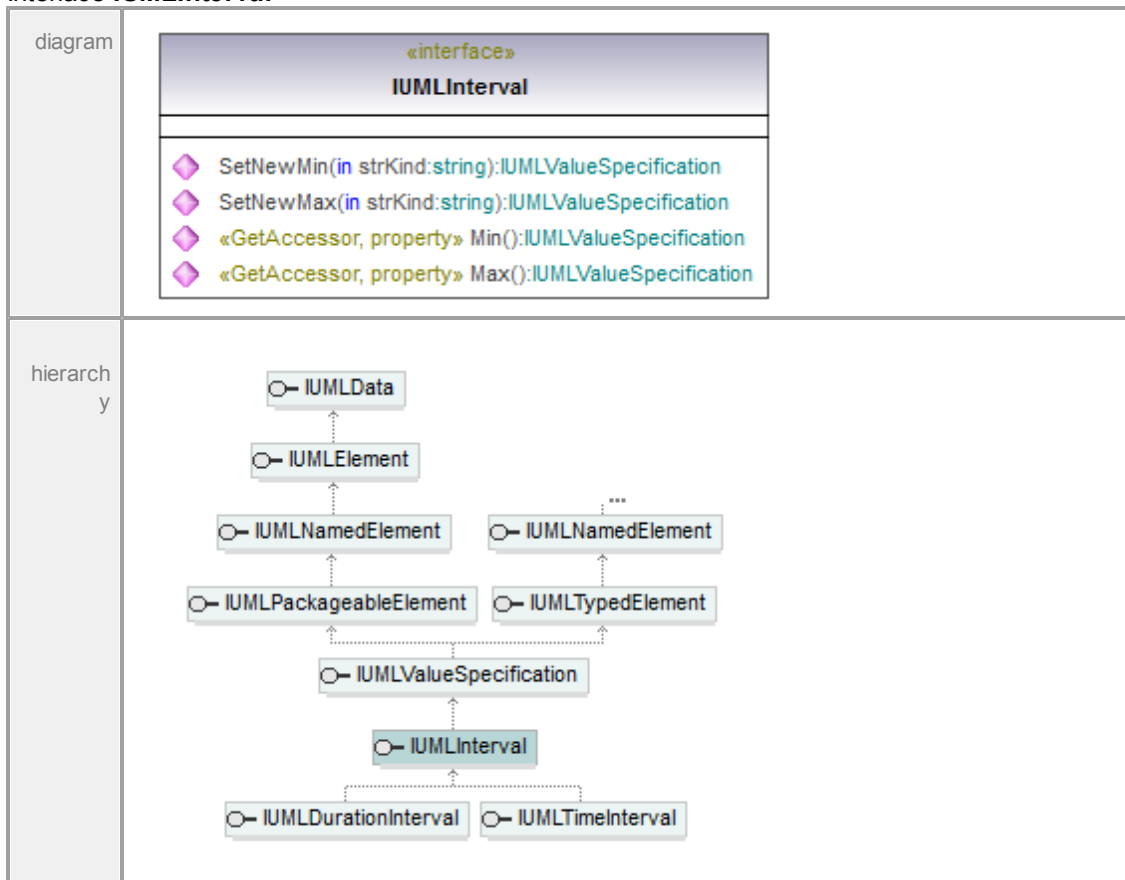
Operation **IUMLInterruptibleActivityRegion::Nodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLActivityNode .					

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLInterval

Interface **IUMLInterval**Operation **IUMLInterval::Max**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLValueSpe			

	cification

Operation **IUMLInterval::Min**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLValueSpecification			

Operation **IUMLInterval::SetNewMax**

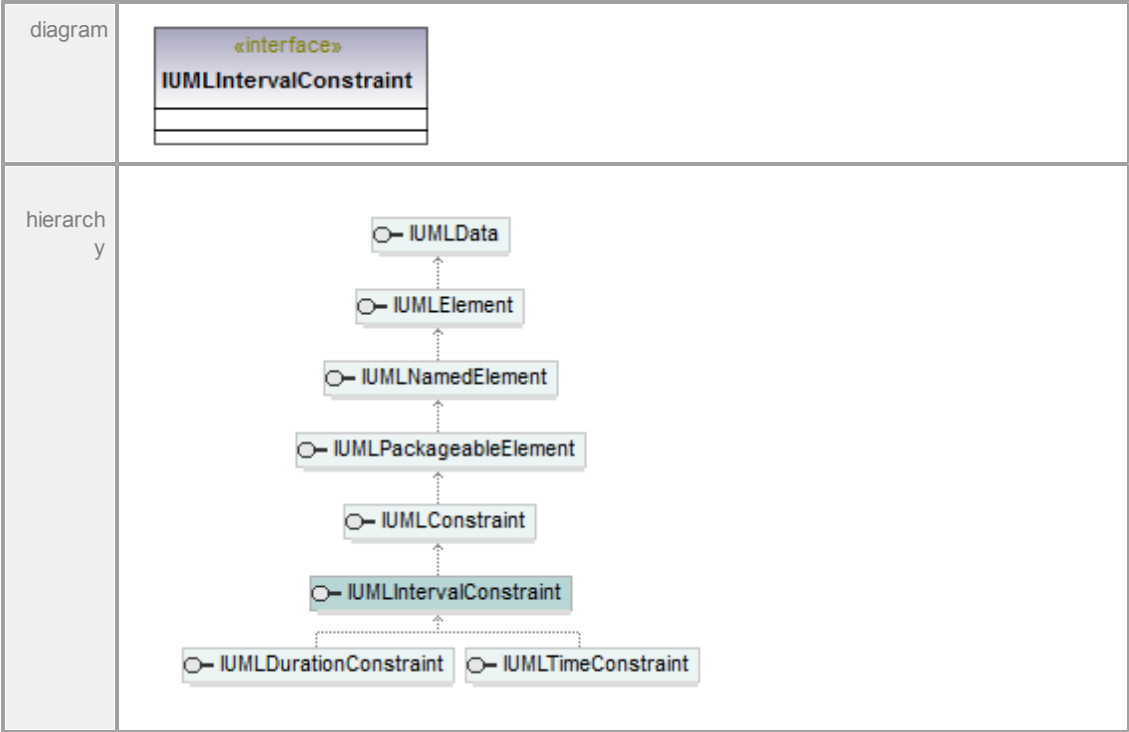
parameter	name	direction	type	type modifier	multiplicity	default
	strKind return	in return	string IUMLValueSpecification			

Operation **IUMLInterval::SetNewMin**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind return	in return	string IUMLValueSpecification			

UModelAPI - IUMLIntervalConstraint

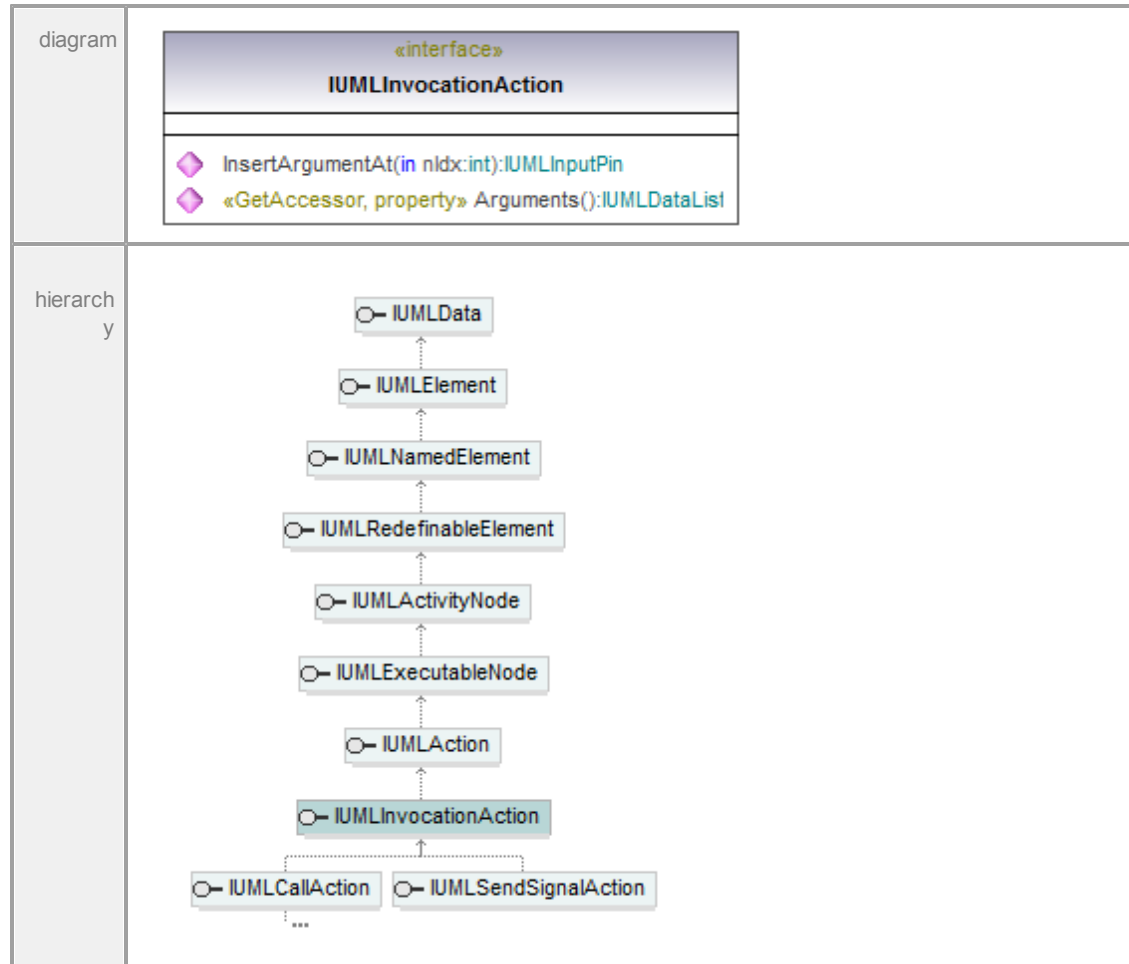
Interface **IUMLIntervalConstraint**



UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLInvocationAction

Interface **IUMLInvocationAction**Operation **IUMLInvocationAction::Arguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLInputPin .					

Operation **IUMLInvocationAction::InsertArgumentAt**

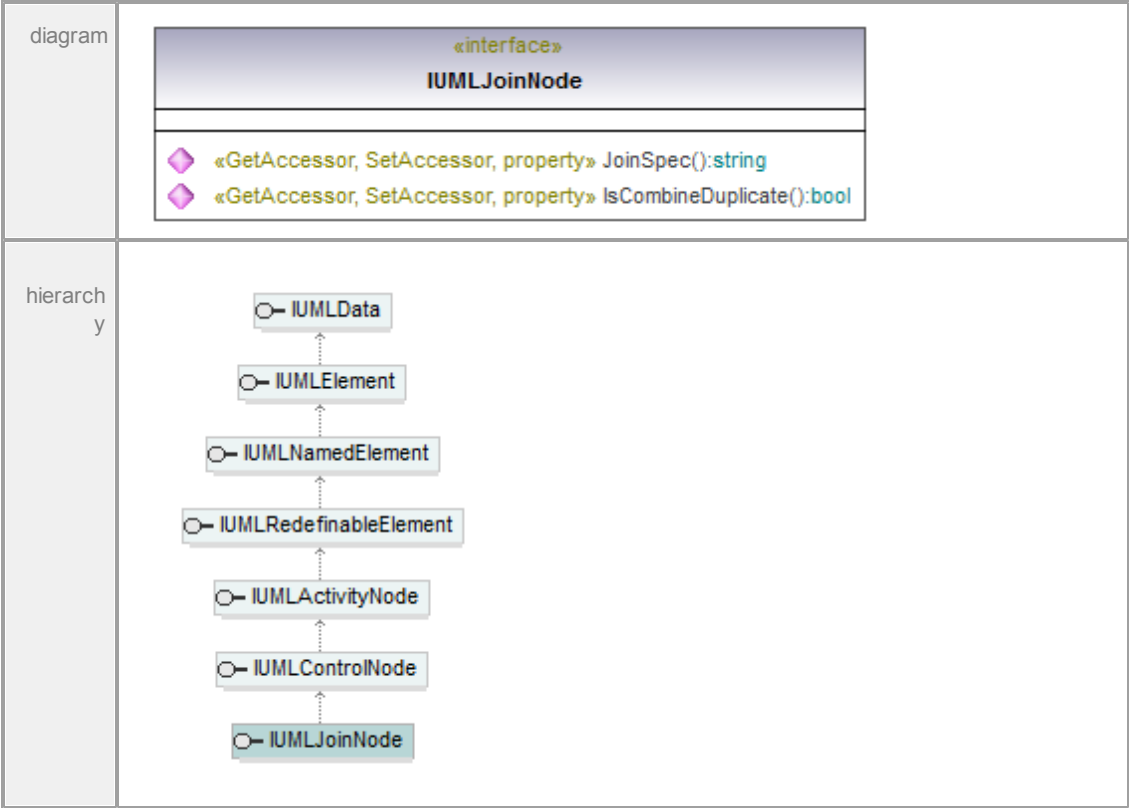
parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	IUMLInputPin			

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLJoinNode

Interface **IUMLJoinNode**



Operation **IUMLJoinNode::IsCombineDuplicate**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLJoinNode::JoinSpec**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

UModelAPI - IUMLLifeLine

Interface IUMLLifeLine

diagram		
hierarchy		
typedElements	Interface IUMLDataAll Interface IUMLInteraction Interface IUMLMessage Interface IUMLOccurrenceSpecification Interface IUMLStateInvariant	Operation Covered GetSourceLifeline GetTargetLifeline InsertLifelineAt Operation InsertLifelineAt Operation GetSourceLifeline GetTargetLifeline Operation Covered Operation Covered

Operation IUMLLifeLine::EraseCoveredByAt

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	void			

Operation IUMLLifeLine::InsertCoveredByAt

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLInteractionFragment			
	return	return	void			

Operation IUMLLifeLine::Represents

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLConnectableElement			

Operation IUMLLifeLine::Selector

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLValueSpecification			

Operation **IUMLLifetime::SetNewSelector**

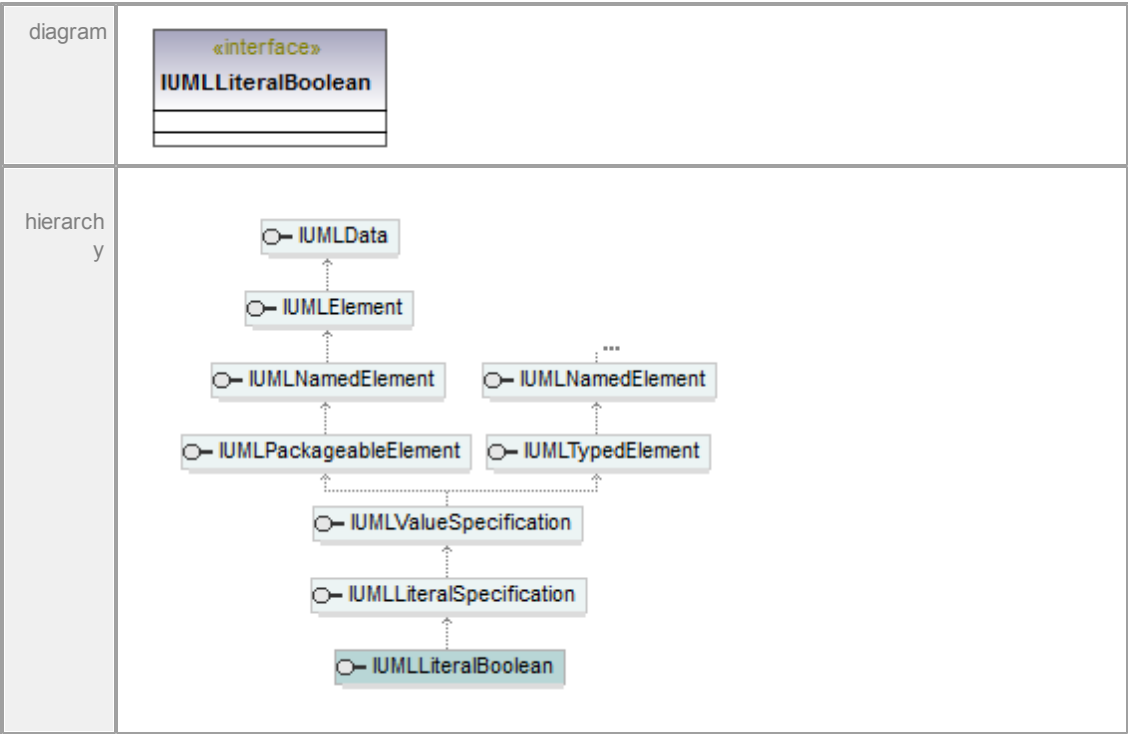
parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLValueSpecification			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLLiteralBoolean

Interface **IUMLLiteralBoolean**



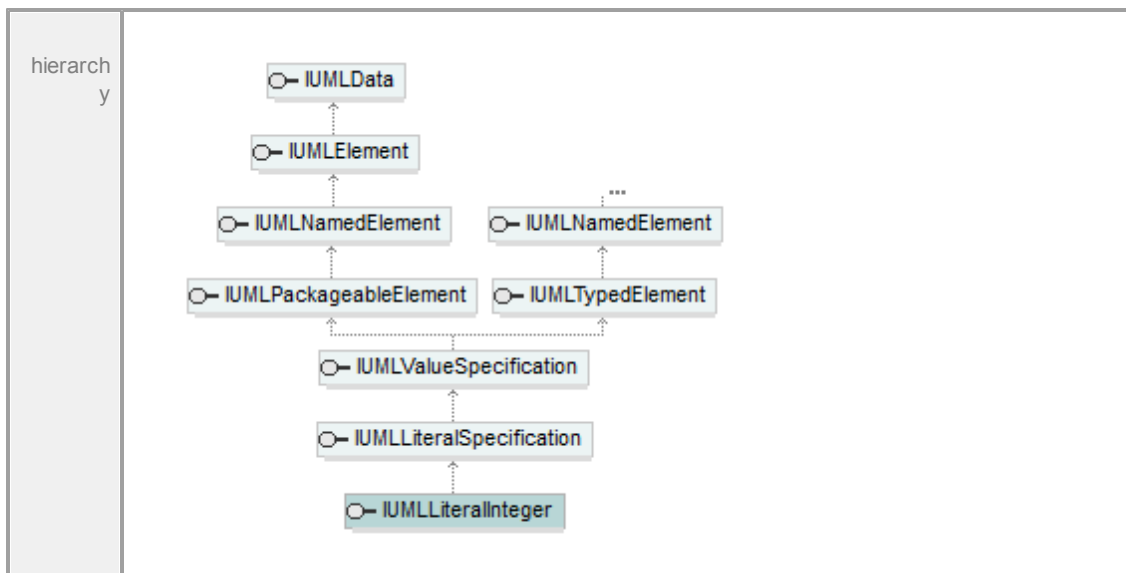
UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLLiteralInteger

Interface **IUMLLiteralInteger**



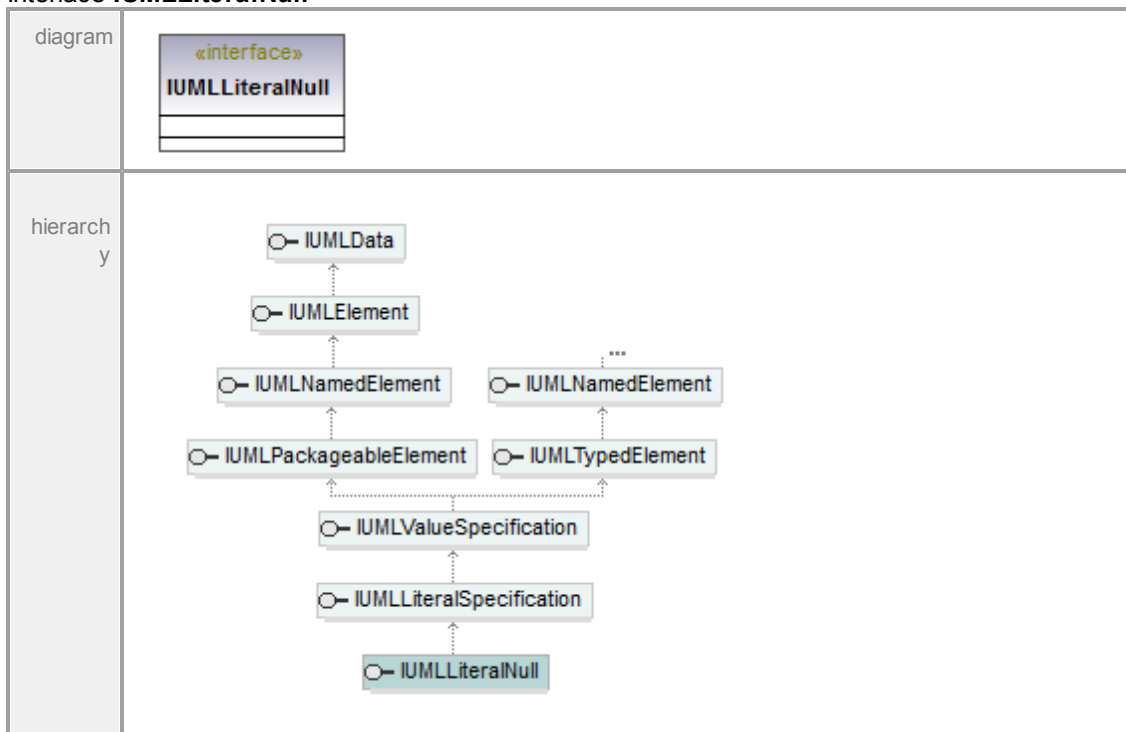


UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLLiteralNull

Interface IUMLLiteralNull

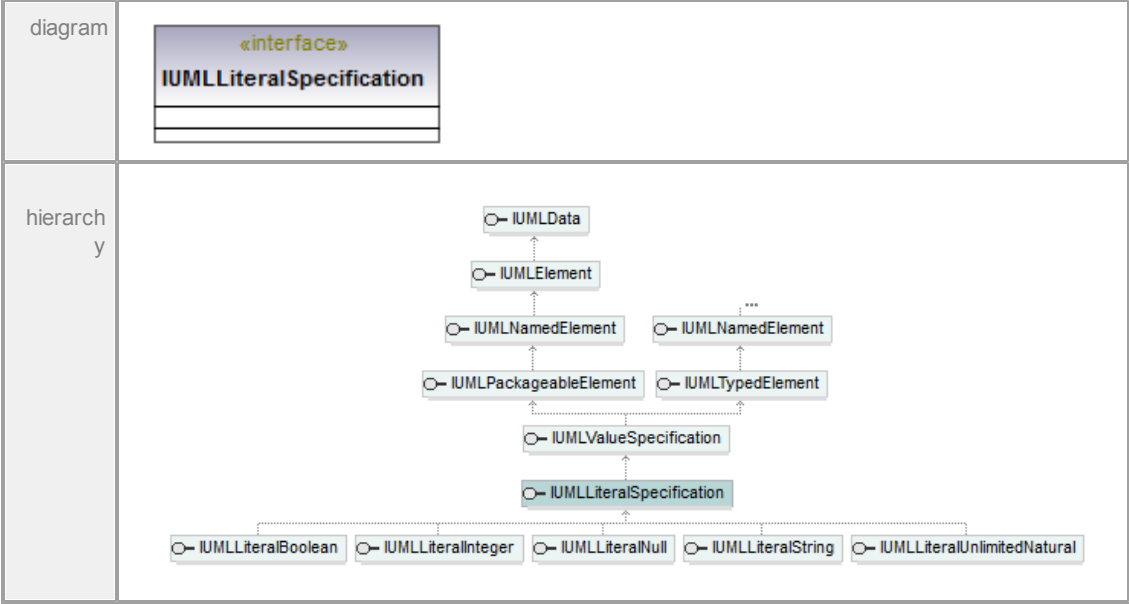


UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLLiteralSpecification

Interface **IUMLLiteralSpecification**

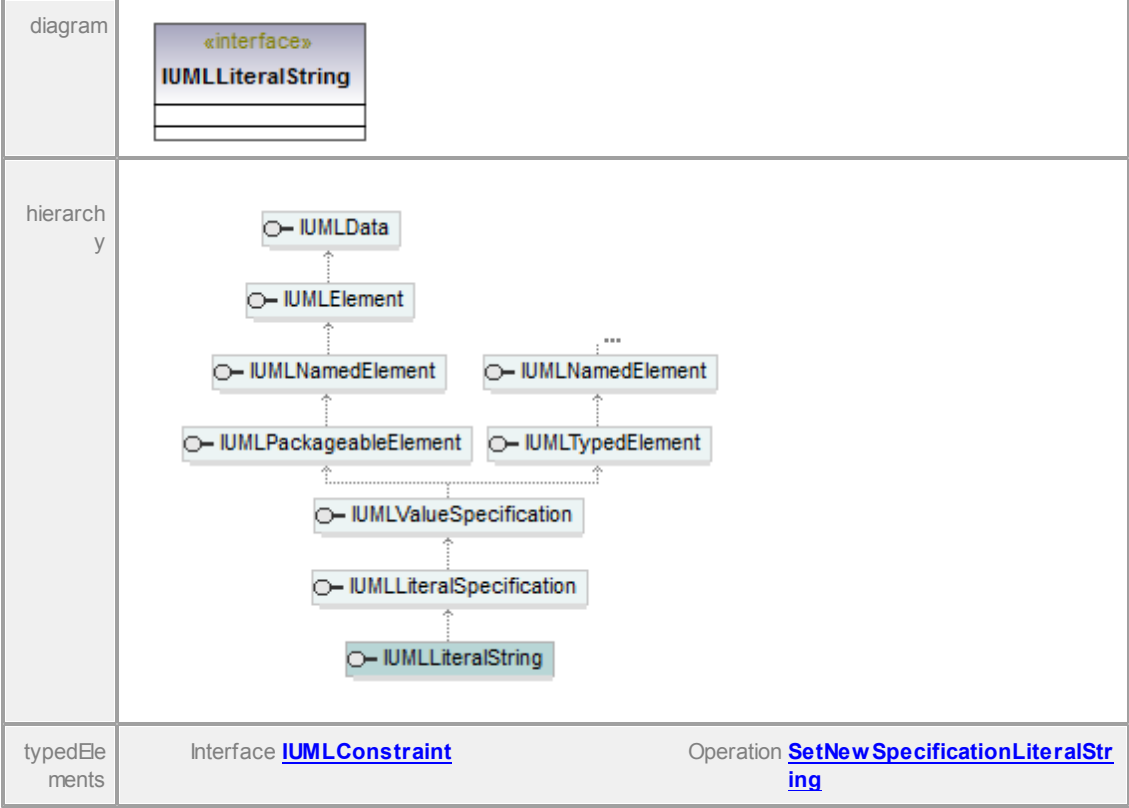


UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLLiteralString

Interface **IUMLLiteralString**



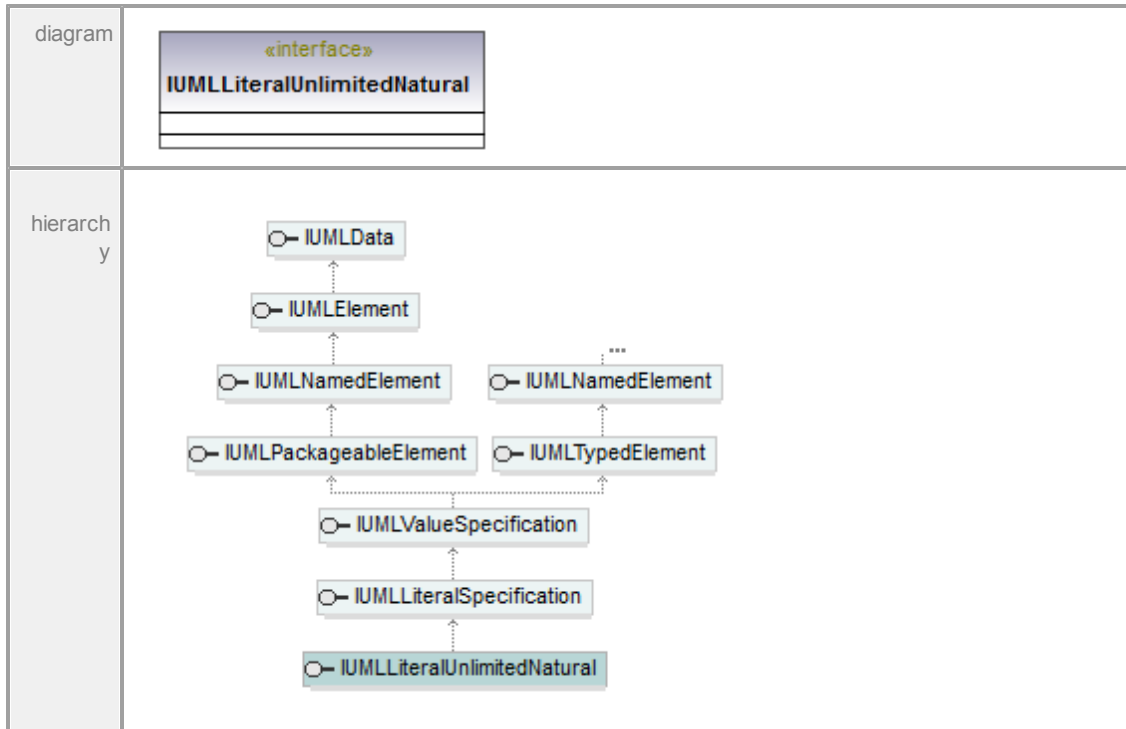
Interface IUMLDataAll	Operation SetNewDefaultValueLiteralString
Interface IUMLInstanceSpecification	Operation SetNewSpecificationLiteralString
Interface IUMLParameter	Operation SetNewDefaultValueLiteralString
Interface IUMLProperty	Operation SetNewDefaultValueLiteralString

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLLiteralUnlimitedNatural

Interface **IUMLLiteralUnlimitedNatural**

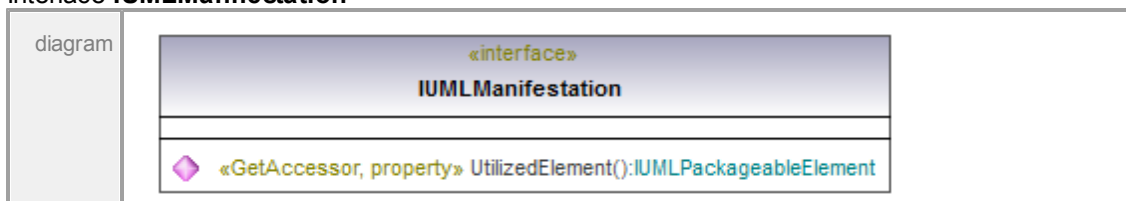


UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLManifestation

Interface **IUMLManifestation**



hierarchy	<pre>graph BT IUMLManifestation --> IUMLAbstraction IUMLAbstraction --> IUMLDependency IUMLDependency --> IUMLPackageableElement IUMLDependency --> IUMLDirectedRelationship IUMLPackageableElement --> IUMLNamedElement IUMLNamedElement --> IUMLElement IUMLDirectedRelationship --> IUMLRelationship IUMLRelationship --> IUMLElement IUMLData --> IUMLElement IUMLRelationship -.-> IUMLElement</pre>	
typedElements	Interface IUMLArtifact Interface IUMLDataAll	Operation InsertManifestationAt Operation InsertManifestationAt

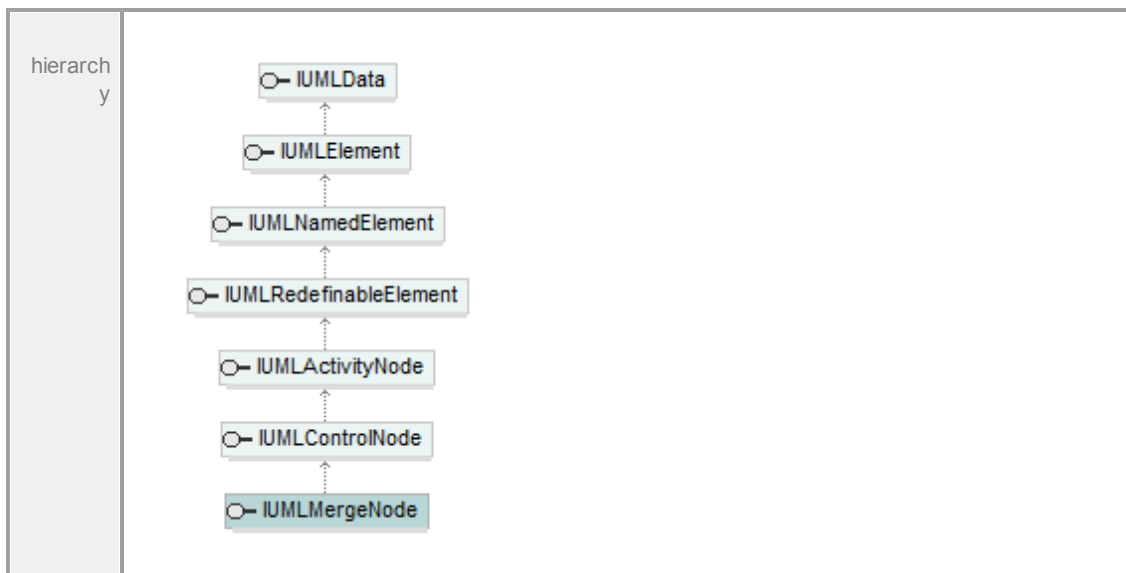
Operation **IUMLManifestation::UtilizedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLPackageableElement			

UModelAPI - IUMLMergeNode

Interface **IUMLMergeNode**

diagram	



UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLMessage

Interface IUMLMessage



typedElements	Interface IUMLDataAll Interface IUMLInteraction Interface IUMLMessageEnd	Operation InsertMessageAt Message Operation InsertMessageAt Message Operation Message

Operation **IUMLMessage::GetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLOperation			

Operation **IUMLMessage::GetSourceLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLLifeline			

Operation **IUMLMessage::GetTargetLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLLifeline			

Operation **IUMLMessage::InsertOwnedArgumentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	ndx	in	int			
	strKind	in	string			
	return	return	IUMLValueSpecification			

Operation **IUMLMessage::MessageKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ENUMUMLMessageKind			

Operation **IUMLMessage::MessageSort**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ENUMUMLMessageSort			

Operation **IUMLMessage::OwnedArguments**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLValueSpecification .					

Operation **IUMLMessage::ReceiveEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLMessageEnd			

Operation **IUMLMessage::SendEvent**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	IUMLMessageEnd

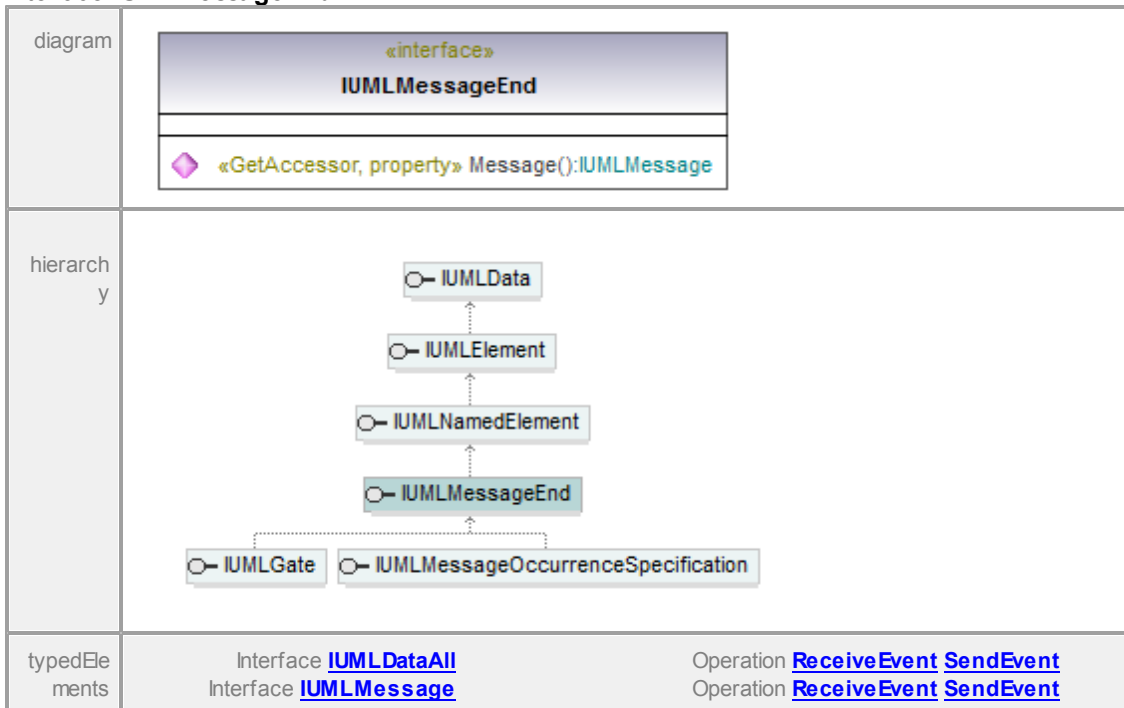
Operation **IUMLMessage::SetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	ipVal	in	IUMLOperation			
	return	return	void			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLMessageEnd

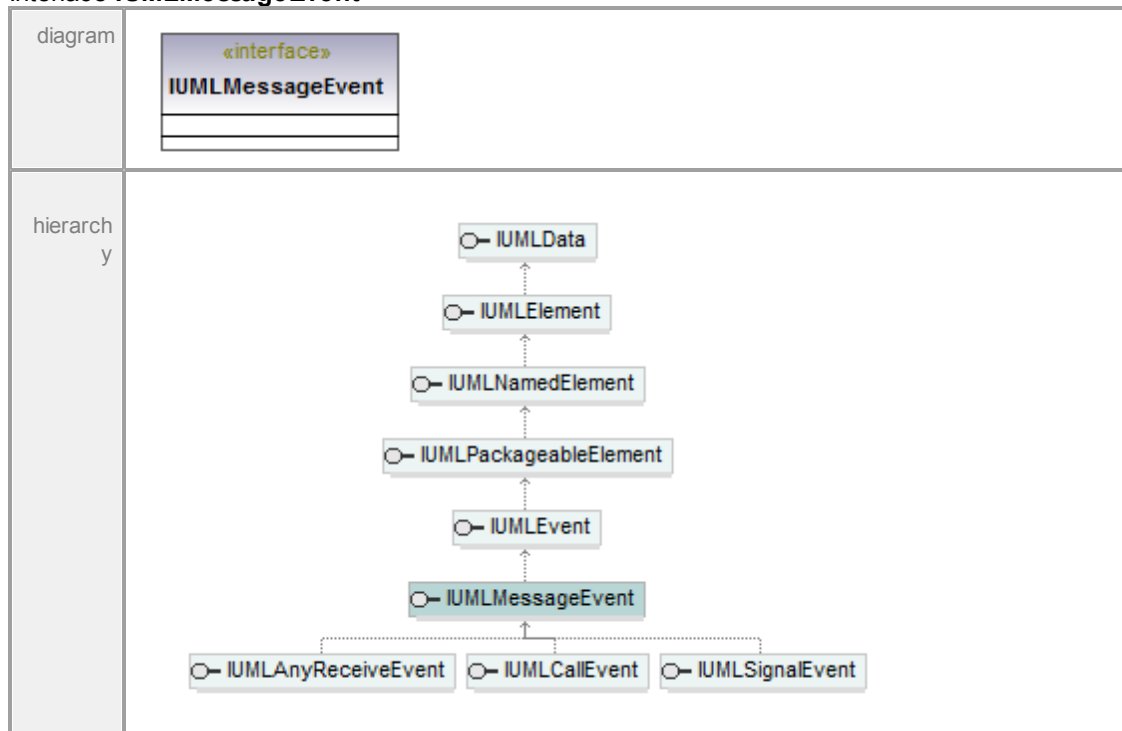
Interface **IUMLMessageEnd**Operation **IUMLMessageEnd::Message**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLMessage			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

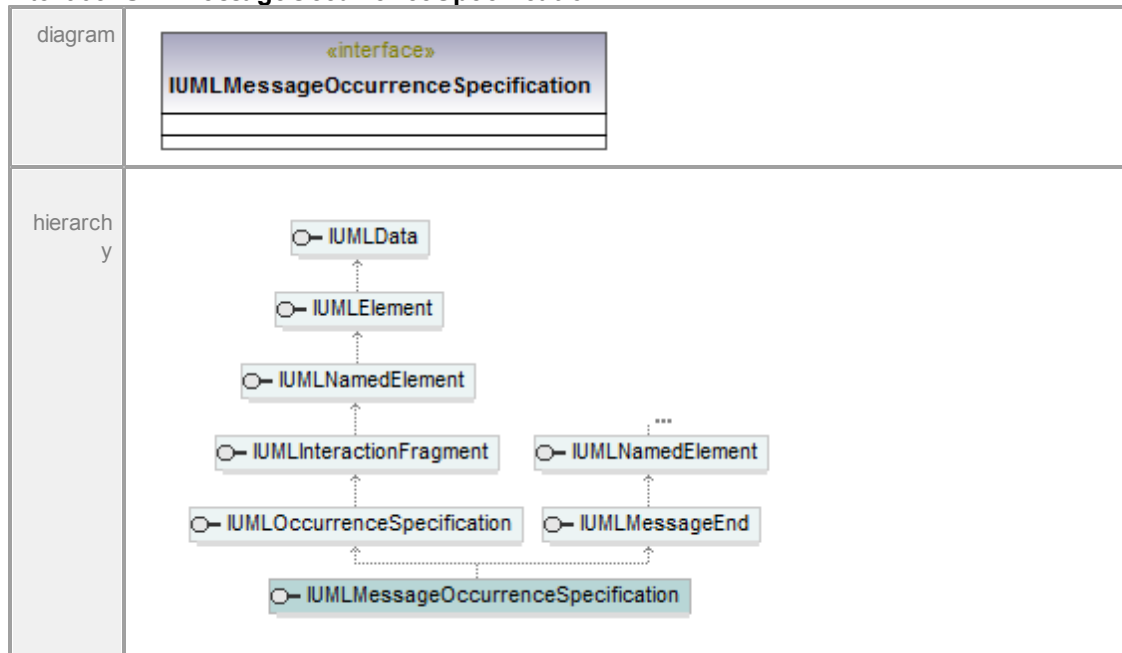
08/03/15 13:42:06

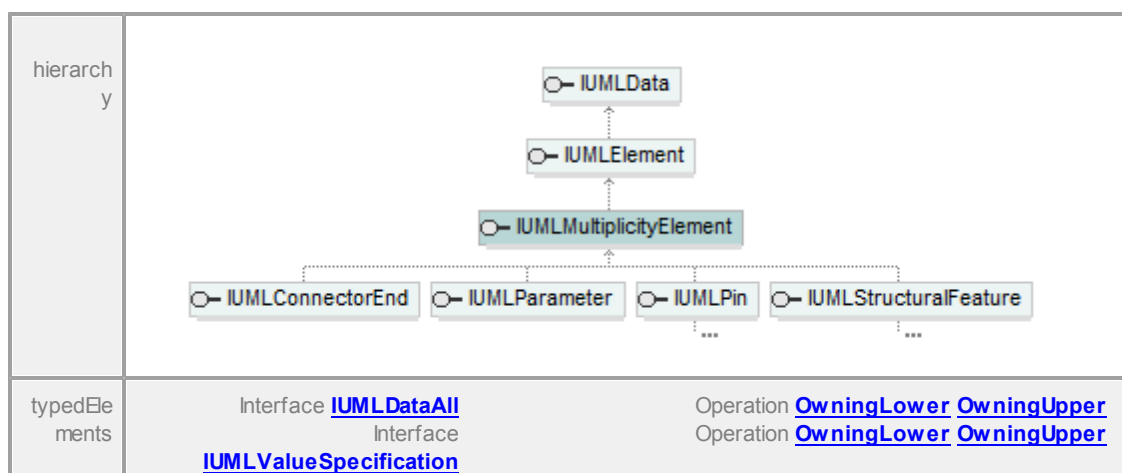
UModelAPI - IUMLMessageEvent

Interface **IUMLMessageEvent**UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLMessageOccurrenceSpecification

Interface **IUMLMessageOccurrenceSpecification**



Operation **IUMLMultiplicityElement::GetMultiplicity**

parameter	name	direction	type	type modifier	multiplicity	default
	bWithBrackets	in	bool			
	return	return	string			

Operation **IUMLMultiplicityElement::InsertLowerUpperValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	strLower	in	string			
	strUpper	in	string			
	return	return	void			

Operation **IUMLMultiplicityElement::IsOrdered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLMultiplicityElement::IsUnique**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLMultiplicityElement::LowerValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLValueSpecification .					

Operation **IUMLMultiplicityElement::SetMultiplicity**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	in	string			
	return	return	void			

Operation **IUMLMultiplicityElement::UpperValues**

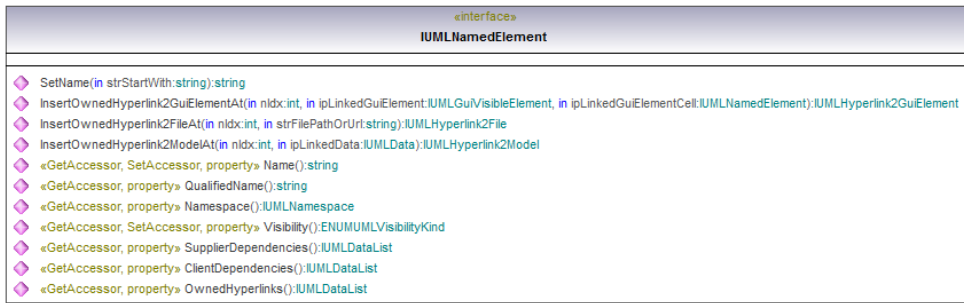
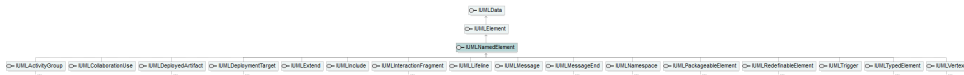
parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documentation	A list of elements of type IUMLValueSpecification .					

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLNamedElement

Interface **IUMLNamedElement**

diagram		
hierarchy		
typedElements	Interface IUMLCommentTextHyperlink Interface IUMLDataAll Interface IUMLGuiTextHyperlink Interface IUMLHyperlink2GuiElement Interface IUMLInformationFlow Interface IUMLNamedElement Interface IUMLNamespace Interface IUMLTimeObservation	Operation SetHyperlinkGuiElementAddress Operation FindOwnedMemberWithQualifiedName Operation InsertInformationSourceAt Operation InsertInformationTargetAt Operation InsertOwnedHyperlink2GuiElementAt Operation LinkedGuiElementCell Operation SetHyperlinkGuiElementAddress TimeObservationEvent Operation SetHyperlinkGuiElementAddress Operation LinkedGuiElementCell Operation InsertInformationSourceAt Operation InsertInformationTargetAt Operation InsertOwnedHyperlink2GuiElementAt Operation FindOwnedMemberWithQualifiedName Operation TimeObservationEvent

Operation **IUMLNamedElement::ClientDependencies**

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
document	A list of elements of type IUMLDependency .					

tation	

Operation **IUMLNamedElement::InsertOwnedHyperlink2FileAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	strFilePathOrUrl	in	string			
	return	return	IUMLHyperlink2File			

Operation **IUMLNamedElement::InsertOwnedHyperlink2GuiElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipLinkedGuiElement	in	IUMLGuiVisibleElement			
	ipLinkedGuiElementCell	in	IUMLNamedElement			
	return	return	IUMLHyperlink2GuiElement			

Operation **IUMLNamedElement::InsertOwnedHyperlink2ModelAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipLinkedData	in	IUMLData			
	return	return	IUMLHyperlink2Model			

Operation **IUMLNamedElement::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLNamedElement::Namespace**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLNamespace			

Operation **IUMLNamedElement::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLHyperlink .					

Operation **IUMLNamedElement::QualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLNamedElement::SetName**

parameter	name	direction	type	type modifier	multiplicity	default

r	strStartWith return	in return	string string
documenta- tion	This function will find and set a unique name (starting with 'strStartWith') that the element is distinguishable in its parent namespace.		

Operation **IUMLNamedElement::SupplierDependencies**

paramete- r	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documenta- tion	A list of elements of type IUMLDependency .					

Operation **IUMLNamedElement::Visibility**

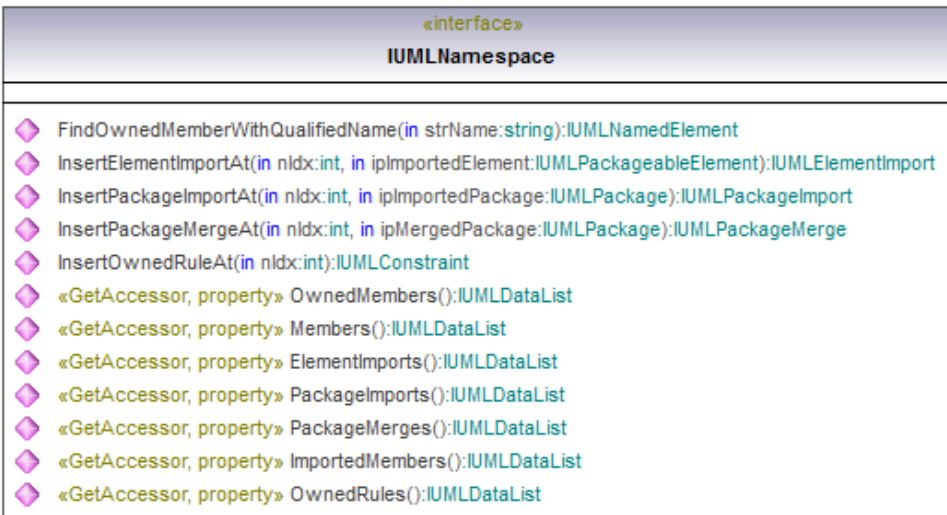
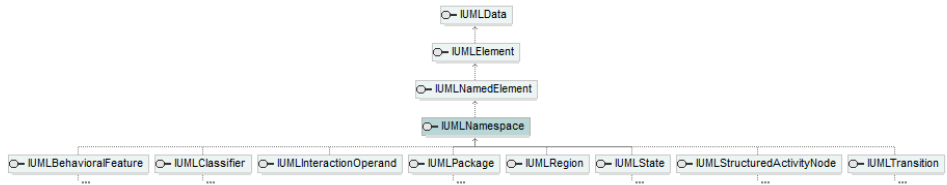
paramete- r	name return	direction return	type ENUMUMLVisib- ilityKind	type modifier	multiplicity	default

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLNamespace

Interface **IUMLNamespace**

diagram		
hierarch- y		
typedEle- ments	Interface IUMLConstraint Interface IUMLDataAll Interface IUMLElementImport	Operation Context Operation Context ImportingNamespace Namespace Operation ImportingNamespace

	Interface IUMLNamedElement Interface IUMLPackageImport	Operation Namespace Operation ImportingNamespace

Operation **IUMLNamespace::ElementImports**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLElementImport .					

Operation **IUMLNamespace::FindOwnedMemberWithQualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default
r	strName return	in return	string IUMLNamedElement			

Operation **IUMLNamespace::ImportedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLPackageableElement .					

Operation **IUMLNamespace::InsertElementImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx ipImportedElement return	in in return	int IUMLPackageableElement IUMLElementImport			

Operation **IUMLNamespace::InsertOwnedRuleAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx return	in return	int IUMLConstraint			

Operation **IUMLNamespace::InsertPackageImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx ipImportedPackage return	in in return	int IUMLPackage IUMLPackageImport			

Operation **IUMLNamespace::InsertPackageMergeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx ipMergedPackage return	in in return	int IUMLPackage IUMLPackageM			

	erge

Operation **IUMLNamespace::Members**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLNamedElement .					

Operation **IUMLNamespace::OwnedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLNamedElement .					

Operation **IUMLNamespace::OwnedRules**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLConstraint .					

Operation **IUMLNamespace::PackageImports**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLPackageImport .					

Operation **IUMLNamespace::PackageMerges**

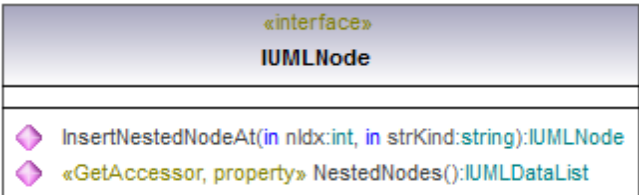
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLPackageMerge .					

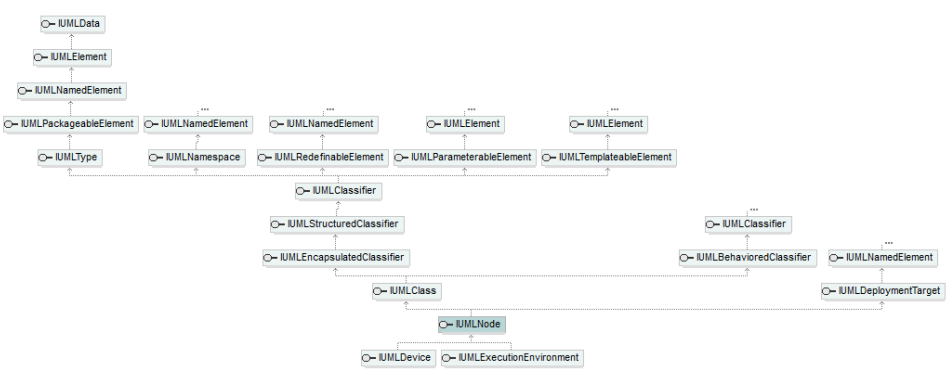
UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLNode

Interface **IUMLNode**

diagram	 <pre> «interface» IUMLNode + InsertNestedNodeAt(in ndx:int, in strKind:string):IUMLNode + «GetAccessor, property» NestedNodes():IUMLDataList </pre>

hierarchy	
typedElements	Interface IUMLDataAll Interface IUMLNode Operation InsertNestedNodeAt Operation InsertNestedNodeAt

Operation **IUMLNode::InsertNestedNodeAt**

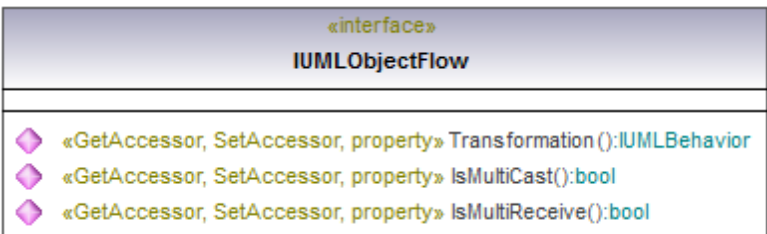
parameter	name	direction	type	type modifier	multiplicity	default
	ndx	in	int			
	strKind	in	string			
	return	return	IUMLNode			

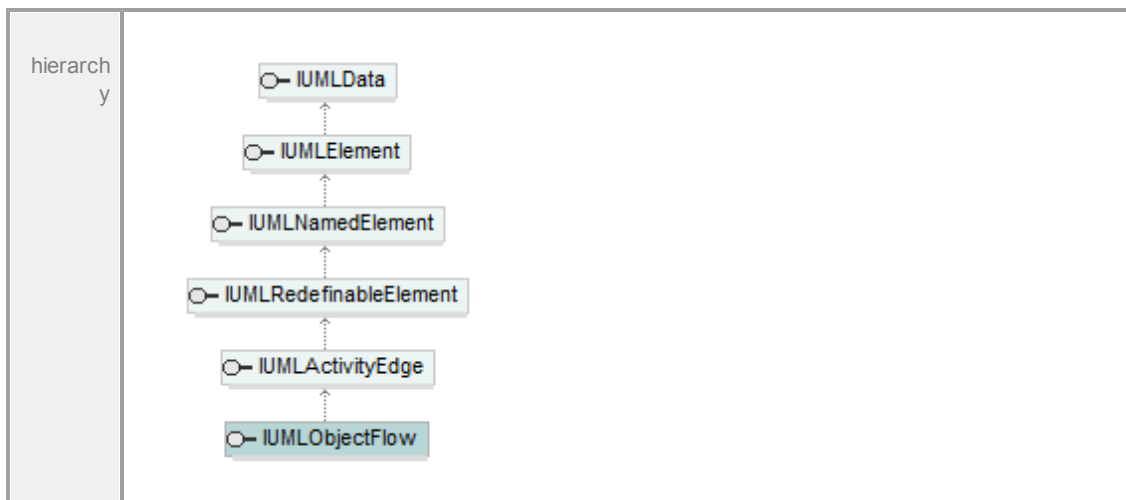
Operation **IUMLNode::NestedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLNode .					

UModelAPI - IUMLObjectFlow

Interface **IUMLObjectFlow**

diagram	



Operation **IUMLObjectFlow::IsMultiCast**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLObjectFlow::IsMultiReceive**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLObjectFlow::Transformation**

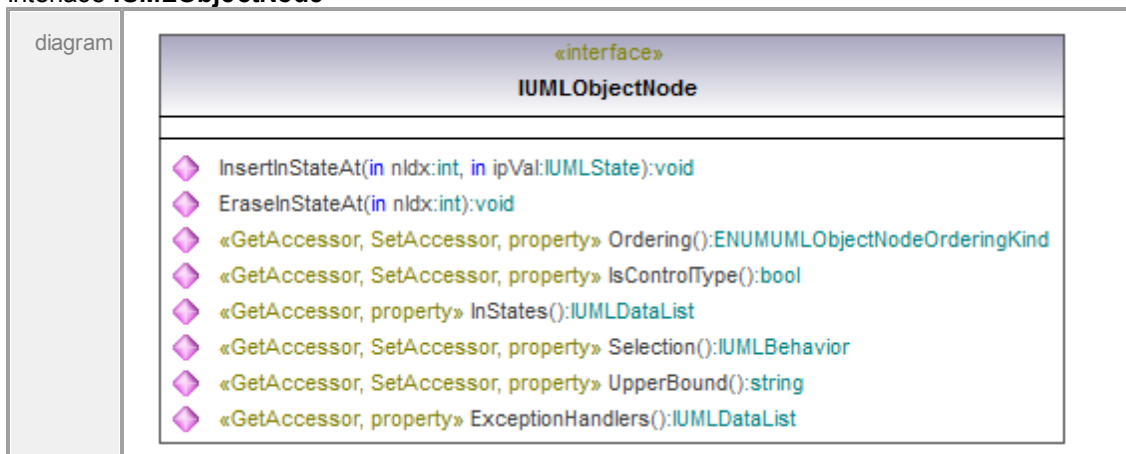
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLBehavior			

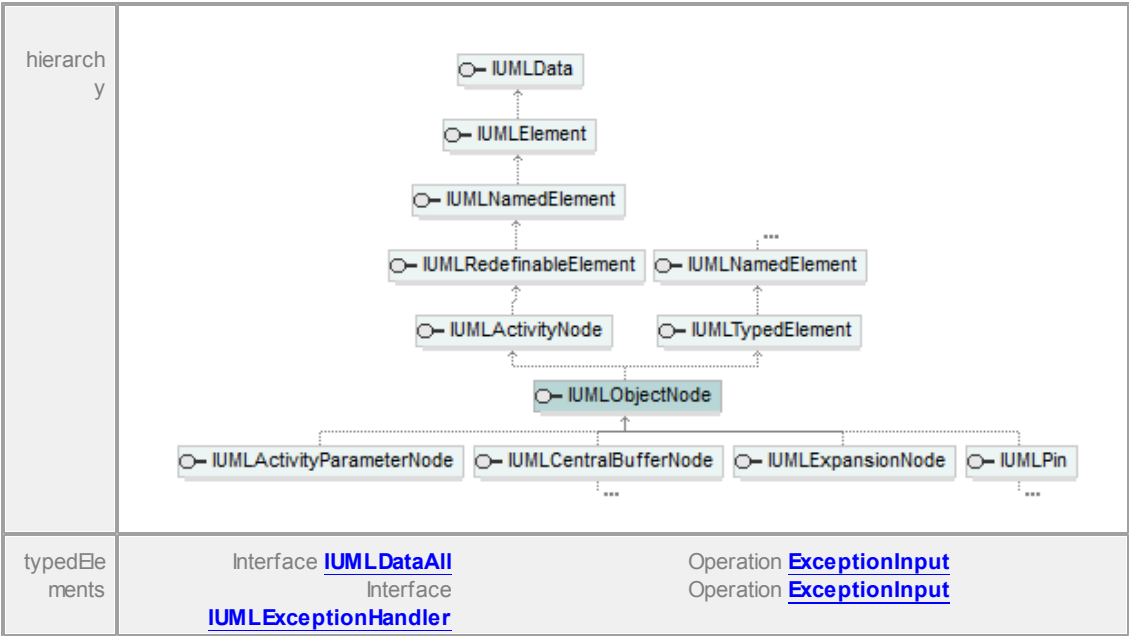
UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLObjectNode

Interface **IUMLObjectNode**





Operation **IUMLObjectNode::EraseInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	void			

Operation **IUMLObjectNode::ExceptionHandlers**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLExceptionHandler .					

Operation **IUMLObjectNode::InsertInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLState			
	return	return	void			

Operation **IUMLObjectNode::InStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLState .					

Operation **IUMLObjectNode::IsControlType**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLObjectNode::Ordering**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLObjectNodeOrderingKind			

Operation **IUMLObjectNode::Selection**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLObjectNode::UpperBound**

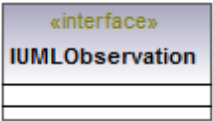
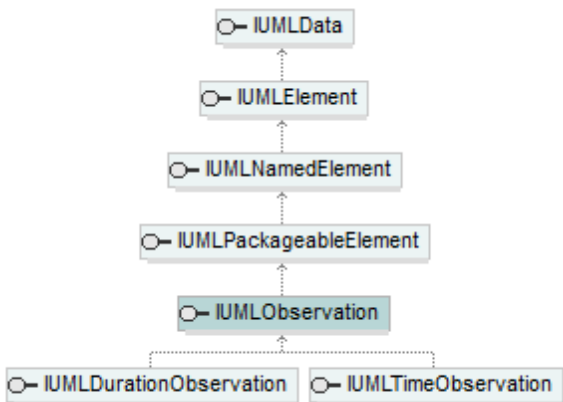
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLObservation

Interface **IUMLObservation**

diagram					
hierarchy					
typedElements	Interface IUMLDataAll Interface IUMLDuration Interface IUMLTimeExpression Operation InsertObservationAt Operation InsertObservationAt Operation InsertObservationAt				

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLOccurrenceSpecification

Interface **IUMLOccurrenceSpecification**

diagram	The diagram shows the IUMLOccurrenceSpecification interface with four methods: <code>OccurringEvent():IUMLEvent</code>, <code>ExecutionSpecificationStart():IUMLExecutionSpecification</code>, <code>ExecutionSpecificationFinish():IUMLExecutionSpecification</code>, and <code>Covered():IUMLLifeline</code>. Below the interface, a hierarchy of classes is shown: IUMLData (parent of IUMLElement), IUMLElement (parent of IUMLNamedElement), IUMLNamedElement (parent of IUMLInteractionFragment), IUMLInteractionFragment (parent of IUMLOccurrenceSpecification), and IUMLMessageOccurrenceSpecification (parent of IUMLOccurrenceSpecification).		
typedElements	Interface IUMLDataAll Interface IUMLExecutionSpecification	Operation Finish Start Operation Finish Start	

Operation **IUMLOccurrenceSpecification::Covered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLLifeline			

Operation **IUMLOccurrenceSpecification::ExecutionSpecificationFinish**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLExecutionSpecification			

Operation **IUMLOccurrenceSpecification::ExecutionSpecificationStart**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLExecutionSpecification			

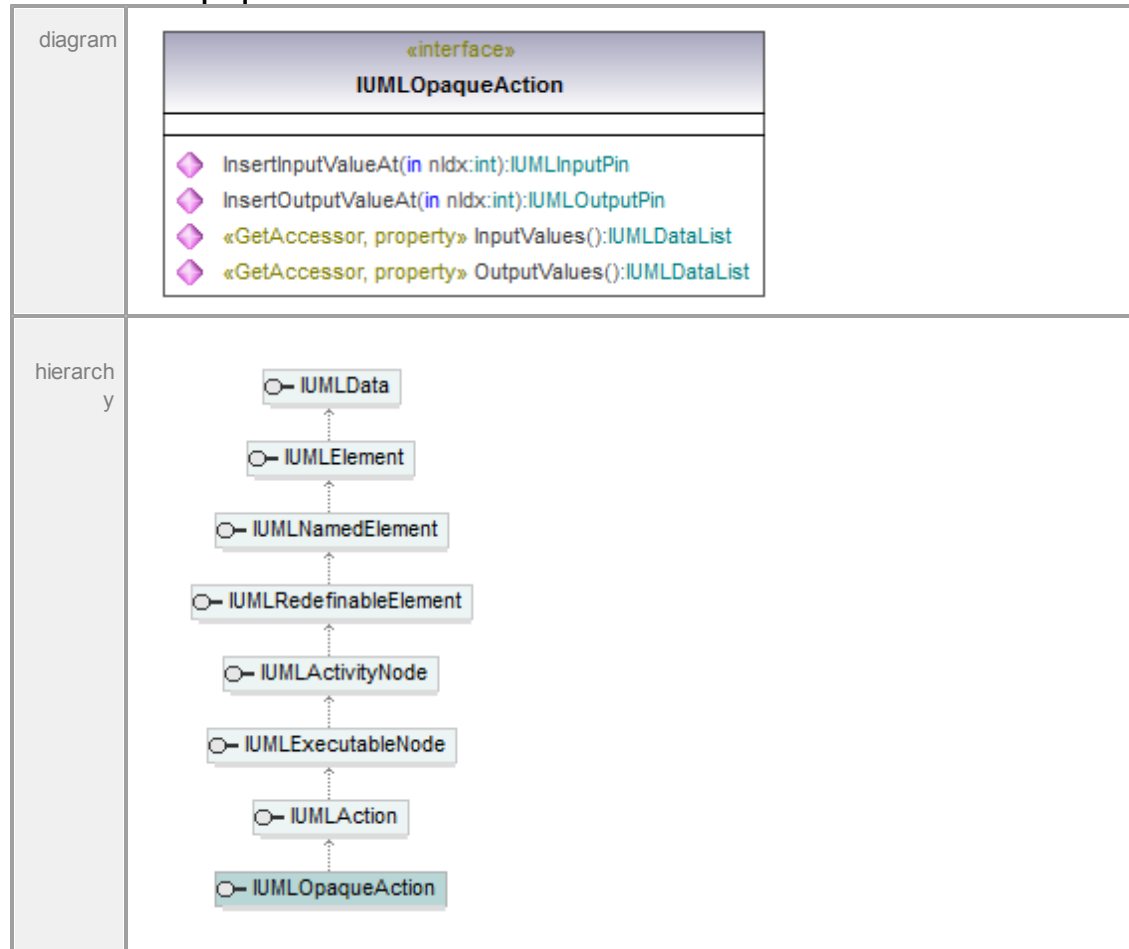
Operation **IUMLOccurrenceSpecification::OccurringEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLEvent			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLOpaqueAction

Interface **IUMLOpaqueAction**Operation **IUMLOpaqueAction::InputValues**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

Operation **IUMLOpaqueAction::InsertInputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	IUMLInputPin			

Operation **IUMLOpaqueAction::InsertOutputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	IUMLOutputPin			

Operation **IUMLOpaqueAction::OutputValues**

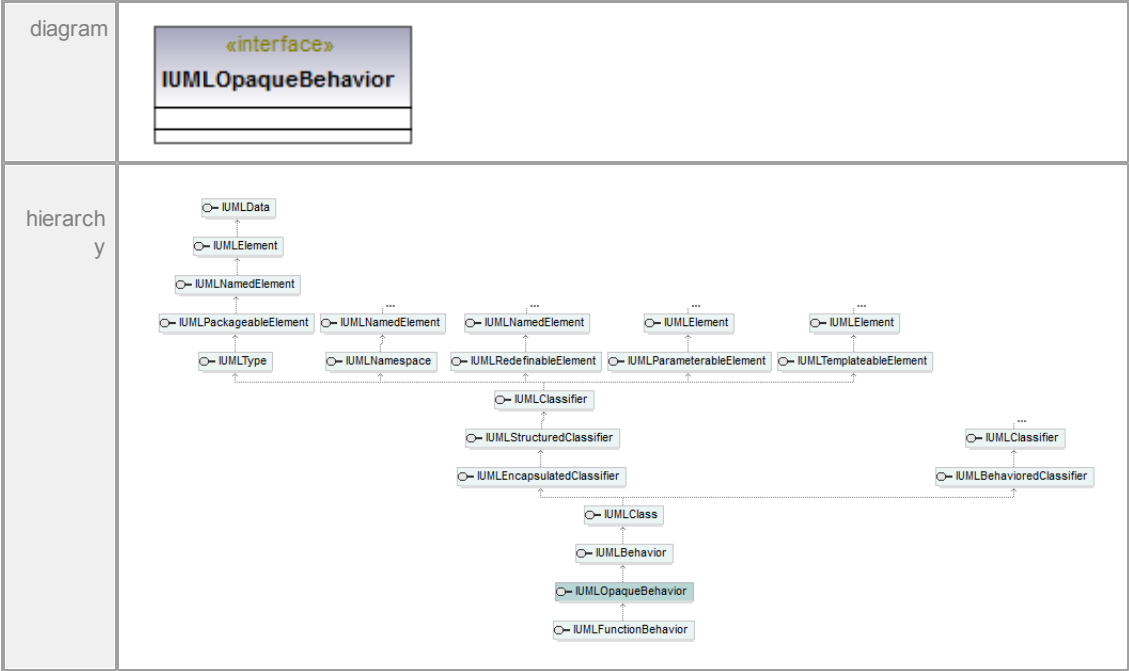
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLOpaqueBehavior

Interface IUMLOpaqueBehavior



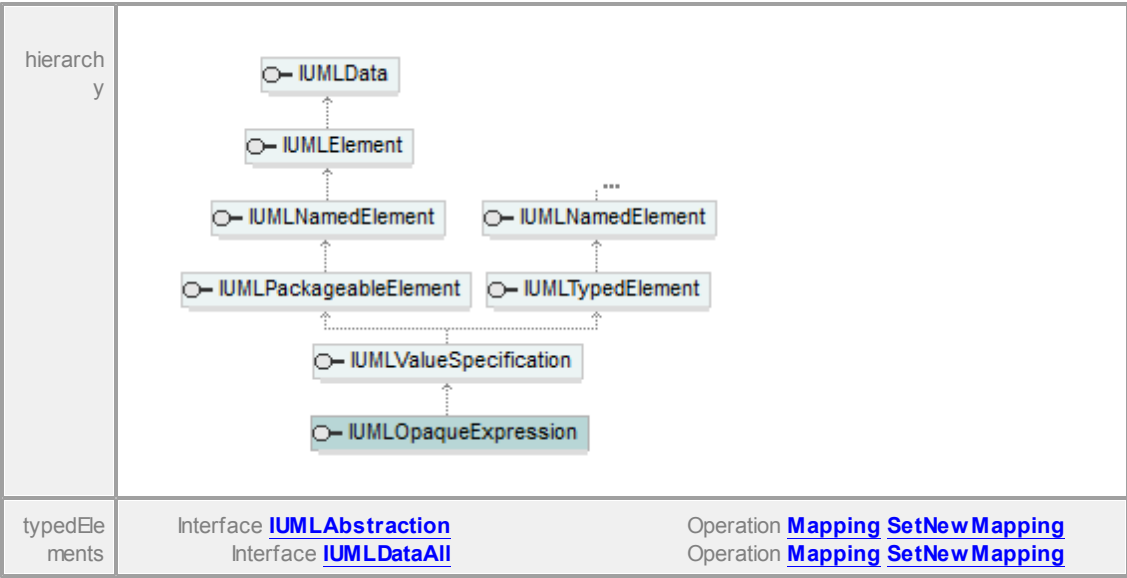
UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLOpaqueExpression

Interface IUMLOpaqueExpression





Operation **IUMLOpaqueExpression::Body**

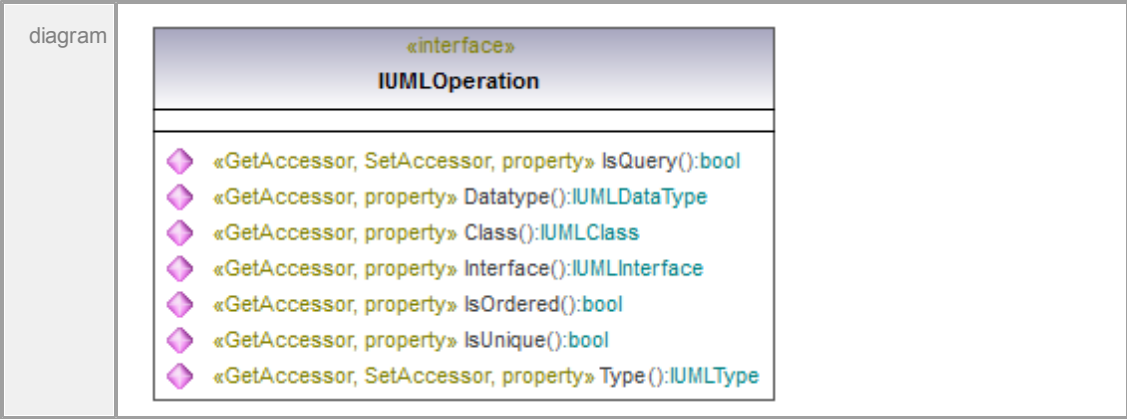
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

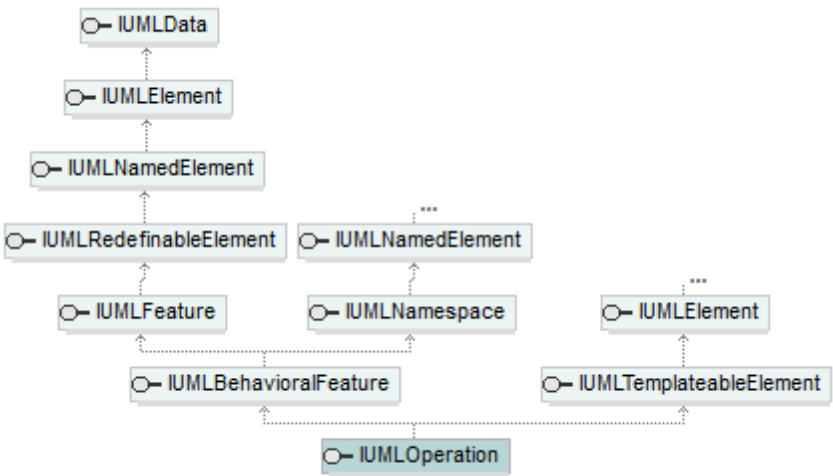
Operation **IUMLOpaqueExpression::Language**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

UModelAPI - IUMLOperation

Interface **IUMLOperation**



hierarchy		
typedElements	Interface IDocument Interface IUMLArtifact Interface IUMLCallEvent Interface IUMLCallOperationAction Interface IUMLClass Interface IUMLDataAll Interface IUMLDataType Interface IUMLGuiSequenceDiagram Interface IUMLInterface Interface IUMLMessage Interface IUMLParameter	Operation GenerateSequenceDiagram Operation InsertOwnedOperationAt Operation Operation Operation CallOperation Operation InsertOwnedOperationAt Operation CallOperation CodeOperation GetOperation InsertOwnedOperationAt Operation SetOperation Operation InsertOwnedOperationAt Operation CodeOperation Operation InsertOwnedOperationAt Operation GetOperation SetOperation Operation Operation

Operation **IUMLOperation::Class**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLClass			

Operation **IUMLOperation::Datatype**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataType			

Operation **IUMLOperation::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLInterface			

Operation **IUMLOperation::IsOrdered**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLOperation::IsQuery**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLOperation::IsUnique**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

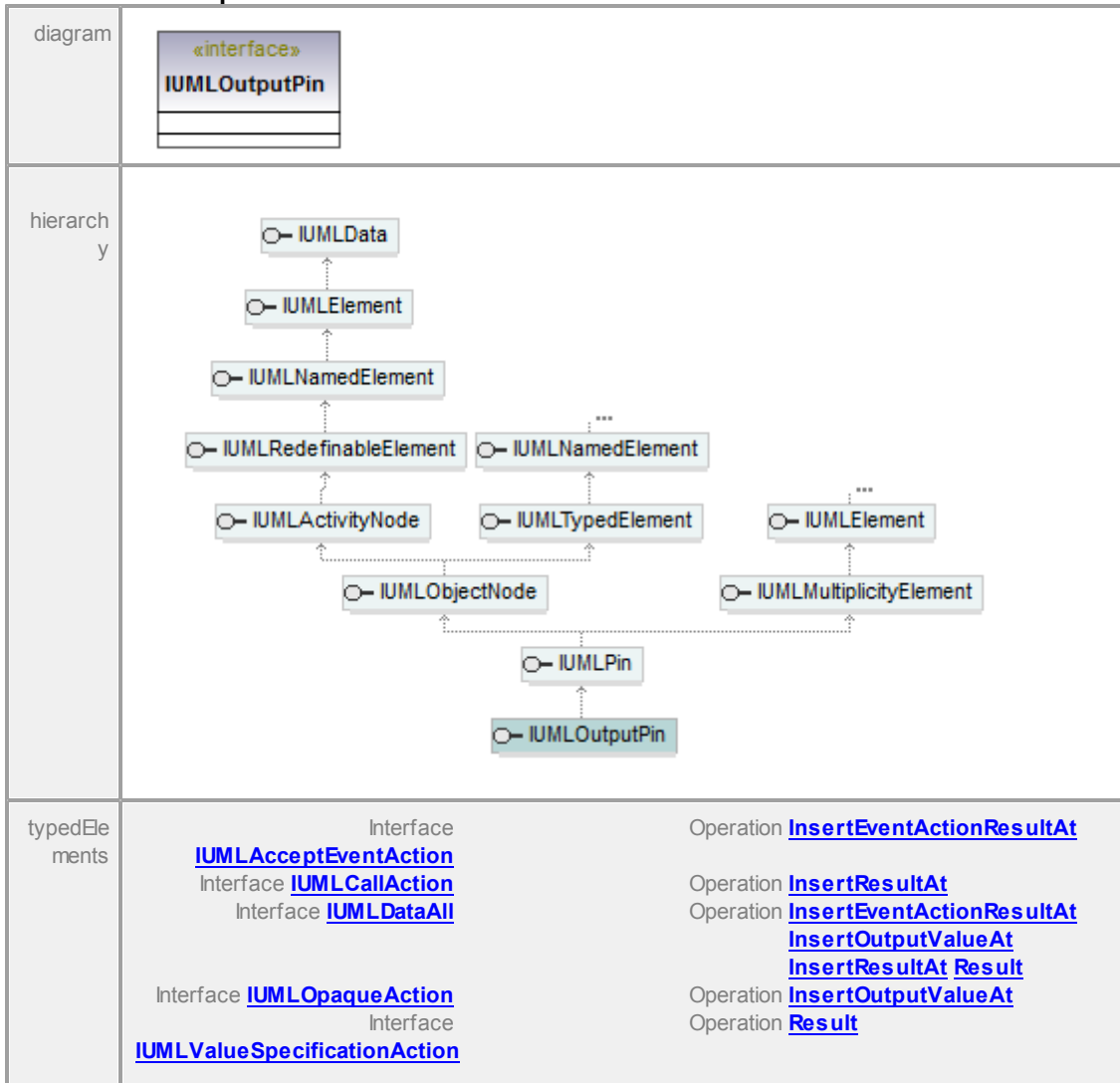
Operation **IUMLOperation::Type**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLType			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLOutputPin

Interface **IUMLOutputPin**

UML documentation generated by UModel UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLPackage

Interface IUMLPackage

diagram	<pre> «interface» IUMLPackage InsertPackagedElementAt(in nldx:int, in strKind:string):IUMLPackageableElement InsertPackagedElementRelationshipAt(in nldx:int, in strKind:string, in ipFrom:IUMLElement, in ipTo:IUMLElement):IUMLPackageableElement InsertProfileApplicationAt(in nldx:int, in ipAppliedProfile:IUMLProfile):IUMLProfileApplication IsCodeLangNamespace(in nCodeLang:ENUMCodeLang):bool IsCodeLangNamespaceRoot(in nCodeLang:ENUMCodeLang):bool «GetAccessor, property» PackagedElements():IUMLDataList «GetAccessor, property» OwnedTypes():IUMLDataList «GetAccessor, property» NestedPackages():IUMLDataList «GetAccessor, property» NestingPackage():IUMLPackage «GetAccessor, property» ProfileApplications():IUMLDataList «GetAccessor, property» IsShared():bool «GetAccessor, property» OwnedStereotypes():IUMLDataList </pre>	
hierarchy	<pre> graph TD IUMLModel -- > IUMLPackage IUMLProfile -- > IUMLPackage IUMLPackageableElement -- > IUMLPackage IUMLNamespace -- > IUMLPackage IUMLNamedElement -- > IUMLPackageableElement IUMLData -- > IUMLNamedElement IUMLElement -- > IUMLNamedElement IUMLPackageableElement -- > IUMLData </pre>	
typedElements	Interface IDocument Interface IImportSourceDig Interface IModelTransformationDig Interface IUMLDataAll Interface IUMLNamespace Interface IUMLPackage Interface IUMLPackageableElement Interface IUMLPackageImport Interface IUMLPackageMerge	Operation RootPackage Operation ImportTarget Operation SourcePackage Operation TargetPackage Operation ApplyingPackage Operation ImportedPackage Operation InsertPackageImportAt Operation InsertPackageMergeAt Operation MergedPackage Operation NestingPackage Operation OwningPackage Package Operation ReceivingPackage Operation InsertPackageImportAt Operation InsertPackageMergeAt Operation NestingPackage Operation OwningPackage Operation ImportedPackage Operation MergedPackage Operation ReceivingPackage

	Interface IUMLProfileApplication Interface IUMLType	Operation ApplyingPackage Operation Package

Operation **IUMLPackage::InsertPackagedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	strKind	in	string			
	return	return	IUMLPackageableElement			

Operation **IUMLPackage::InsertPackagedElementRelationshipAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	strKind	in	string			
	ipFrom	in	IUMLElement			
	ipTo	in	IUMLElement			
	return	return	IUMLPackageableElement			

Operation **IUMLPackage::InsertProfileApplicationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipAppliedProfile	in	IUMLProfile			
	return	return	IUMLProfileApplication			

Operation **IUMLPackage::IsCodeLangNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	in	ENUMCodeLanguage			
	return	return	bool			

Operation **IUMLPackage::IsCodeLangNamespaceRoot**

parameter	name	direction	type	type modifier	multiplicity	default
	nCodeLang	in	ENUMCodeLanguage			
	return	return	bool			

Operation **IUMLPackage::IsShared**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLPackage::NestedPackages**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLPackage .					

Operation **IUMLPackage::NestingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLPackage			

Operation **IUMLPackage::OwnedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLPackage::OwnedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLType .					

Operation **IUMLPackage::PackedElements**

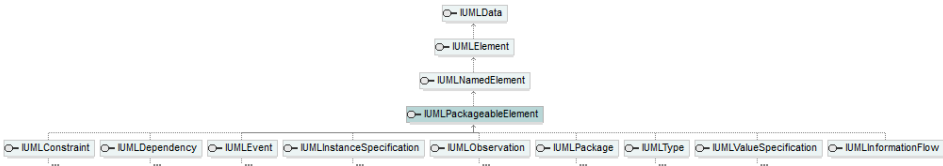
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLPackageableElement .					

Operation **IUMLPackage::ProfileApplications**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLProfileApplication .					

UModelAPI - IUMLPackageableElement

Interface **IUMLPackageableElement**

diagram		
hierarchy		
typedElements	Interface IUMLArtifact Interface IUMLDataAll	Operation InsertManifestationAt Operation ImportedElement

	Interface IUMLElementImport Interface IUMLManifestation Interface IUMLNamespace Interface IUMLPackage	Operation InsertElementImportAt Operation InsertManifestationAt Operation InsertPackagedElementAt Operation InsertPackagedElementRelationshipAt Operation UtilizedElementImportedElement Operation UtilizedElement Operation InsertElementImportAt Operation InsertPackagedElementAt Operation InsertPackagedElementRelationshipAt

Operation [IUMLPackageableElement::OwningPackage](#)

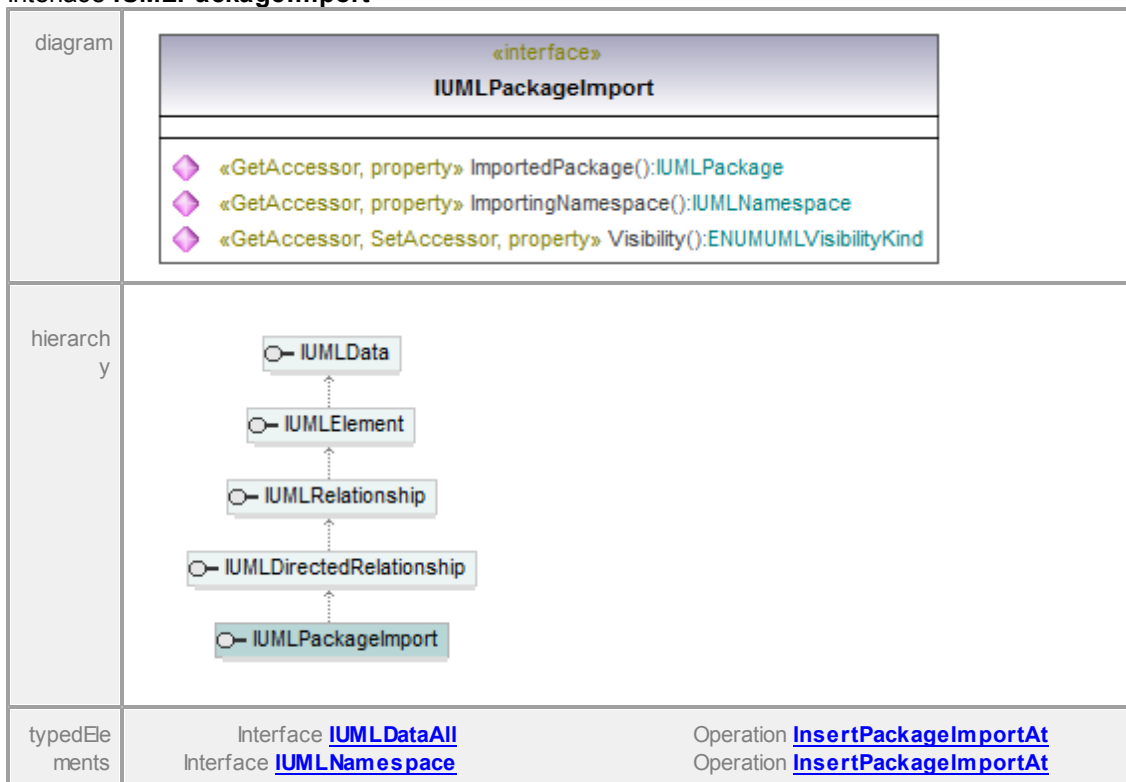
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLPackage			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLPackageImport

Interface [IUMLPackageImport](#)



Operation [IUMLPackageImport::ImportedPackage](#)

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLPackage			

Operation [IUMLPackageImport::ImportingNamespace](#)

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLNamespace			

Operation **IUMLPackageImport::Visibility**

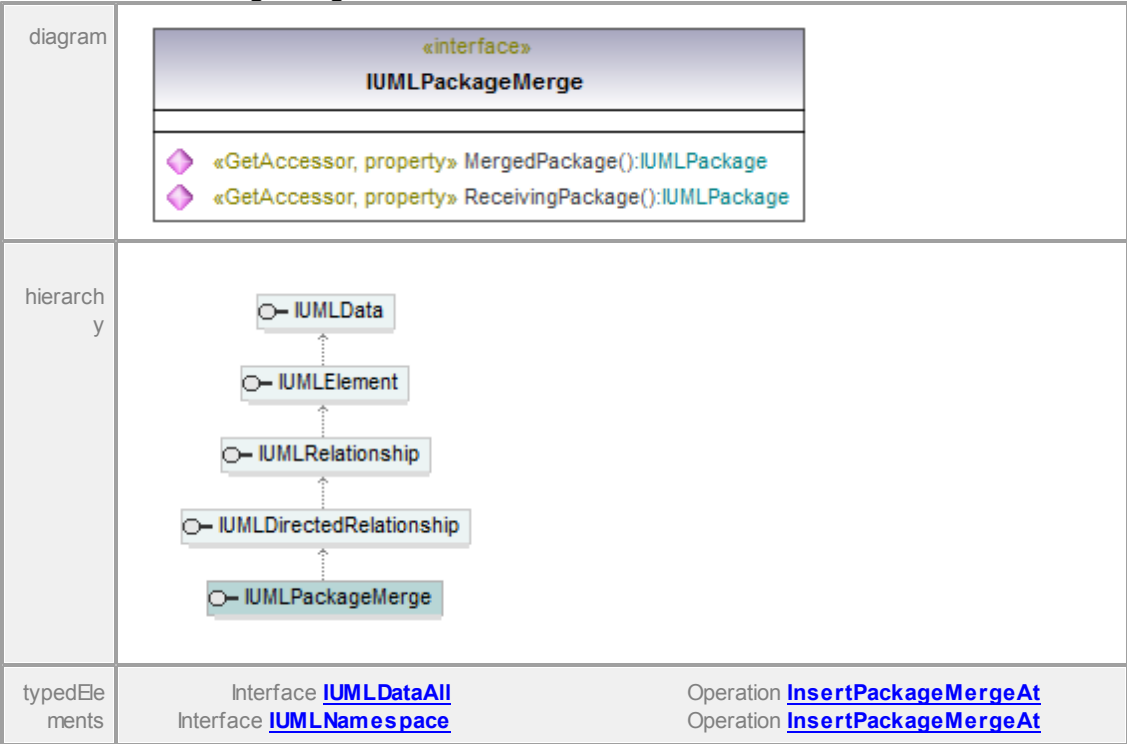
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ENUMUMLVisibilityKind			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLPackageMerge

Interface **IUMLPackageMerge**



Operation **IUMLPackageMerge::MergedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLPackage			

Operation **IUMLPackageMerge::ReceivingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLPackage			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLParameter

Interface **IUMLParameter**Operation **IUMLParameter::Default**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLParameter::DefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLValueSpecification			

Operation **IUMLParameter::Direction**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLParameterDirection			

Operation **IUMLParameter::IsVarArgList**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLParameter::Operation**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLOperation			

Operation **IUMLParameter::SetNewDefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	return	return	IUMLValueSpecification			

Operation **IUMLParameter::SetNewDefaultValueInstanceValue**

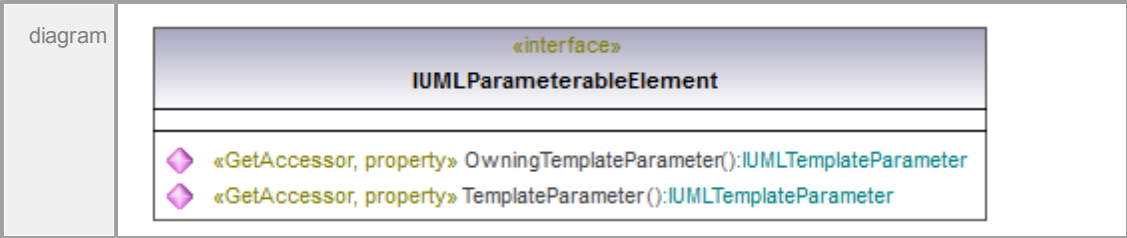
parameter	name	direction	type	type modifier	multiplicity	default
r	ipInstance	in	IUMLInstanceSpecification			
	return	return	IUMLInstanceValue			

Operation **IUMLParameter::SetNewDefaultValueLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
r	strNewVal	in	string			
	return	return	IUMLLiteralString			

UModelAPI - IUMLParameterableElement

Interface **IUMLParameterableElement**



hierarchy			
typedElements	Interface IUMLDataAll Interface IUMLTemplateBinding Interface IUMLTemplateParameter Interface IUMLTemplateParameterSubstitution	Operation Actual InsertParameterSubstitution At OwnedActual OwnedParameteredElement ParameteredElement SetNew OwnedParameteredElement Operation InsertParameterSubstitution At Operation OwnedParameteredElement ParameteredElement SetNew OwnedParameteredElement Operation Actual OwnedActual	

Operation IUMLParameterableElement::OwningTemplateParameter

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLTemplateParameter			

Operation IUMLParameterableElement::TemplateParameter

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLTemplateParameter			

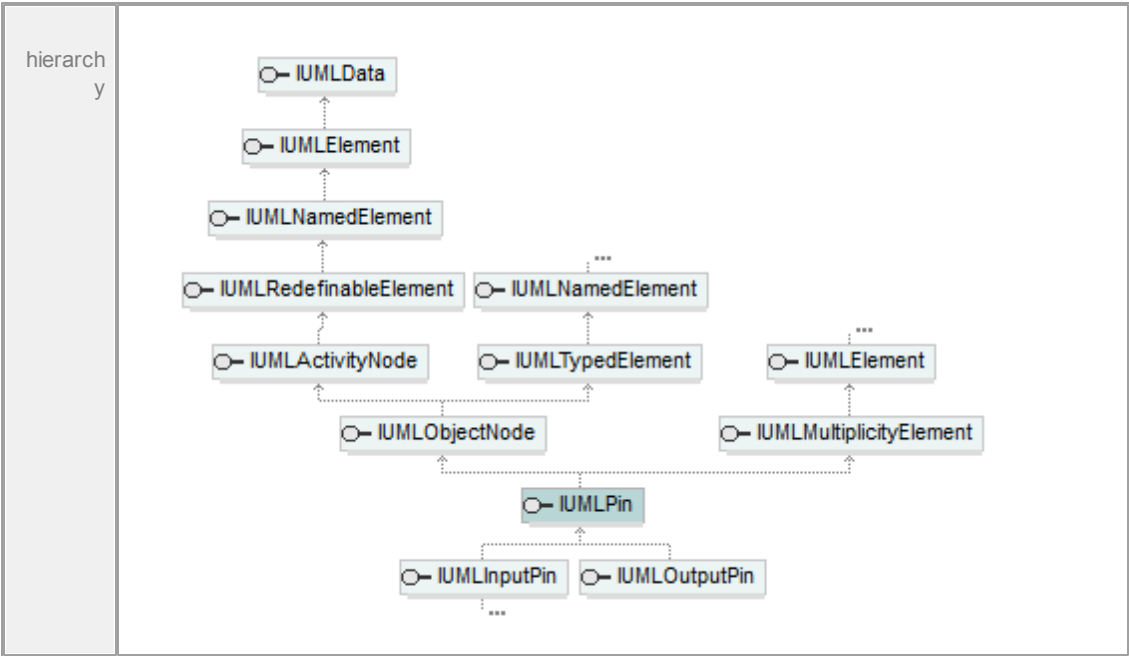
UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLPin

Interface IUMLPin

diagram	

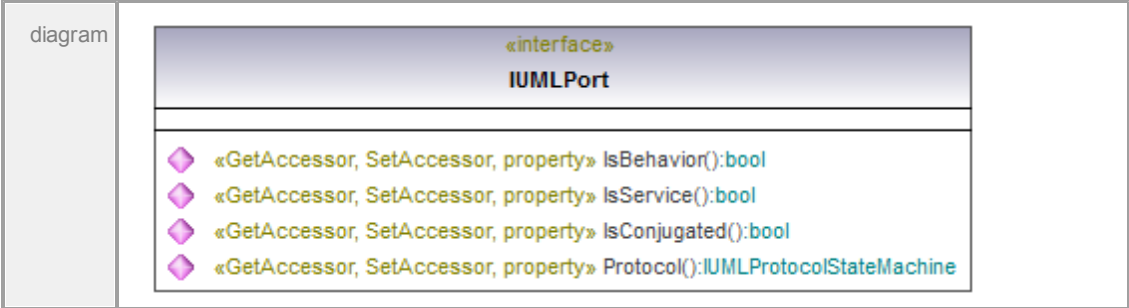


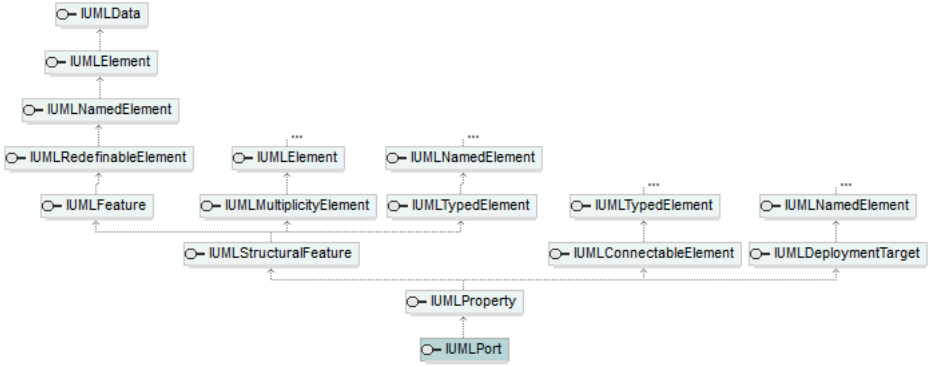
Operation **IUMLPin::IsControl**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

UModelAPI - IUMLPort

Interface **IUMLPort**



hierarchy		
typedElements	Interface IUMLDataAll Interface IUMLStructuredClassifier	Operation InsertOwnedPortAt Operation InsertOwnedPortAt

Operation **IUMLPort::IsBehavior**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLPort::IsConjugated**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLPort::IsService**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

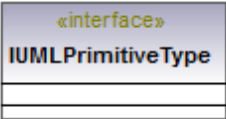
Operation **IUMLPort::Protocol**

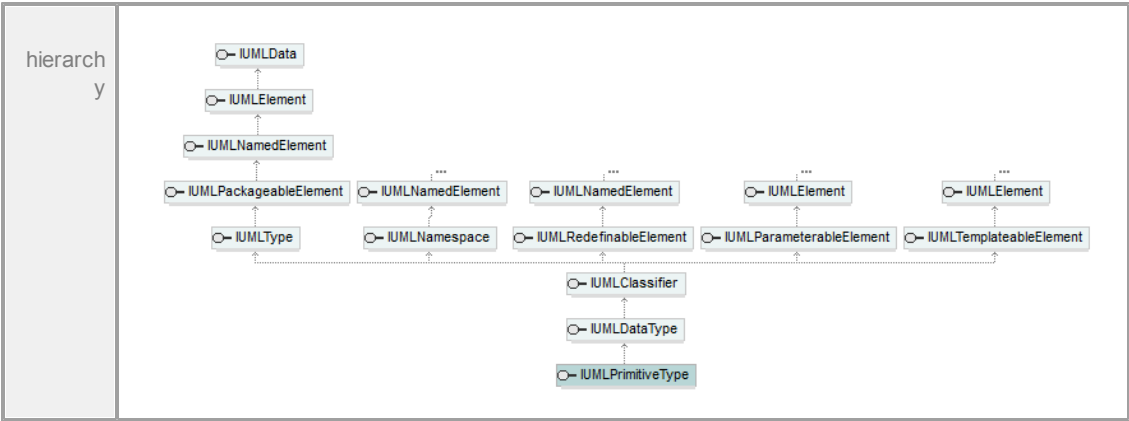
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLProtocolStateMachine			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLPrimitiveType

Interface **IUMLPrimitiveType**

diagram	



UML documentation generated by [UModel UML Editor http://www.altova.com/umodel](http://www.altova.com/umodel) 08/03/15 13:42:06

UModelAPI - IModelProfile

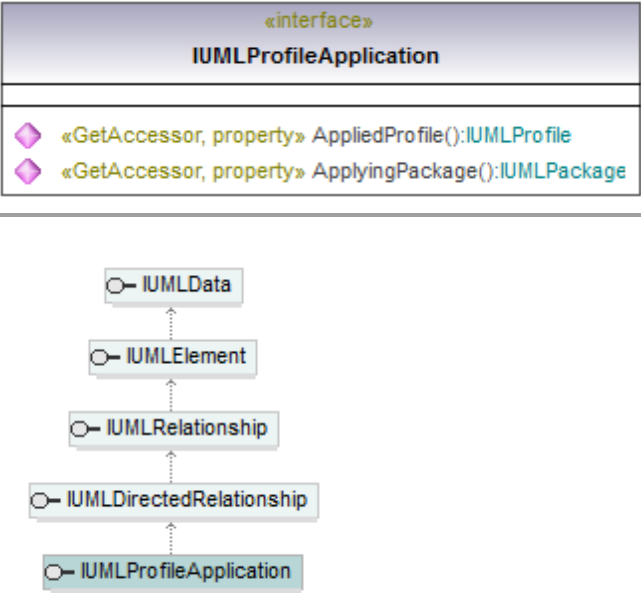
Interface **IModelProfile**

diagram		
hierarchy		
typedElements	Interface IModelDataAll Interface IModelPackage Interface IModelProfileApplication	Operation AppliedProfile Operation InsertProfileApplicationAt Operation InsertProfileApplicationAt Operation AppliedProfile

UML documentation generated by [UModel UML Editor http://www.altova.com/umodel](http://www.altova.com/umodel) 08/03/15 13:42:06

UModelAPI - IUMLProfileApplication

Interface **IUMLProfileApplication**

diagram	 The diagram shows the IUMLProfileApplication interface with two methods: <code>AppliedProfile():IUMLProfile</code> and <code>ApplyingPackage():IUMLPackage</code>. Below it, a hierarchy of classes is shown: IUMLData (parent of IUMLElement), IUMLRelationship (parent of IUMLDirectedRelationship), and IUMLProfileApplication (parent of IUMLDirectedRelationship).	
typedElements	Interface IUMLDataAll Interface IUMLPackage	Operation InsertProfileApplicationAt Operation InsertProfileApplicationAt

Operation **IUMLProfileApplication::AppliedProfile**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLProfile			

Operation **IUMLProfileApplication::ApplyingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLPackage			

UModelAPI - IUMLProperty

Interface **IUMLProperty**

diagram	«interface» IUMLProperty ◆ InsertQualifierAt(in nldx:int):IUMLProperty ◆ SetNewDefaultValue(in strKind:string):IUMLValueSpecification ◆ SetNewDefaultValueLiteralString(in strNewVal:string):IUMLLiteralString ◆ SetNewDefaultValueInstanceValue(in iplInstance:IUMLInstanceSpecification):IUMLInstanceValue ◆ «GetAccessor, SetAccessor, property» IsDerived():bool ◆ «GetAccessor, SetAccessor, property» IsDerivedUnion():bool ◆ «GetAccessor, property» IsComposite():bool ◆ «GetAccessor, property» Opposite():IUMLProperty ◆ «GetAccessor, property» Association():IUMLAssociation ◆ «GetAccessor, property» OwningAssociation():IUMLAssociation ◆ «GetAccessor, SetAccessor, property» IsNavigable():bool ◆ «GetAccessor, SetAccessor, property» IsOwnedEnd():bool ◆ «GetAccessor, property» AssociationEnd():IUMLProperty ◆ «GetAccessor, property» Qualifiers():IUMLDataList ◆ «GetAccessor, SetAccessor, property» Aggregation():ENUMUMLAggregationKind ◆ «GetAccessor, property» OwningSignal():IUMLSignal ◆ «GetAccessor, property» Datatype():IUMLDataType ◆ «GetAccessor, property» Class():IUMLClass ◆ «GetAccessor, property» Interface():IUMLInterface ◆ «GetAccessor, property» DefaultValue():IUMLValueSpecification ◆ «GetAccessor, SetAccessor, property» Default():string ◆ «GetAccessor, property» Classifier():IUMLClassifier	
hierarchy	<pre> graph BT IUMLPort -- > IUMLProperty IUMLStructuralFeature -- > IUMLProperty IUMLStructuralFeature -- > IUMLFeature IUMLStructuralFeature -- > IUMLMultiplicityElement IUMLStructuralFeature -- > IUMLTypedElement IUMLFeature -- > IUMLNamedElement IUMLFeature -- > IUMLRedefinableElement IUMLMultiplicityElement -- > IUMLNamedElement IUMLMultiplicityElement -- > IUMLTypedElement IUMLTypedElement -- > IUMLNamedElement IUMLTypedElement -- > IUMLConnectableElement IUMLConnectableElement -- > IUMLNamedElement IUMLConnectableElement -- > IUMLDeploymentTarget IUMLNamedElement -- > IUMLData IUMLNamedElement -- > IUMLElement IUMLElement -- > IUMLData IUMLElement -- > IUMLElement </pre>	
typedElements	Interface IUMLArtifact Interface IUMLDataAll Interface IUMLDataType Interface IUMLInterface Interface IUMLProperty	Operation InsertOwnedAttributeAt Operation AssociationEnd Operation InsertOwnedAttributeAt Operation InsertQualifierAt Opposite Operation OwningProperty Operation InsertOwnedAttributeAt Operation InsertOwnedAttributeAt Operation AssociationEnd Operation InsertQualifierAt Opposite

	Interface IUMLSignal	Operation InsertOwnedAttributeAt
	Interface IUMLStructuredClassifier	Operation InsertOwnedAttributeAt
	Interface IUMLValueSpecification	Operation OwningProperty

Operation **IUMLProperty::Aggregation**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLAggregationKind			

Operation **IUMLProperty::Association**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLAssociation			

Operation **IUMLProperty::AssociationEnd**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLProperty			

Operation **IUMLProperty::Class**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLClass			

Operation **IUMLProperty::Classifier**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLClassifier			

Operation **IUMLProperty::Datatype**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDatatype			

Operation **IUMLProperty::Default**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLProperty::DefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLValueSpecification			

Operation **IUMLProperty::InsertQualifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	idx	in	int			
	return	return	IUMLProperty			

Operation **IUMLProperty::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLInterface			

Operation **IUMLProperty::IsComposite**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLProperty::IsDerived**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLProperty::IsDerivedUnion**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLProperty::IsNavigable**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLProperty::IsOwnedEnd**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLProperty::Opposite**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLProperty			

Operation **IUMLProperty::OwningAssociation**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLAssociation			

Operation **IUMLProperty::OwningSignal**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLSignal			

Operation **IUMLProperty::Qualifiers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLProperty .					

Operation **IUMLProperty::SetNewDefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			

	return	return	IUMLValueSpecification			

Operation **IUMLProperty::SetNewDefaultValueInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
	ipInstance	in	IUMLInstanceSpecification			
	return	return	IUMLInstanceValue			

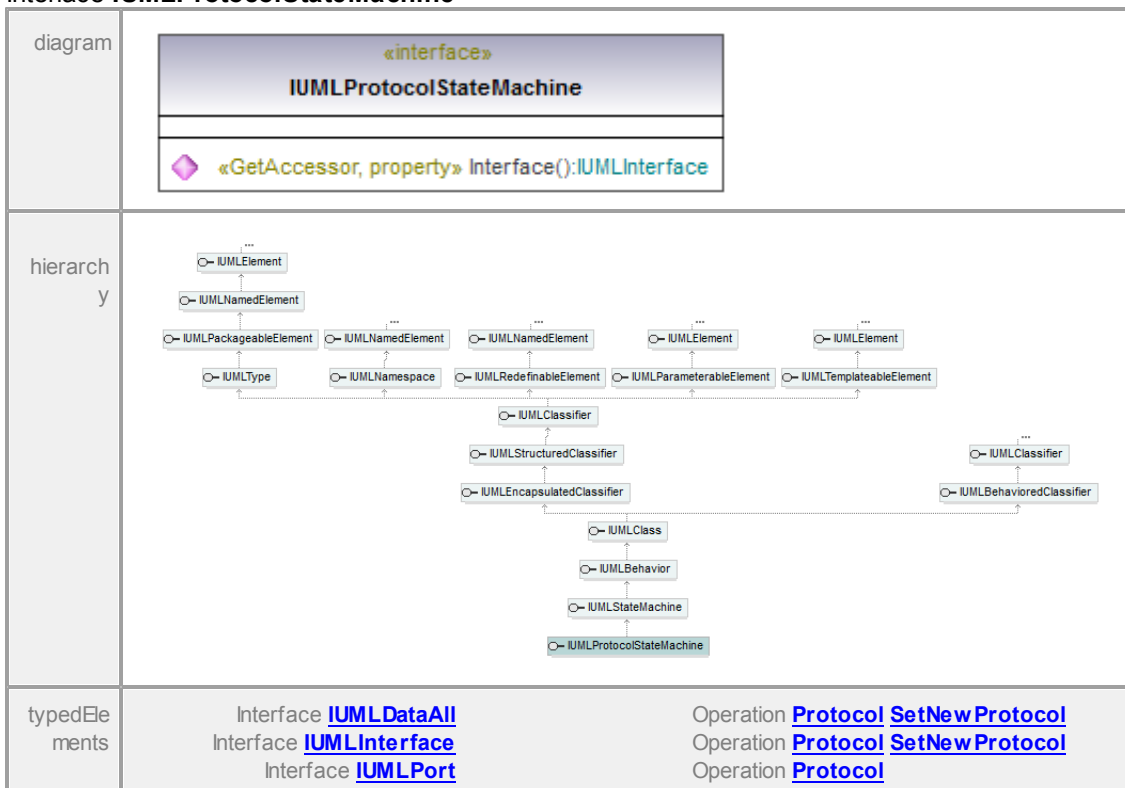
Operation **IUMLProperty::SetNewDefaultValueLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
	strNewVal	in	string			
	return	return	IUMLLiteralString			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLProtocolStateMachine

Interface **IUMLProtocolStateMachine**Operation **IUMLProtocolStateMachine::Interface**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLInterface			

UModelAPI - IUMLProtocolTransition

Interface **IUMLProtocolTransition**

diagram	«interface» IUMLProtocolTransition ◆ SetNewPreCondition(in strKind:string):IUMLConstraint ◆ SetNewPostCondition(in strKind:string):IUMLConstraint ◆ «GetAccessor, property» PreCondition():IUMLConstraint ◆ «GetAccessor, property» PostCondition():IUMLConstraint ◆ «GetAccessor, property» Referred():IUMLDataList	
hierarch y	<pre>graph BT IUMLData --> IUMLElement IUMLElement --> IUMLNamedElement IUMLNamedElement --> IUMLNamespace IUMLNamedElement --> IUMLRedefinableElement IUMLNamespace --> IUMLTransition IUMLRedefinableElement --> IUMLTransition IUMLTransition --> IUMLProtocolTransition</pre>	
typedEle ments	Interface IUMLConstraint Interface IUMLDataAll	Operation OwningTransition Operation OwningTransition

Operation **IUMLProtocolTransition::PostCondition**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLConstraint			

Operation **IUMLProtocolTransition::PreCondition**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLConstraint			

Operation **IUMLProtocolTransition::Referred**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

Operation **IUMLProtocolTransition::SetNewPostCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind return	in return	string IUMLConstrai t			

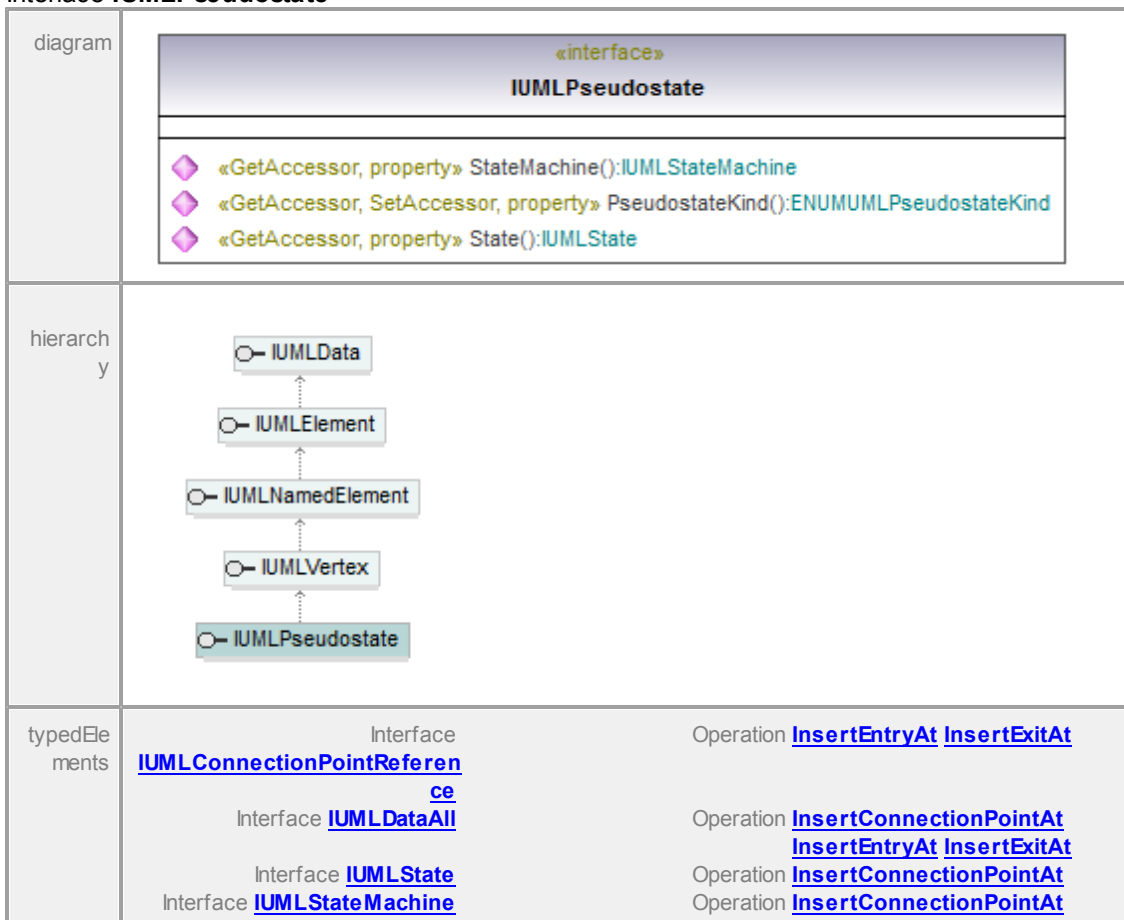
Operation **IUMLProtocolTransition::SetNewPreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind return	in return	string IUMLConstrai t			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLPseudostate

Interface **IUMLPseudostate**Operation **IUMLPseudostate::PseudostateKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ENUMUMLPseu dostateKind			

Operation **IUMLPseudostate::State**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLState			

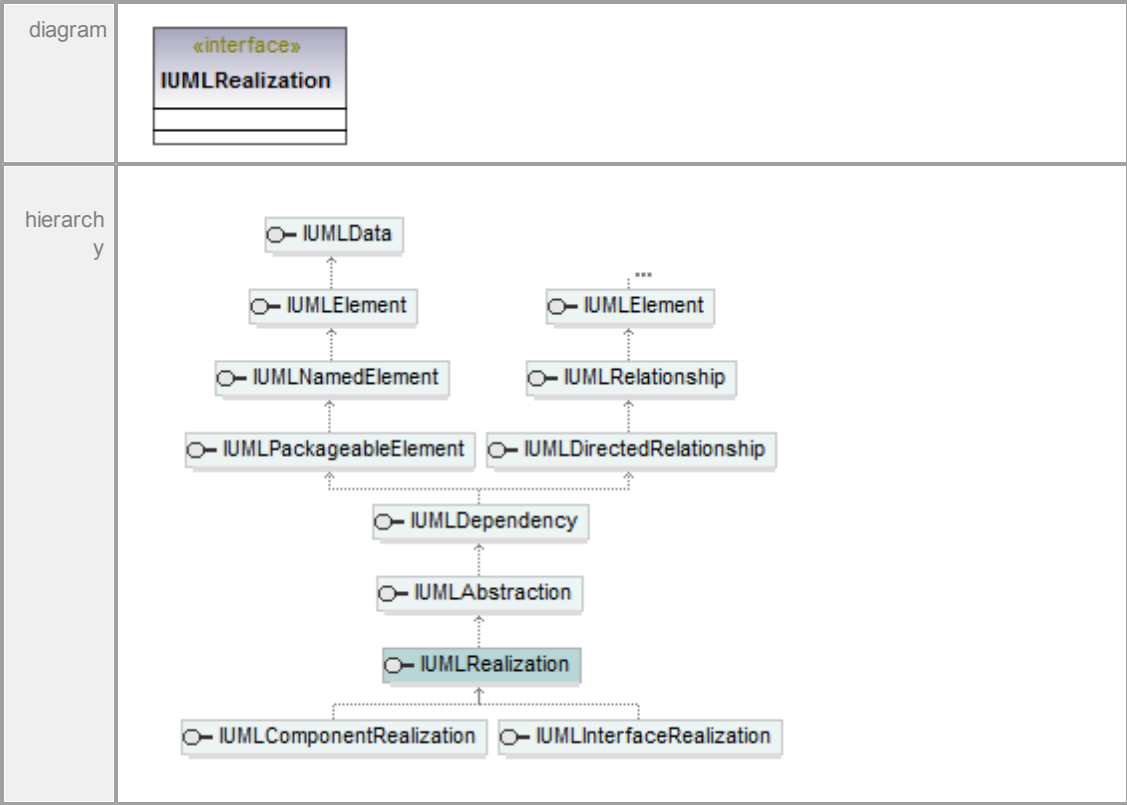
Operation **IUMLPseudostate::StateMachine**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLStateMachine			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLRealization

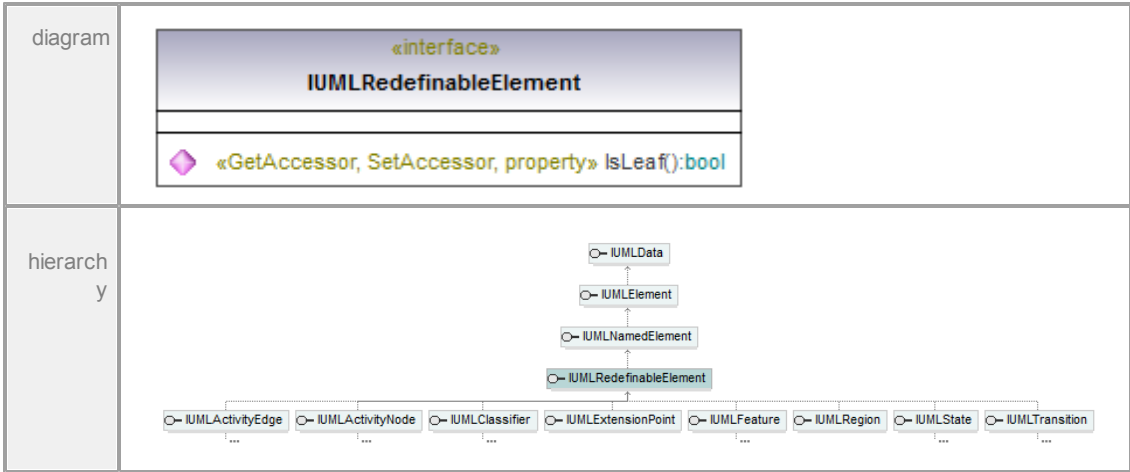
Interface **IUMLRealization**



UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLRedefinableElement

Interface **IUMLRedefinableElement**

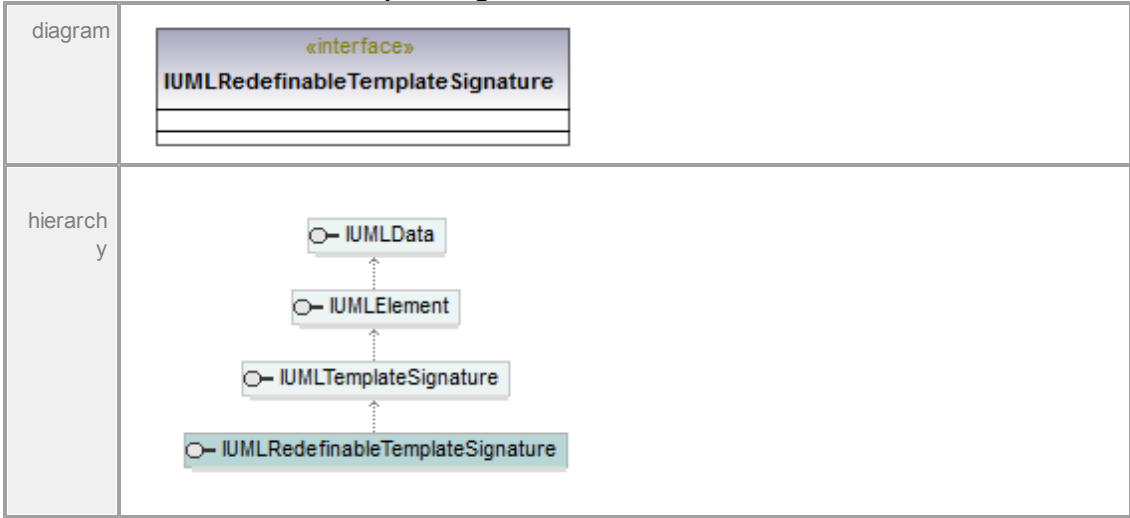


Operation **IUMLRedefinableElement::IsLeaf**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

UModelAPI - IUMLRedefinableTemplateSignature

Interface **IUMLRedefinableTemplateSignature**



UModelAPI - IUMLRegion

Interface **IUMLRegion**

diagram	<pre> classDiagram class IUMLData class IUMLElement class IUMLNamedElement class IUMLRedefinableElement class IUMLNamespace class IUMLRegion IUMLData < -- IUMLElement IUMLElement < -- IUMLNamedElement IUMLNamedElement < -- IUMLRedefinableElement IUMLNamedElement < -- IUMLNamespace IUMLRedefinableElement < -- IUMLRegion IUMLNamespace < -- IUMLRegion </pre>	
typedElements	Interface IUMLDataAll Interface IUMLState Interface IUMLStateMachine Interface IUMLVertex	Operation Container InsertRegionAt Operation InsertRegionAt Operation InsertRegionAt Operation Container

Operation **IUMLRegion::InsertSubVertexAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	strKind	in	string			
	return	return	IUMLVertex			

Operation **IUMLRegion::InsertTransitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipSource	in	IUMLVertex			
	ipTarget	in	IUMLVertex			
	return	return	IUMLTransition			

Operation **IUMLRegion::State**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLState			

Operation **IUMLRegion::StateMachine**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLStateMachine			

Operation IUMLRegion::SubVertices

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLVertex .					

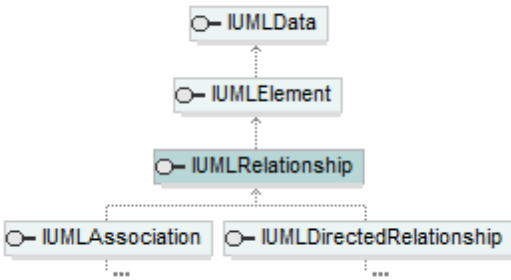
Operation IUMLRegion::Transitions

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLTransition .					

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLRelationship**Interface IUMLRelationship**

diagram						
hierarchy						
typedElements	Interface IUMLDataAll		Operation InsertInformationFlowRealizationAt			
	Interface IUMLInformationFlow		Operation InsertInformationFlowRealizationAt			

Operation IUMLRelationship::RelatedElements

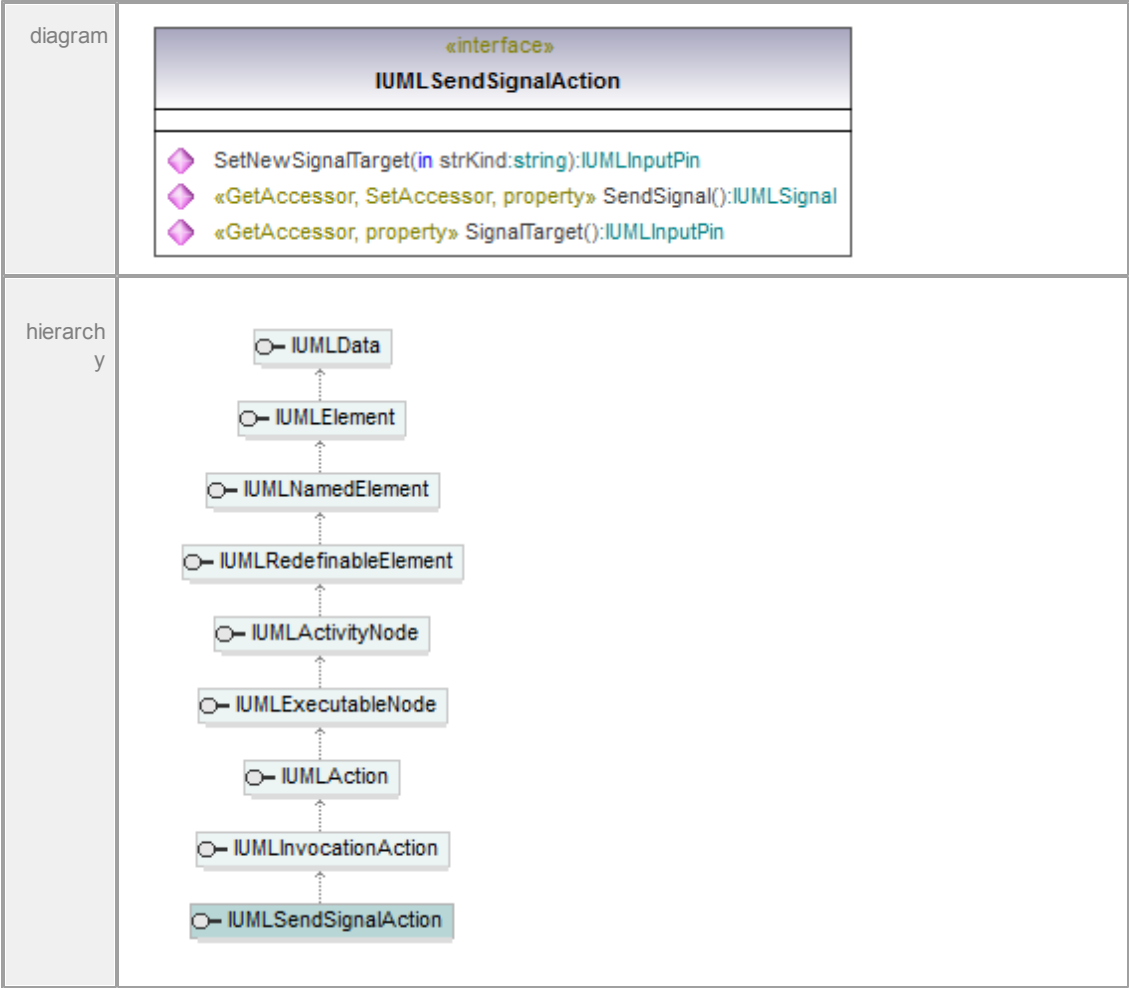
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
document	A list of elements of type IUMLElement .					

tation	

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLSendSignalAction

Interface IUMLSendSignalAction



Operation IUMLSendSignalAction::SendSignal

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLSignal			

Operation IUMLSendSignalAction::SetNewSignalTarget

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	in	string			
	return	return	IUMLInputPin			

Operation IUMLSendSignalAction::SignalTarget

parameter	name	direction	type	type modifier	multiplicity	default

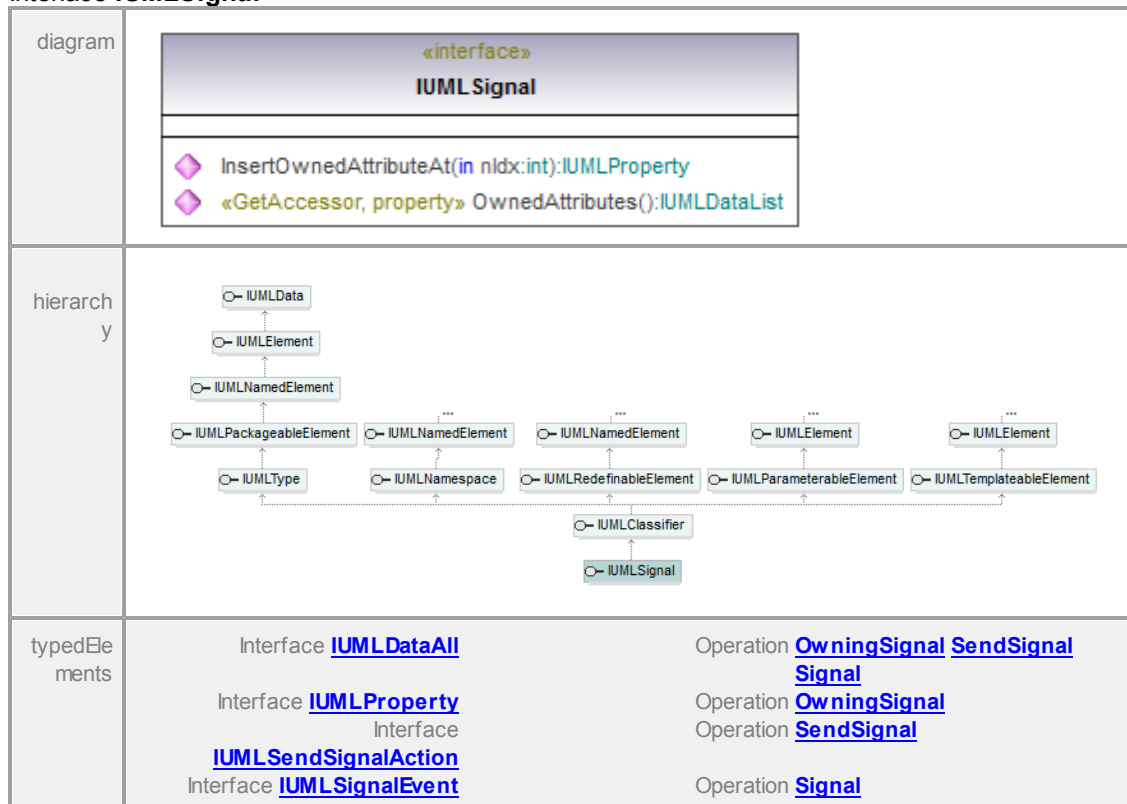
r	return	return	IUMLInputPin

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLSignal

Interface IUMLSignal



Operation IUMLSignal::InsertOwnedAttributeAt

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	IUMLProperty			

Operation IUMLSignal::OwnedAttributes

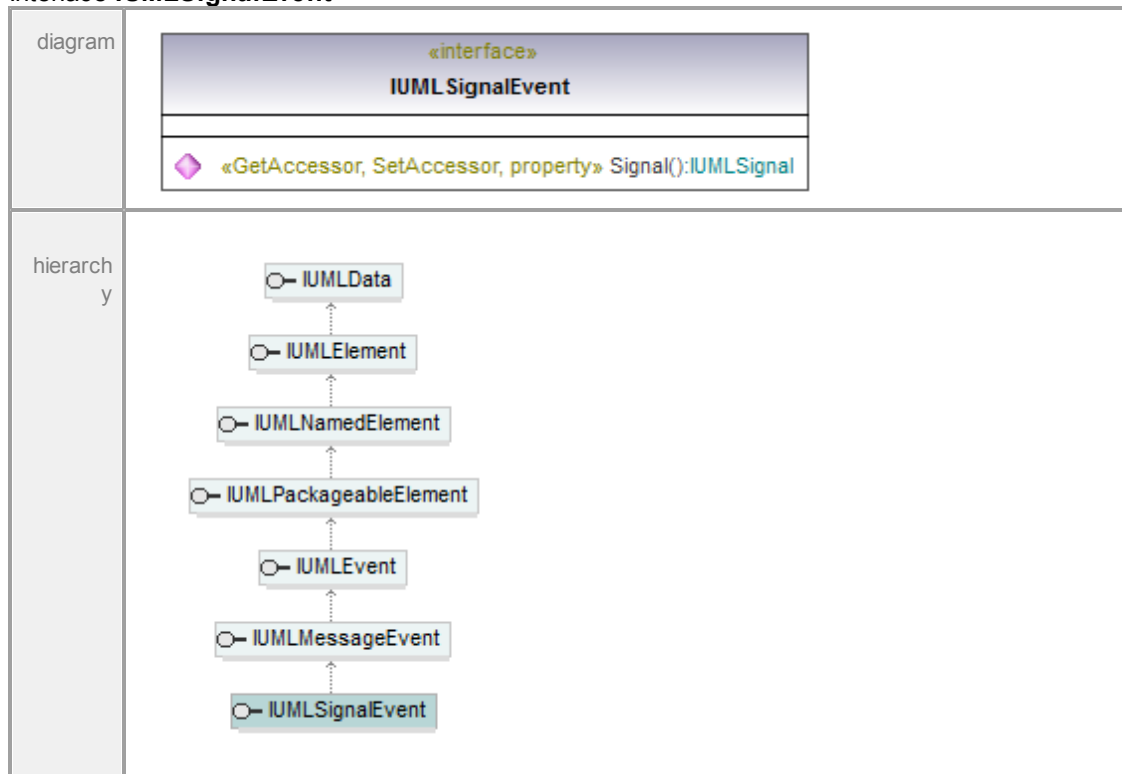
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLProperty .					

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLSignalEvent

Interface IUMLSignalEvent



Operation IUMLSignalEvent::Signal

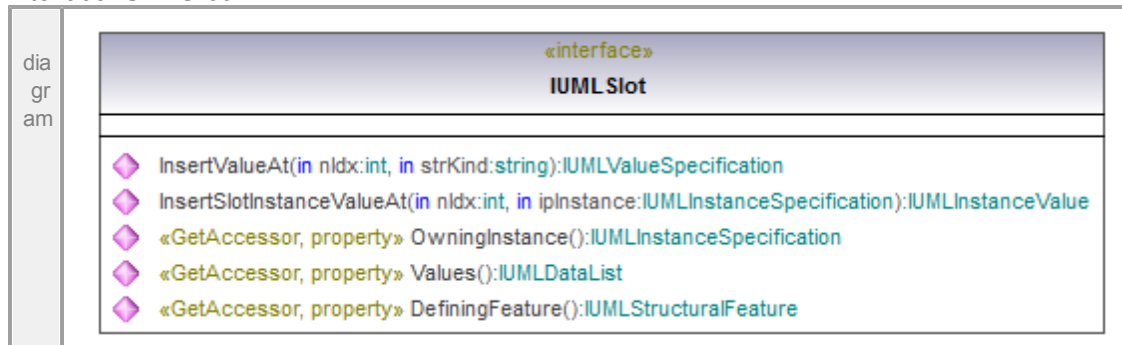
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLSignal			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLSlot

Interface IUMLSlot



hierarchy			
types and elements	Interface IUMLDataAll Interface IUMLInstanceSpecification Interface IUMLValueSpecification	Operation InsertSlotAt OwningSlot Operation InsertSlotAt Operation OwningSlot	

Operation IUMLSlot::DefiningFeature

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLStructuralFeature			

Operation IUMLSlot::InsertSlotInstanceValueAt

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipInstance	in	IUMLInstanceSpecification			
	return	return	IUMLInstanceValue			

Operation IUMLSlot::InsertValueAt

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	strKind	in	string			
	return	return	IUMLValueSpecification			

Operation IUMLSlot::OwningInstance

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLInstanceSpecification			

Operation IUMLSlot::Values

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLValueSpecification .					

UModelAPI - IUMLState

Interface **IUMLState**

diagram		
hierarchy		
typedElements	Interface IUMLConnectionPointReference Interface IUMLConstraint Interface IUMLDataAll Interface IUMLObjectNode Interface IUMLPseudostate Interface IUMLRegion	Operation State Operation OwningState Operation InsertInStateAt OwningState Operation InsertInStateAt Operation State Operation State

Operation **IUMLState::ConnectionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLPseudostate .					

Operation **IUMLState::Connections**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLConnectionPointReference .					

Operation **IUMLState::DoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLState::Entry**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLState::Exit**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLState::InsertConnectionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int IUMLConnectionPointReference			

Operation **IUMLState::InsertConnectionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int IUMLPseudostate			

Operation **IUMLState::InsertRegionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int IUMLRegion			

Operation **IUMLState::IsComposite**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLState::IsOrthogonal**

parameter	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IUMLState::IsSimple**

parameter	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IUMLState::IsSubmachineState**

parameter	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IUMLState::Regions**

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documentation	A list of elements of type IUMLRegion .					

Operation **IUMLState::SetNewDoActivity**

parameter	name strKind return	direction in return	type string IUMLBehavior	type modifier	multiplicity	default

Operation **IUMLState::SetNewEntry**

parameter	name strKind return	direction in return	type string IUMLBehavior	type modifier	multiplicity	default

Operation **IUMLState::SetNewExit**

parameter	name strKind return	direction in return	type string IUMLBehavior	type modifier	multiplicity	default

Operation **IUMLState::SetNewStateInvariant**

parameter	name strKind return	direction in return	type string IUMLConstraint	type modifier	multiplicity	default

Operation **IUMLState::StateInvariant**

parameter	name return	direction return	type IUMLConstraint	type modifier	multiplicity	default

Operation **IUMLState::Submachine**

parameter	name	direction	type	type modifier	multiplicity	default

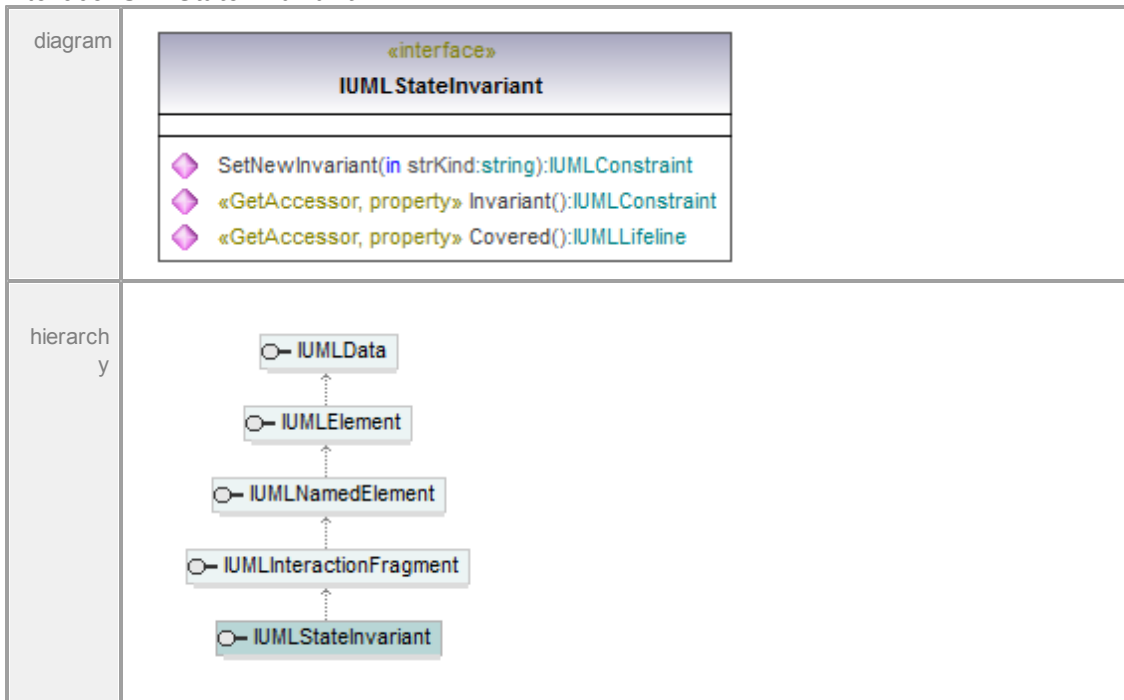
r	return	return	IUMLStateMach hine

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLStateInvariant

Interface IUMLStateInvariant



Operation IUMLStateInvariant::Covered

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLLifeline			

Operation IUMLStateInvariant::Invariant

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLConstraint			

Operation IUMLStateInvariant::SetNewInvariant

parameter	name	direction	type	type modifier	multiplicity	default
	strKind return	in return	string IUMLConstraint			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLStateMachine

Interface IUMLStateMachine

diagram	<pre> interface «interface» IUMLStateMachine InsertRegionAt(in nldx:int):IUMLRegion InsertConnectionPointAt(in nldx:int):IUMLPseudostate «GetAccessor, property» Regions():IUMLDataList «GetAccessor, property» SubmachineStates():IUMLDataList «GetAccessor, property» ConnectionPoints():IUMLDataList </pre>	
hierarchy	<pre> classDiagram class IUMLData class IUMLNamedElement class IUMLPackageableElement class IUMLType class IUMLNamespace class IUMLNamedElement class IUMLRedefinableElement class IUMLParameterableElement class IUMLTemplateableElement class IUMLClassifier class IUMLStructuredClassifier class IUMLEncapsulatedClassifier class IUMLClass class IUMLBehavior class IUMLStateMachine class IUMLProtocolStateMachine class IUMLClassifier class IUMLBehavedClassifier IUMLData < -- IUMLNamedElement IUMLNamedElement < -- IUMLPackageableElement IUMLPackageableElement < -- IUMLType IUMLPackageableElement < -- IUMLNamespace IUMLPackageableElement < -- IUMLNamedElement IUMLPackageableElement < -- IUMLRedefinableElement IUMLPackageableElement < -- IUMLParameterableElement IUMLPackageableElement < -- IUMLTemplateableElement IUMLNamedElement < -- IUMLClassifier IUMLClassifier < -- IUMLStructuredClassifier IUMLClassifier < -- IUMLEncapsulatedClassifier IUMLClassifier < -- IUMLClassifier IUMLClassifier < -- IUMLBehavedClassifier IUMLClassifier < -- IUMLClass IUMLClass < -- IUMLBehavior IUMLBehavior < -- IUMLStateMachine IUMLStateMachine < -- IUMLProtocolStateMachine </pre>	
typed Elements	Interface IDocument Interface IUMLDataAll Interface IUMLPseudostate Interface IUMLRegion Interface IUMLState	Operation GenerateStateMachineCode Operation StateMachine Submachine Operation StateMachine Operation StateMachine Operation Submachine

Operation IUMLStateMachine::ConnectionPoints

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLPseudostate .					

Operation **IUMLStateMachine::InsertConnectionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	return	return	<u>IUMLPseudostate</u>			

Operation **IUMLStateMachine::InsertRegionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	IUMLRegion			

Operation **IUMLStateMachine::Regions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLRegion .					

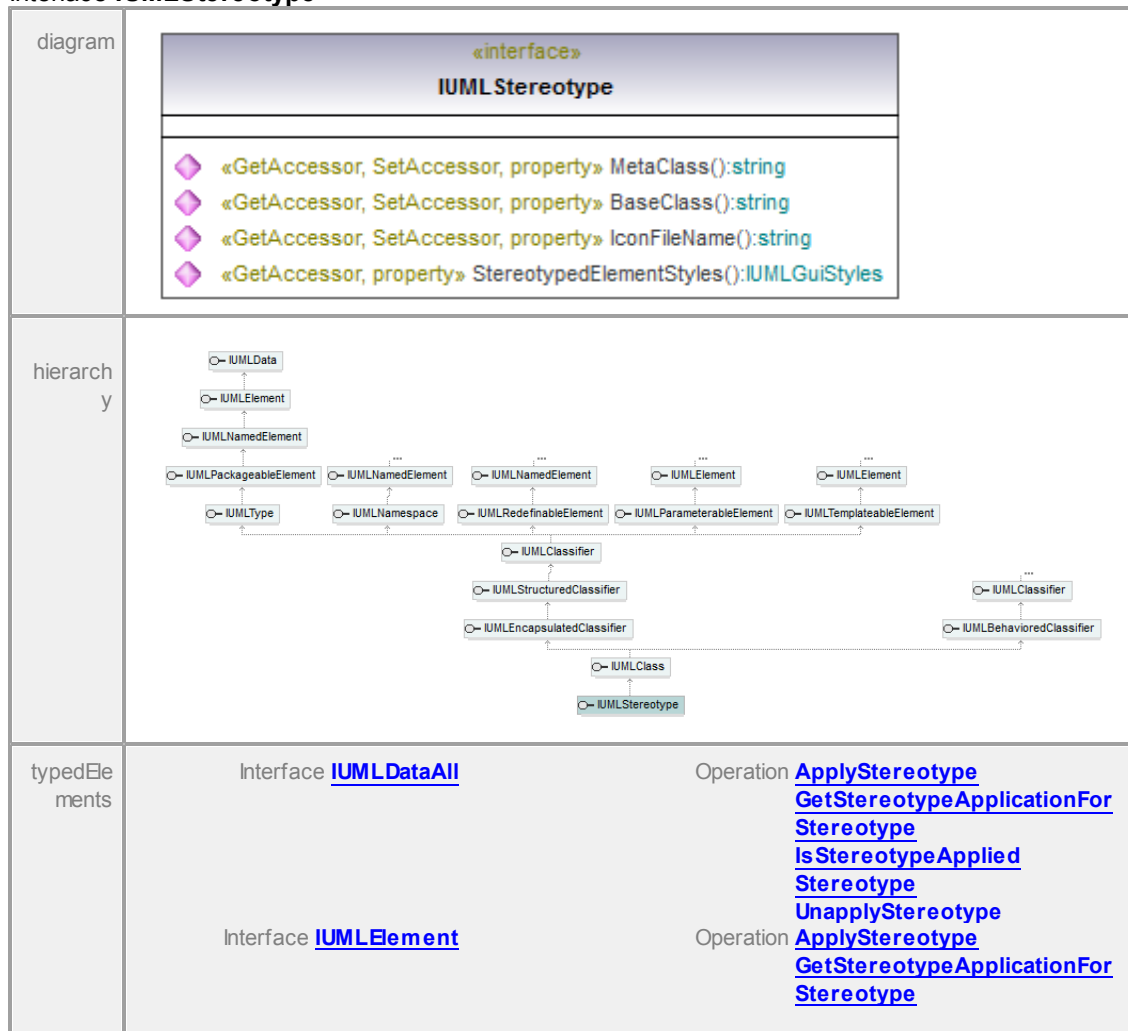
Operation **IUMLStateMachine::SubmachineStates**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLState .					

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLStereotype

Interface **IUMLStereotype**

	Interface	Operation
	IUMLStereotypeApplication	IsStereotypeApplied UnapplyStereotype Stereotype

Operation **IUMLStereotype::BaseClass**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLStereotype::IconFileName**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLStereotype::MetaClass**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

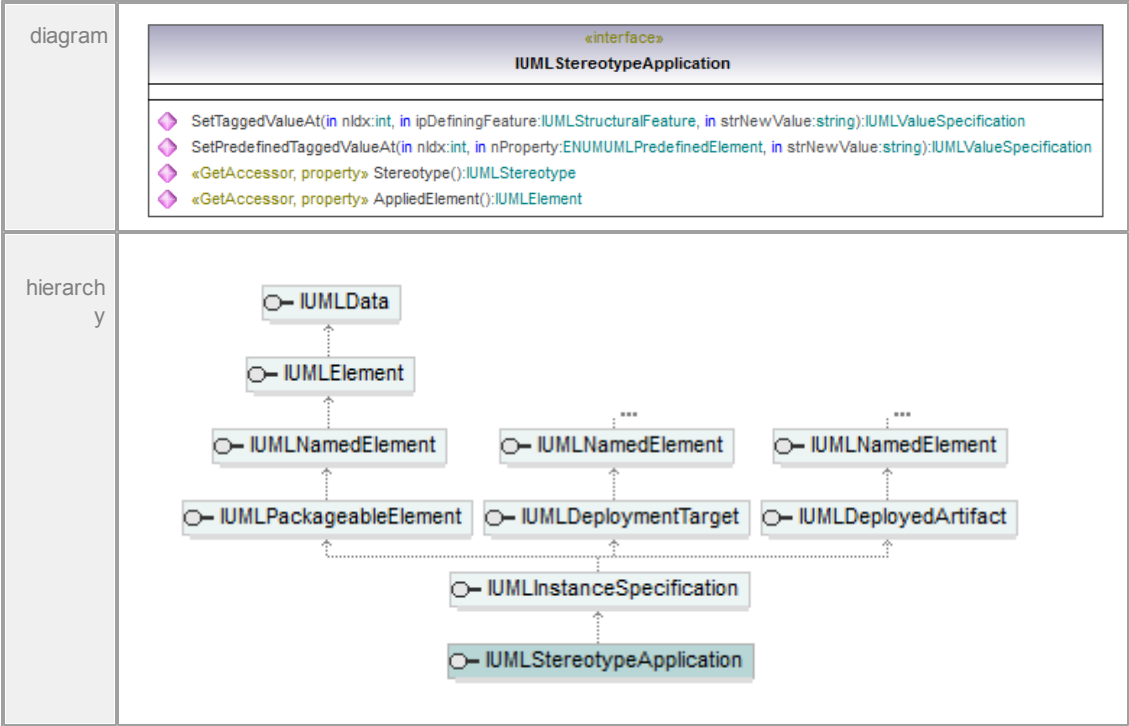
Operation **IUMLStereotype::StereotypedElementStyles**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiStyles			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLStereotypeApplication

Interface **IUMLStereotypeApplication**



typedElements	Interface IUMLDataAll	Operation ApplyPredefinedStereotype ApplyStereotype GetStereotypeApplicationForPredefinedStereotype GetStereotypeApplicationForStereotype
	Interface IUMLElement	Operation ApplyPredefinedStereotype ApplyStereotype GetStereotypeApplicationForPredefinedStereotype GetStereotypeApplicationForStereotype

Operation **IUMLStereotypeApplication::AppliedElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLElement			

Operation **IUMLStereotypeApplication::SetPredefinedTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	idx	in	int			
	propertyName	in	ENUMUMLPredefinedElement			
	strNewValue	in	string			
	return	return	IUMLValueSpecification			

Operation **IUMLStereotypeApplication::SetTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
	idx	in	int			
	ipDefiningFeature	in	IUMLStructuralFeature			
	strNewValue	in	string			
	return	return	IUMLValueSpecification			

Operation **IUMLStereotypeApplication::Stereotype**

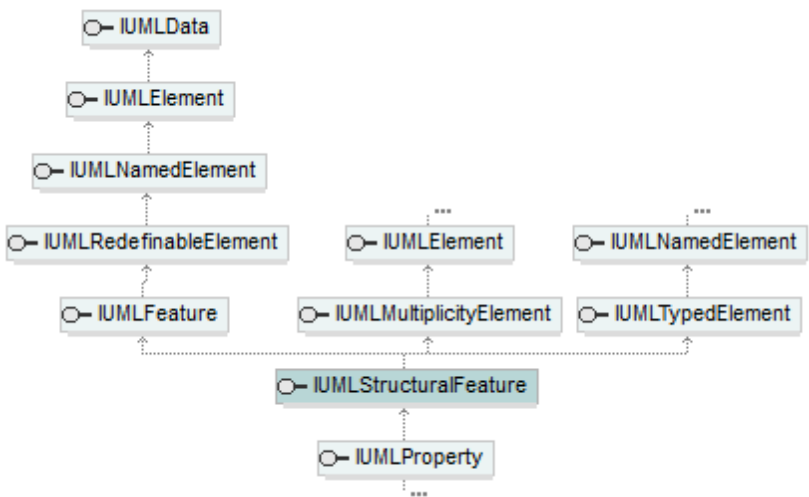
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLStereotype			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLStructuralFeature

Interface **IUMLStructuralFeature**

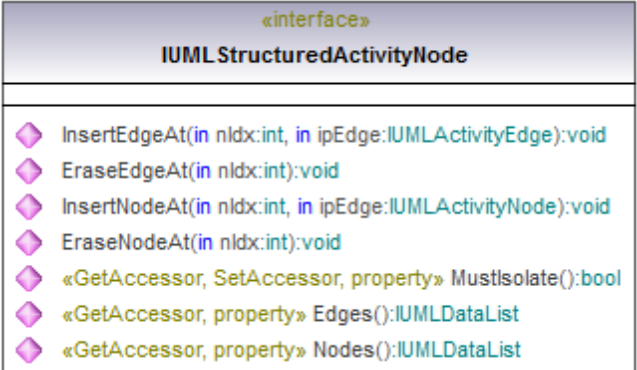
hierarchy	 <pre>graph BT IUMLData < -- IUMLElement IUMLElement < -- IUMLNamedElement IUMLNamedElement < -- IUMLRedefinableElement IUMLRedefinableElement < -- IUMLFeature IUMLFeature < -- IUMLStructuralFeature IUMLStructuralFeature < -- IUMLProperty IUMLStructuralFeature < -- IUMLMultiplicityElement IUMLStructuralFeature < -- IUMLTypedElement IUMLMultiplicityElement < -- IUMLElement IUMLTypedElement < -- IUMLNamedElement</pre>	
typedElements	Interface IUMLDataAll	Operation DefiningFeature InsertSlotAt SetSlotInstanceValueAt SetSlotValueAt SetTaggedValueAt
	Interface IUMLInstanceSpecification	Operation InsertSlotAt SetSlotInstanceValueAt SetSlotValueAt
	Interface IUMLSlot	Operation DefiningFeature
	Interface IUMLStereotypeApplication	Operation SetTaggedValueAt

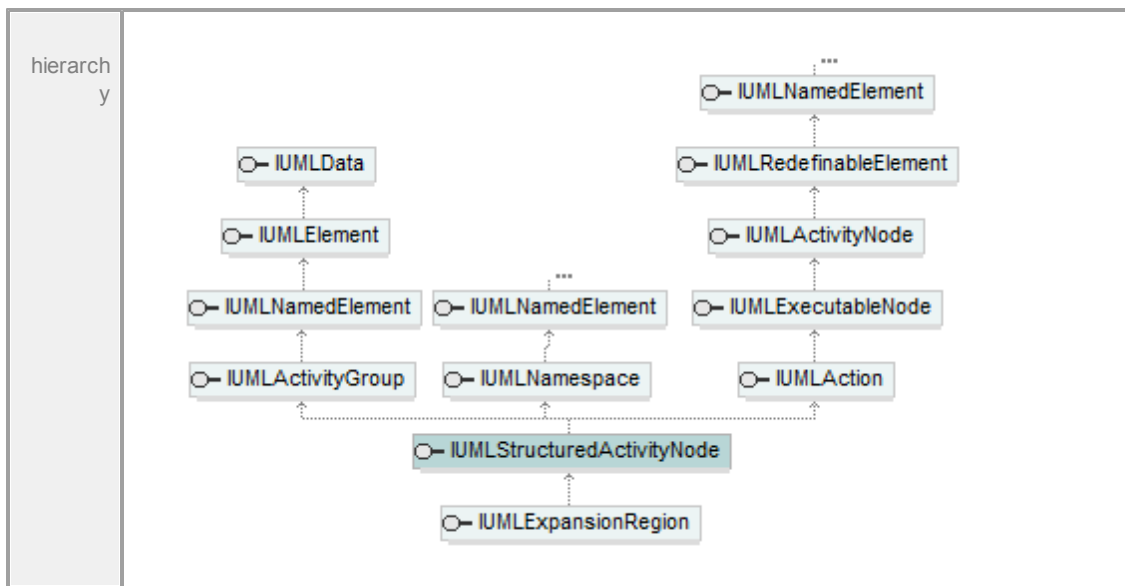
Operation **IUMLStructuralFeature::IsReadOnly**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

UModelAPI - IUMLStructuredActivityNode

Interface **IUMLStructuredActivityNode**

diagram	 <pre>classDiagram class IUMLStructuredActivityNode { <<interface>> InsertEdgeAt(in nIdx:int, in ipEdge:IUMLActivityEdge):void EraseEdgeAt(in nIdx:int):void InsertNodeAt(in nIdx:int, in ipEdge:IUMLActivityNode):void EraseNodeAt(in nIdx:int):void «GetAccessor, SetAccessor, property» MustIsolate():bool «GetAccessor, property» Edges():IUMLDataList «GetAccessor, property» Nodes():IUMLDataList }</pre>



Operation **IModelStructuredActivityNode::Edges**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IModelDataList			

Operation **IModelStructuredActivityNode::EraseEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	return	return	void			

Operation **IModelStructuredActivityNode::EraseNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	return	return	void			

Operation **IModelStructuredActivityNode::InsertEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	ipEdge	in	IModelActivityEdge			
	return	return	void			

Operation **IModelStructuredActivityNode::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nidx	in	int			
	ipEdge	in	IModelActivityNode			
	return	return	void			

Operation **IModelStructuredActivityNode::MustIsolate**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

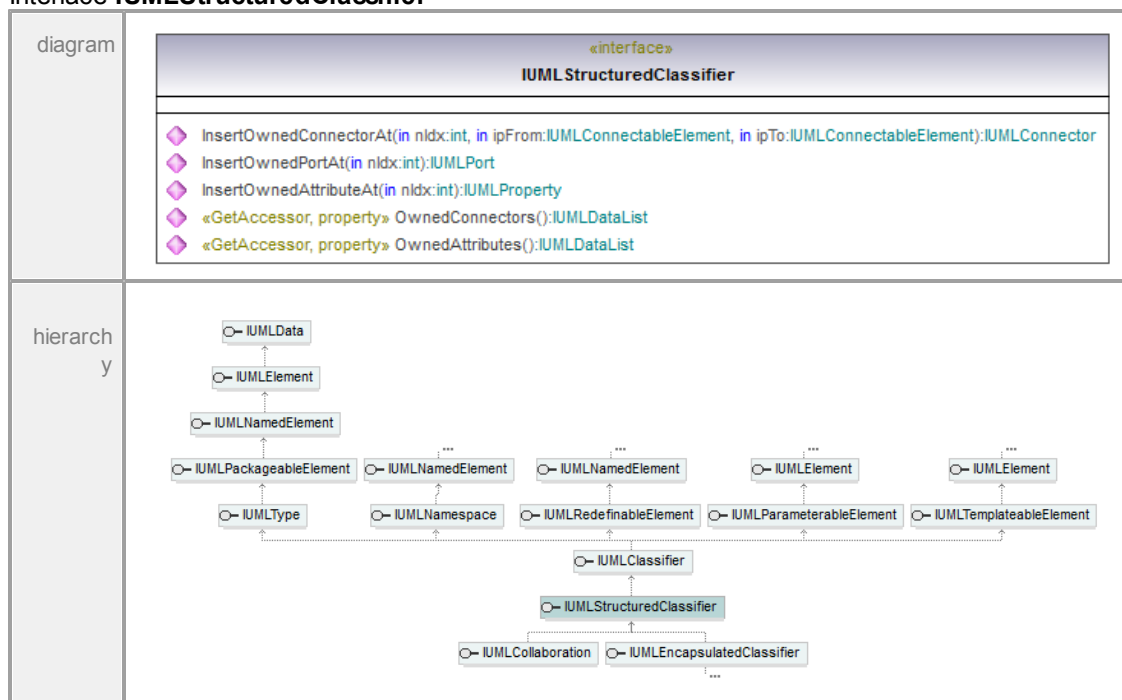
Operation **IUMLStructuredActivityNode::Nodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLStructuredClassifier

Interface **IUMLStructuredClassifier**Operation **IUMLStructuredClassifier::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	return	return	IUMLProperty			

Operation **IUMLStructuredClassifier::InsertOwnedConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			
	ipFrom	in	IUMLConnectableElement			
	ipTo	in	IUMLConnectableElement			
	return	return	IUMLConnector			

Operation **IUMLStructuredClassifier::InsertOwnedPortAt**

parameter	name	direction	type	type modifier	multiplicity	default

r	nldx return	in return	int IUMLPort

Operation **IUMLStructuredClassifier::OwnedAttributes**

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documentation	A list of elements of type IUMLProperty .					

Operation **IUMLStructuredClassifier::OwnedConnectors**

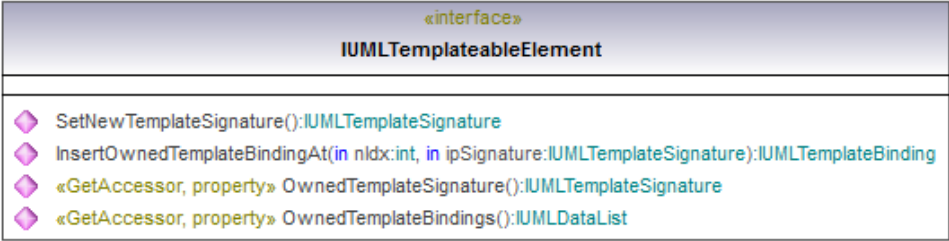
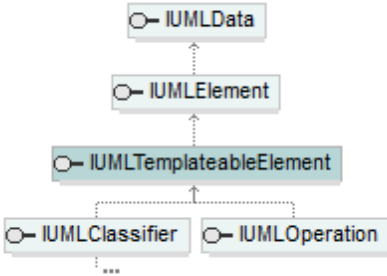
parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documentation	A list of elements of type IUMLConnector .					

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLTemplateableElement

Interface **IUMLTemplateableElement**

diagram		
hierarchy		
typedElements	Interface IUMLDataAll Interface IUMLTemplateBinding Interface IUMLTemplateSignature	Operation BoundElement Template Operation BoundElement Operation Template

Operation **IUMLTemplateableElement::InsertOwnedTemplateBindingAt**

parameter	name	direction	type	type modifier	multiplicity	default

r	nidx	in	int
	ipSignature	in	IUMLTemplateSignature
	return	return	IUMLTemplateBinding

Operation **IUMLTemplateableElement::OwnedTemplateBindings**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLTemplateBinding .					

Operation **IUMLTemplateableElement::OwnedTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTemplateSignature			

Operation **IUMLTemplateableElement::SetNewTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTemplateSignature			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLTemplateBinding

Interface **IUMLTemplateBinding**

diagram	«interface» IUMLTemplateBinding - InsertParameterSubstitutionAt(in nidx:int, in ipFormalParameter:IUMLTemplateParameter, in ipActualParameter:IUMLParameterableElement):IUMLTemplateParameterSubstitution «GetAccessor, property» Signature():IUMLTemplateSignature «GetAccessor, property» BoundElement():IUMLTemplateableElement «GetAccessor, property» ParameterSubstitutions():IUMLDataList	
hierarchy	<pre>graph BT IUMLTemplateBinding -- > IUMLDirectedRelationship IUMLDirectedRelationship -- > IUMLRelationship IUMLRelationship -- > IUMLElement IUMLElement -- > IUMLData</pre>	
typedElements	Interface IUMLDataAll Interface IUMLTemplateableElement	Operation InsertOwnedTemplateBindingAtTemplateBinding Operation InsertOwnedTemplateBindingAt

	Interface IUMLTemplateParameterSubstitution	Operation TemplateBinding

Operation **IUMLTemplateBinding::BoundElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLTemplateableElement			

Operation **IUMLTemplateBinding::InsertParameterSubstitutionAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipFormalParameter	in	IUMLTemplateParameterableElement			
	ipActualParameter	in	IUMLTemplateParameterSubstitution			
	return	return	IUMLTemplateParameterSubstitution			

Operation **IUMLTemplateBinding::ParameterSubstitutions**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLTemplateParameterSubstitution .					

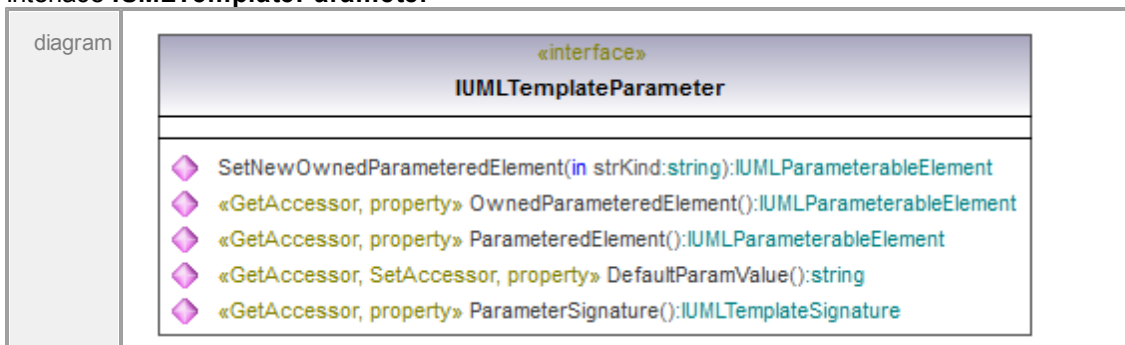
Operation **IUMLTemplateBinding::Signature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLTemplateSignature			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLTemplateParameter

Interface **IUMLTemplateParameter**

hierarchy	<pre>classDiagram class IUMLData class IUMLElement class IUMLTemplateParameter class IUMLClassifierTemplateParameter IUMLData < -- IUMLElement IUMLElement < -- IUMLTemplateParameter IUMLTemplateParameter < -- IUMLClassifierTemplateParameter</pre>		
typedElements	Interface IUMLDataAll Interface IUMLParameterableElement Interface IUMLTemplateBinding Interface IUMLTemplateParameterSubstitution	Operation FormalInsertParameterSubstitutionAtOwningTemplateParameterTemplateParameter Operation OwningTemplateParameterTemplateParameter Operation InsertParameterSubstitutionAt Operation Formal	

Operation **IUMLTemplateParameter::DefaultParamValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLTemplateParameter::OwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLParameterableElement			

Operation **IUMLTemplateParameter::ParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLParameterableElement			

Operation **IUMLTemplateParameter::ParameterSignature**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLTemplateSignature			

Operation **IUMLTemplateParameter::SetNewOwnedParameteredElement**

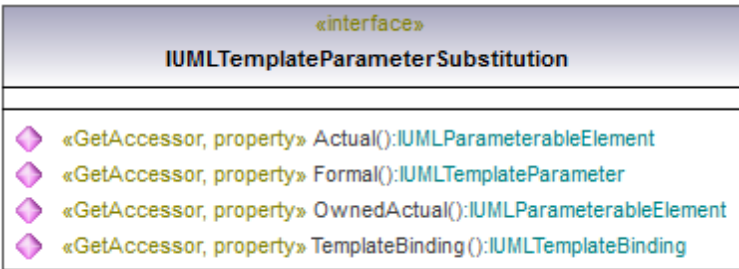
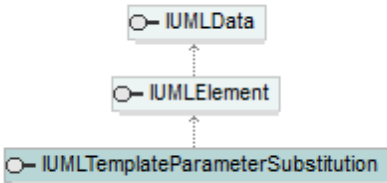
parameter	name	direction	type	type modifier	multiplicity	default
	strKind return	in return	string IUMLParameterableElement			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLTemplateParameterSubstitution

Interface **IUMLTemplateParameterSubstitution**

diagram		
hierarch y		
typedEle ments	Interface IUMLDataAll Interface IUMLTemplateBinding	Operation InsertParameterSubstitution At Operation InsertParameterSubstitution At

Operation **IUMLTemplateParameterSubstitution::Actual**

paramete r	name return	direction return	type IUMLParamete rableElement	type modifier	multiplicity	default

Operation **IUMLTemplateParameterSubstitution::Formal**

paramete r	name return	direction return	type IUMLTemplate Parameter	type modifier	multiplicity	default

Operation **IUMLTemplateParameterSubstitution::OwnedActual**

paramete r	name return	direction return	type IUMLParamete rableElement	type modifier	multiplicity	default

Operation **IUMLTemplateParameterSubstitution::TemplateBinding**

paramete r	name return	direction return	type IUMLTemplate Binding	type modifier	multiplicity	default

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLTemplateSignature

Interface **IUMLTemplateSignature**

diagram	«interface» IUMLTemplateSignature InsertOwnedTemplateParameterAt(in nldx:int):IUMLClassifierTemplateParameter «GetAccessor, property» OwnedTemplateParameters():IUMLDataList «GetAccessor, property» Template():IUMLTemplateableElement	
hierarchy	IUMLData IUMLElement IUMLTemplateSignature IUMLRedefinableTemplateSignature	
typedElements	Interface IUMLDataAll Interface IUMLTemplateableElement Interface IUMLTemplateBinding Interface IUMLTemplateParameter	Operation InsertOwnedTemplateBindingAt OwnedTemplateSignature ParameterSignature Operation SetNew TemplateSignature Signature Operation InsertOwnedTemplateBindingAt OwnedTemplateSignature SetNew TemplateSignature Signature Operation Signature Operation ParameterSignature

Operation **IUMLTemplateSignature::InsertOwnedTemplateParameterAt**

parameter	name nldx return	direction in return	type int IUMLClassifierTemplateParameter	type modifier	multiplicity	default

Operation **IUMLTemplateSignature::OwnedTemplateParameters**

parameter	name return	direction return	type IUMLDataList	type modifier	multiplicity	default
documentation	A list of elements of type IUMLTemplateParameter .					

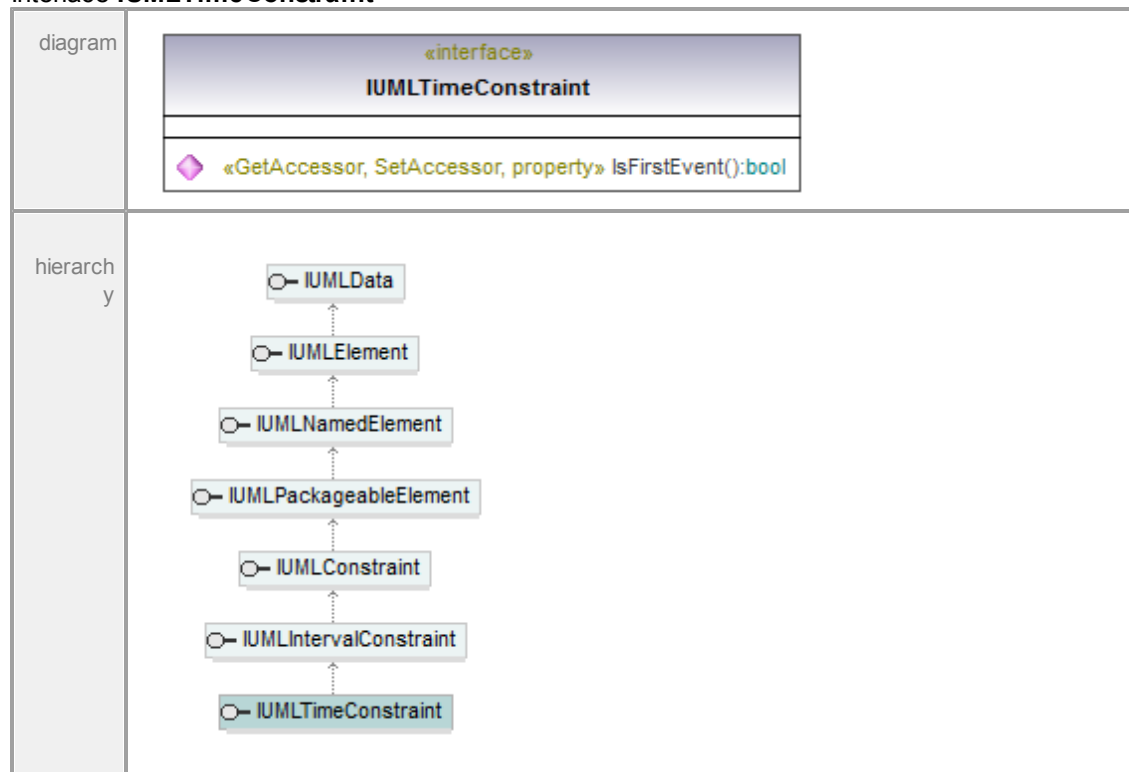
Operation **IUMLTemplateSignature::Template**

parameter	name return	direction return	type IUMLTemplateableElement	type modifier	multiplicity	default

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLTimeConstraint

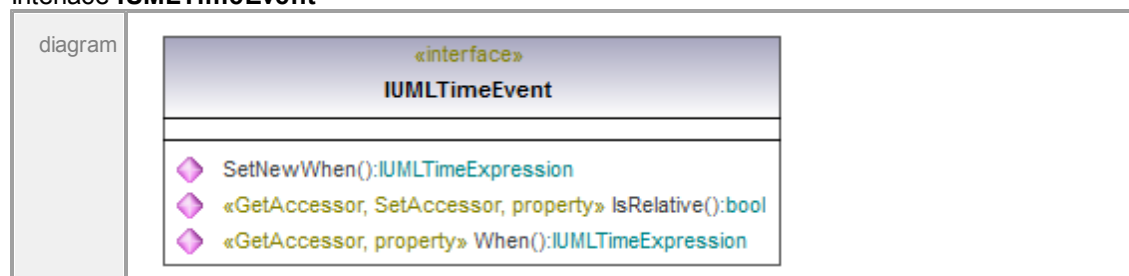
Interface **IUMLTimeConstraint**Operation **IUMLTimeConstraint::IsFirstEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLTimeEvent

Interface **IUMLTimeEvent**



Operation **IUMLTimeEvent::IsRelative**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

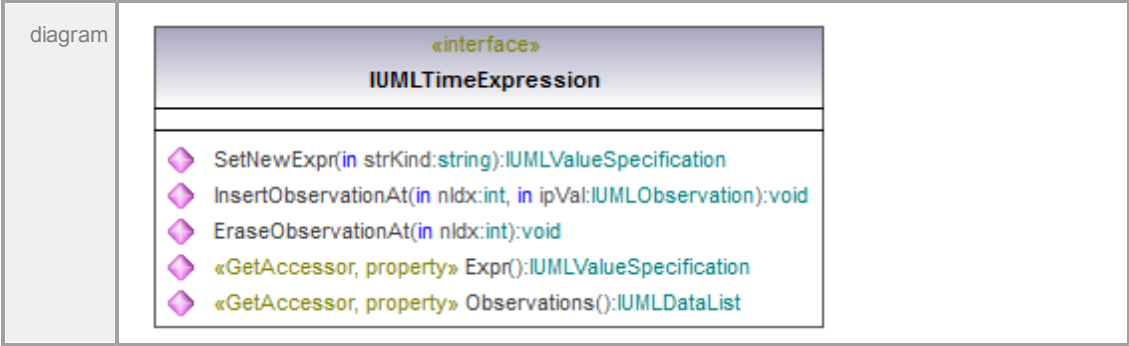
Operation **IUMLTimeEvent::SetNewWhen**

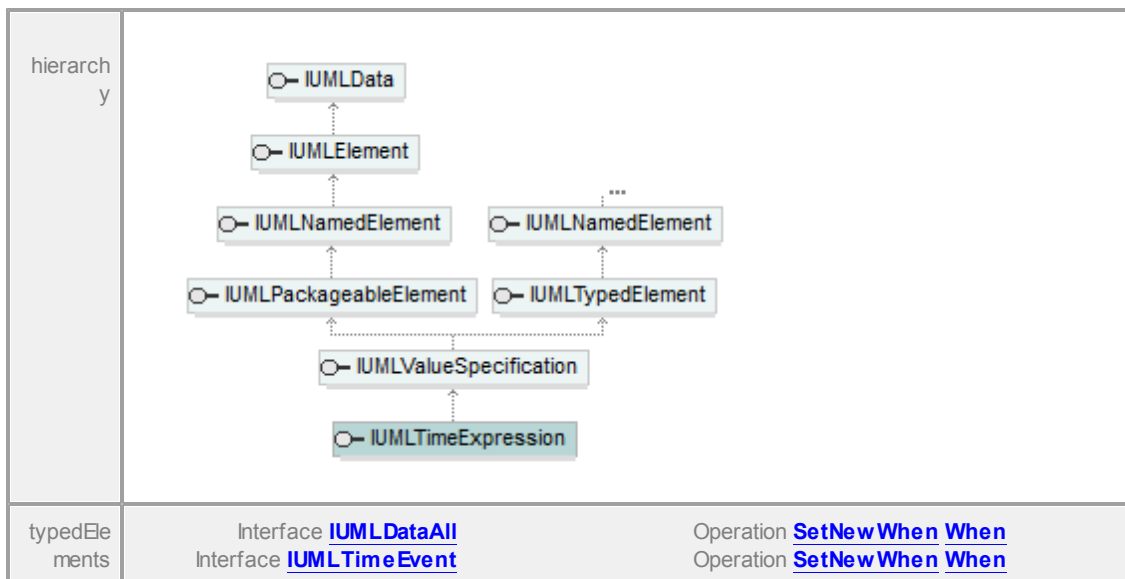
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	<u>IUMLTimeExpression</u>			

Operation **IUMLTimeEvent::When**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	<u>IUMLTimeExpression</u>			

Interface **IUMLTimeExpression**





Operation **IUMLTimeExpression::EraseObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	void			

Operation **IUMLTimeExpression::Expr**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLValueSpecification			

Operation **IUMLTimeExpression::InsertObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLObservation			
	return	return	void			

Operation **IUMLTimeExpression::Observations**

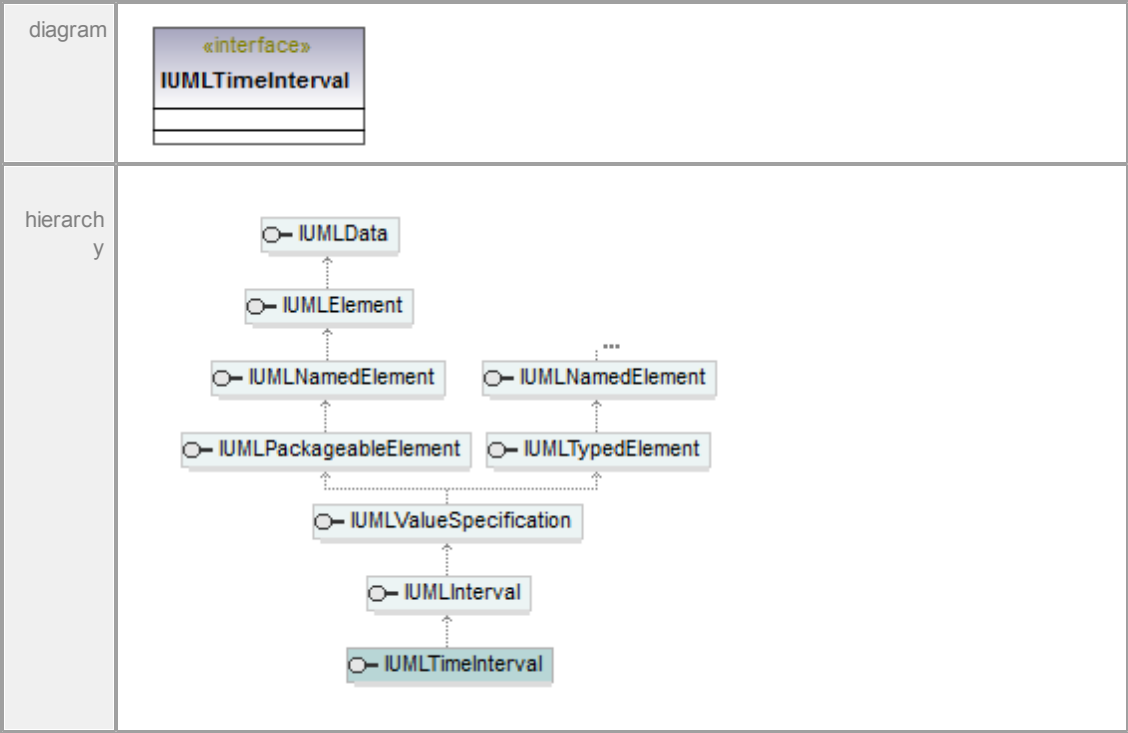
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLObservation .					

Operation **IUMLTimeExpression::SetNewExpr**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	in	string			
	return	return	IUMLValueSpecification			

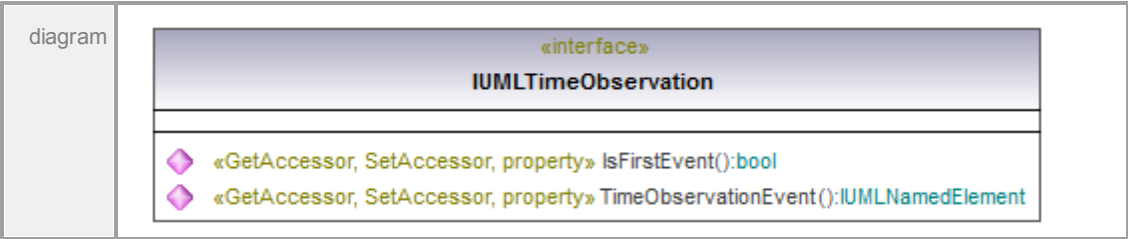
UModelAPI - IUMLTimeInterval

Interface **IUMLTimeInterval**



UModelAPI - IUMLTimeObservation

Interface **IUMLTimeObservation**





Operation **IUMLObservation::IsFirstEvent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLObservation::TimeObservationEvent**

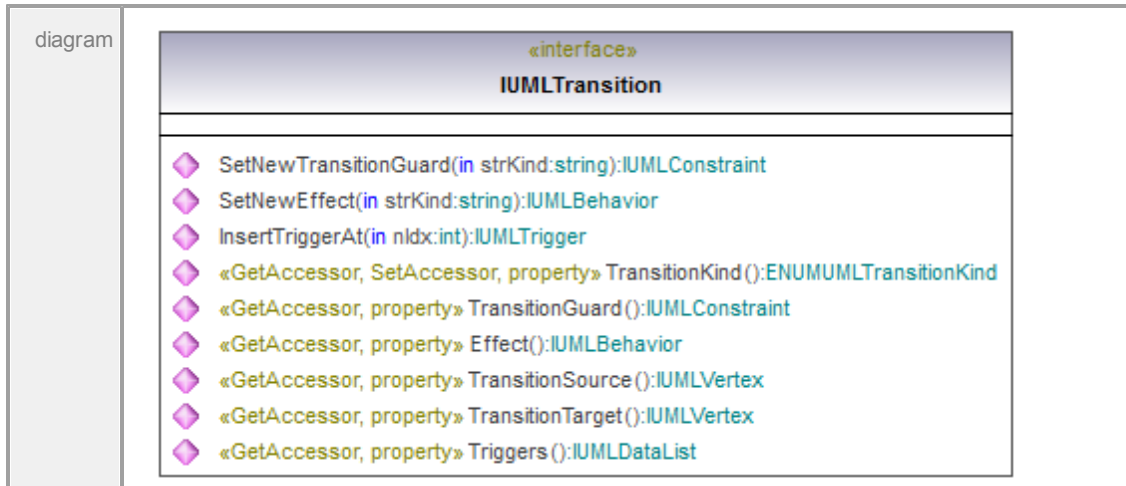
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLNamedElement			

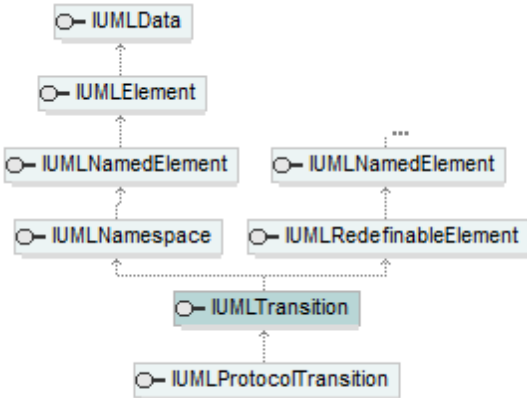
UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLTransition

Interface **IUMLTransition**



hierarchy		
typedElements	Interface IUMLDataAll Interface IUMLRegion	Operation InsertTransitionAt Operation InsertTransitionAt

Operation **IUMLTransition::Effect**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLBehavior			

Operation **IUMLTransition::InsertTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
	idx	in	int			
	return	return	IUMLTrigger			

Operation **IUMLTransition::SetNewEffect**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	in	string			
	return	return	IUMLBehavior			

Operation **IUMLTransition::SetNewTransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	in	string			
	return	return	IUMLConstraint			

Operation **IUMLTransition::TransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLConstraint			

Operation **IUMLTransition::TransitionKind**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ENUMUMLTransitionKind			

Operation **IUMLTransition::TransitionSource**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLVertex			

Operation **IUMLTransition::TransitionTarget**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLVertex			

Operation **IUMLTransition::Triggers**

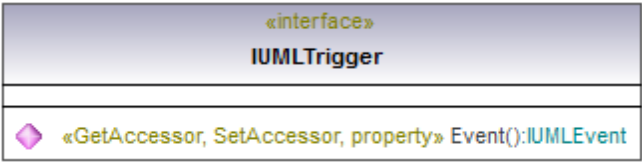
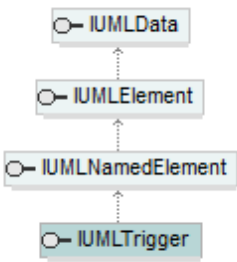
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLTrigger .					

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLTrigger

Interface **IUMLTrigger**

diagram		
hierarchy	 <pre> graph BT IUMLData -- > IUMLElement IUMLElement -- > IUMLNamedElement IUMLNamedElement -- > IUMLTrigger </pre>	
typedElements	Interface IUMLAcceptEventAction Interface IUMLDataAll Interface IUMLTransition	Operation InsertActionTriggerAt Operation InsertActionTriggerAt Operation InsertTriggerAt Operation InsertTriggerAt

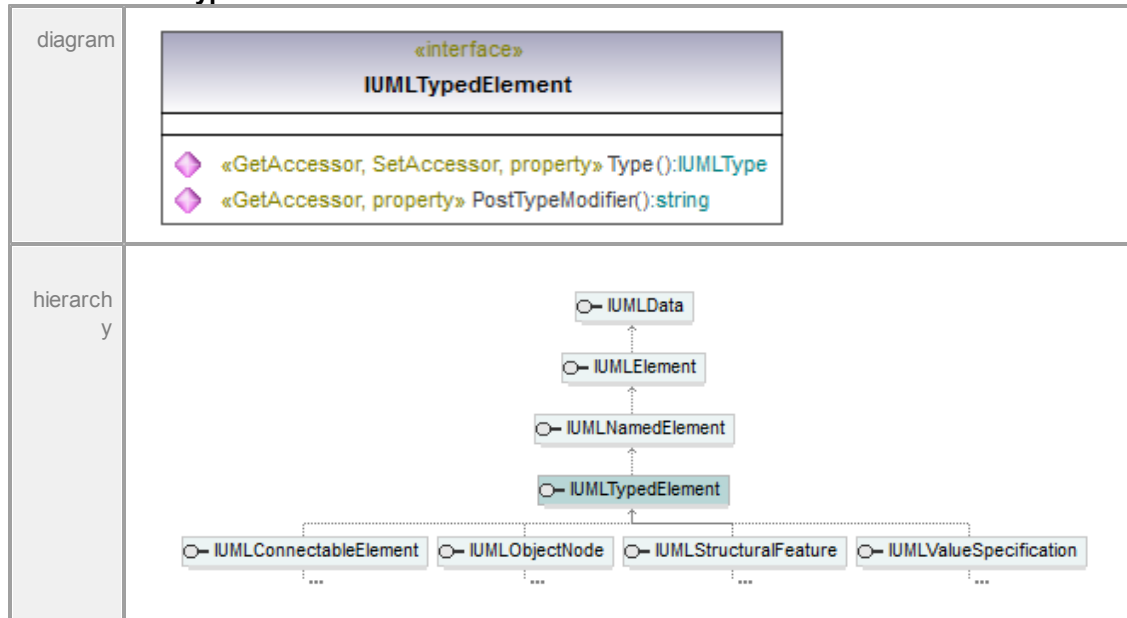
Operation **IUMLTrigger::Event**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEvent			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLTypedElement

Interface **IUMLTypedElement**Operation **IUMLTypedElement::PostTypeModifier**

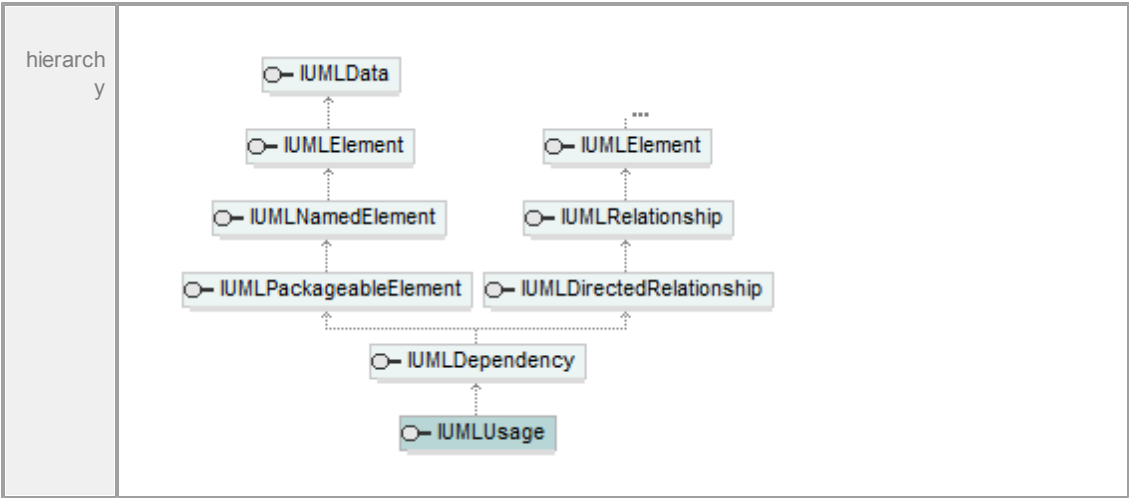
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLTypedElement::Type**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLType			

UModelAPI - IUMLUsage

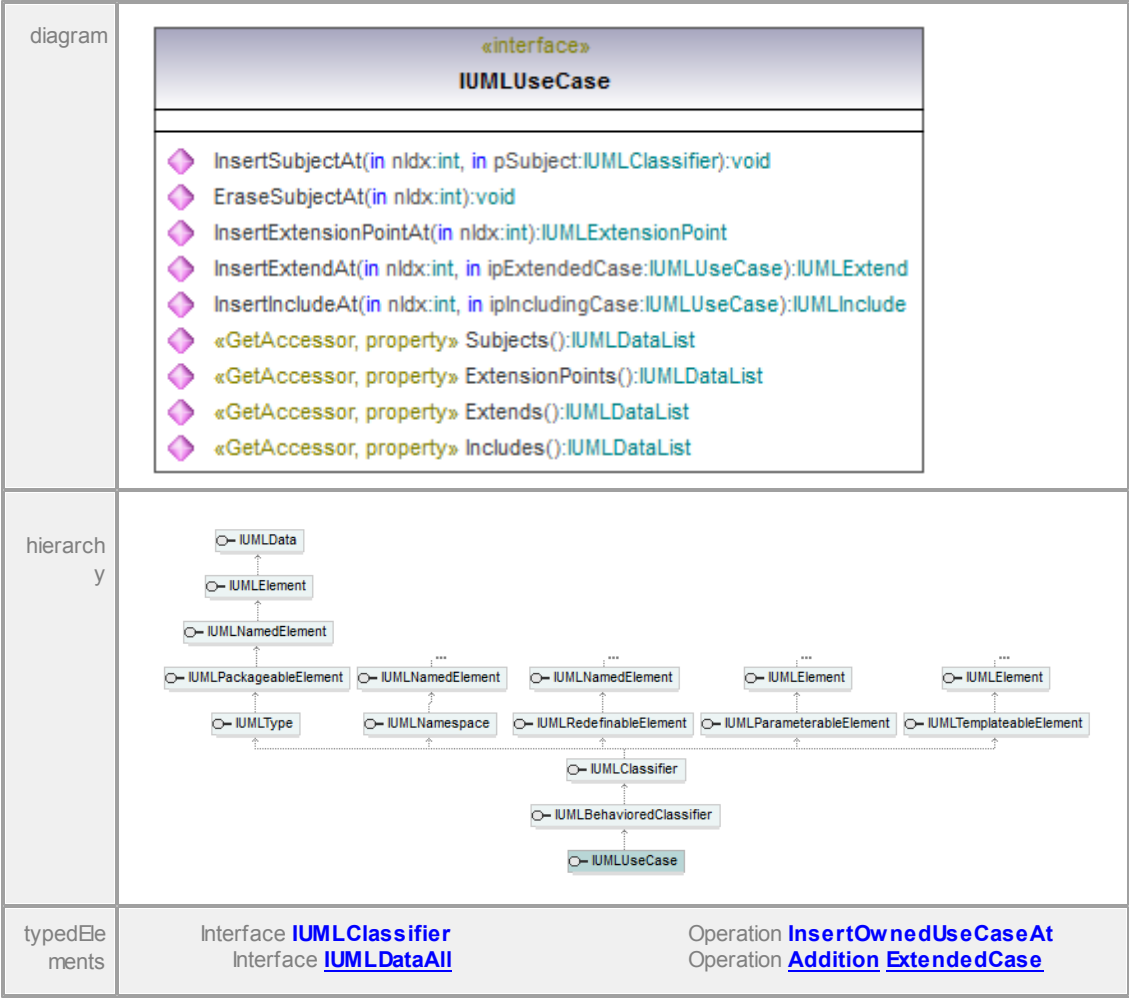
Interface **IUMLUsage**



UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLUseCase

Interface IUMLUseCase



	Interface IUMLExtend Interface IUMLExtensionPoint Interface IUMLInclude Interface IUMLUseCase	Operation Extension IncludingCase Operation InsertExtendAt Operation InsertIncludeAt Operation InsertOwnedUseCaseAt Operation Use Case Operation ExtendedCase Extension Operation Use Case Operation Addition IncludingCase Operation InsertExtendAt Operation InsertIncludeAt

Operation **IUMLUseCase::EraseSubjectAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	void			

Operation **IUMLUseCase::Extends**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLExtend .					

Operation **IUMLUseCase::ExtensionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLExtensionPoint .					

Operation **IUMLUseCase::Includes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLInclude .					

Operation **IUMLUseCase::InsertExtendAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipExtendedCase		IUMLUseCase			
	return	return	IUMLExtend			

Operation **IUMLUseCase::InsertExtensionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	IUMLExtensionPoint			

Operation **IUMLUseCase::InsertIncludeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipIncludingCase	in	IUMLCUseCase			
	return	return	IUMLCInclude			

Operation **IUMLCUseCase::InsertSubjectAt**

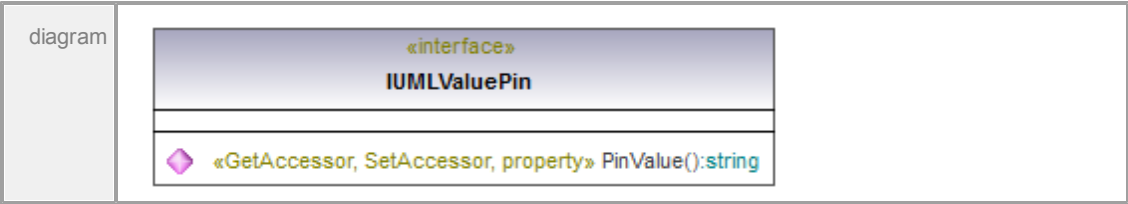
parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	pSubject	in	IUMLCClassifier			
	return	return	void			

Operation **IUMLCUseCase::Subjects**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLCDataList			
documentation	A list of elements of type IUMLCClassifier .					

UModelAPI - IUMLCValuePin

Interface **IUMLCValuePin**





parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

UModelAPI - IUMLValueSpecification

Interface **IUMLValueSpecification**

diagram		
hierarchy		
typedElements	Interface IUMLChangeEvent Interface IUMLConstraint Interface IUMLDataAll Interface IUMLDuration	Operation ChangeExpression Operation SetNew ChangeExpression Operation SetNew Specification Operation ActionValue Operation ChangeExpression Operation DefaultValue Expr Operation InsertOwnedArgumentAt Operation InsertValueAt Max MaxInt Min Operation MinInt Selector Operation SetNew ActionValue Operation SetNew ChangeExpression Operation SetNew DefaultValue Operation SetNew Expr SetNew Max Operation SetNew MaxInt SetNew Min Operation SetNew MinInt Operation SetNew Selector Operation SetNew Specification Operation SetPredefinedTaggedValueAt Operation SetSlotValueAt Operation SetTaggedValueAt Operation Specification Operation Expr SetNew Expr

	Interface IUMLInstanceSpecification	Operation SetNewSpecification
	Interface IUMLInteractionConstraint	Operation SetSlotValueAtSpecification
	Interface IUMLInterval	Operation MaxInt MinInt SetNewMaxInt
		Operation SetNewMinInt
	Interface IUMLLifeline	Operation Max Min SetNewMax
	Interface IUMLMessage	Operation SetNewMin
	Interface IUMLParameter	Operation Selector SetNewSelector
		Operation InsertOwnedArgumentAt
	Interface IUMLProperty	Operation DefaultValue
		Operation SetNewDefaultValue
	Interface IUMLSlot	Operation DefaultValue
	Interface IUMLStereotypeApplication	Operation SetNewDefaultValue
	Interface IUMLTimeExpression	Operation InsertValueAt
		Operation SetPredefinedTaggedValueAt
		Operation SetTaggedValueAt
	Interface IUMLValueSpecificationAction	Operation Expr SetNewExpr
		Operation ActionValue
		Operation SetNewActionValue

Operation **IUMLValueSpecification::BooleanValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLValueSpecification::Expression**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLExpression			

Operation **IUMLValueSpecification::IntegerValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLValueSpecification::IsComputable**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLValueSpecification::IsNull**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLValueSpecification::OwningConstraint**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLConstraint			

Operation **IUMLValueSpecification::OwningInstanceSpec**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLInstanceSpecification			

Operation **IUMLValueSpecification::OwningLower**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLMultiplicityElement			

Operation **IUMLValueSpecification::OwningParameter**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLParameter			

Operation **IUMLValueSpecification::OwningProperty**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLProperty			

Operation **IUMLValueSpecification::OwningSlot**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLSlot			

Operation **IUMLValueSpecification::OwningUpper**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLMultiplicityElement			

Operation **IUMLValueSpecification::StringValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLValueSpecification::UnlimitedValue**

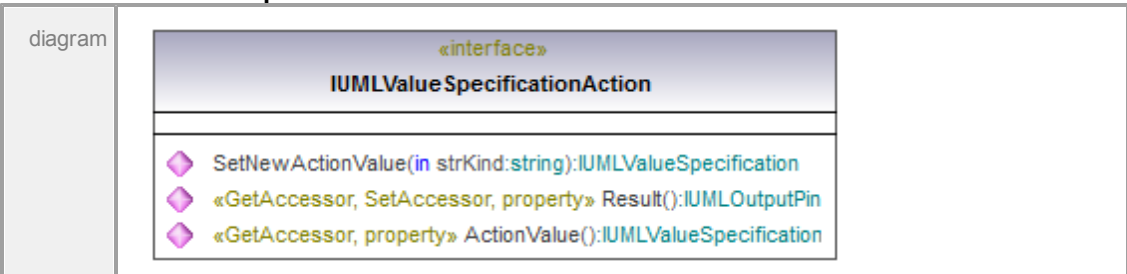
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

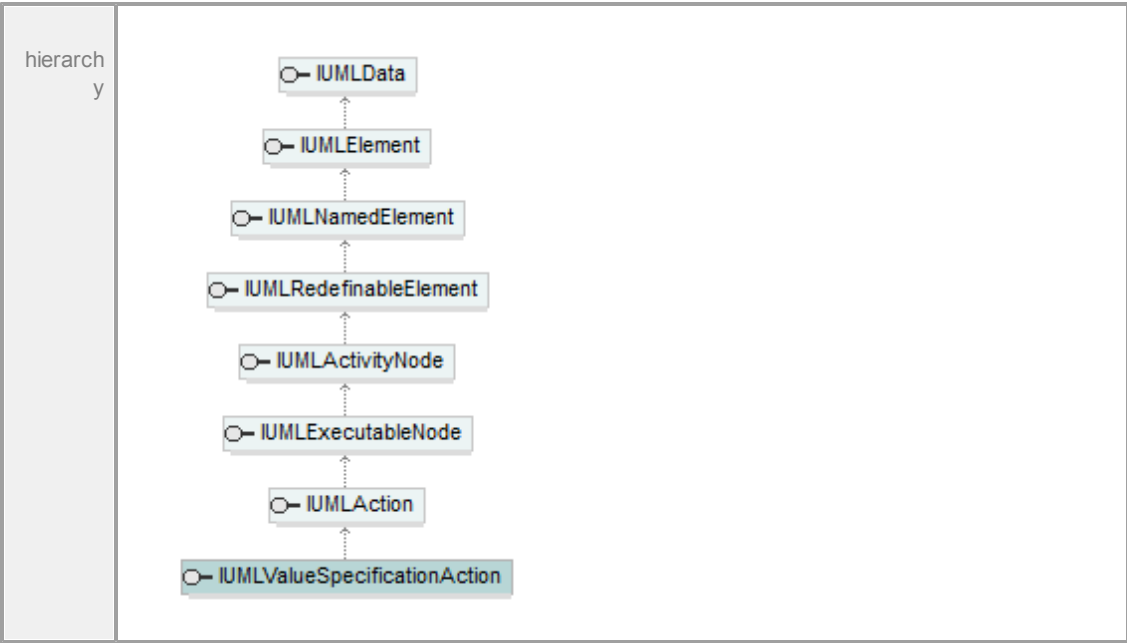
UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLValueSpecificationAction

Interface **IUMLValueSpecificationAction**





Operation **IUMLValueSpecificationAction::ActionValue**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLValueSpecification			

Operation **IUMLValueSpecificationAction::Result**

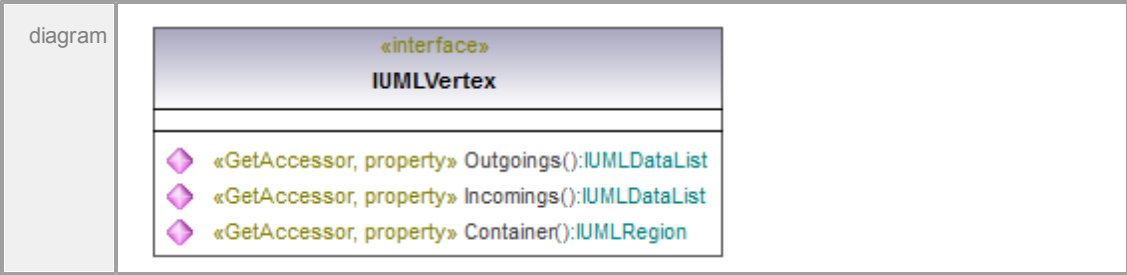
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLOutputPin			

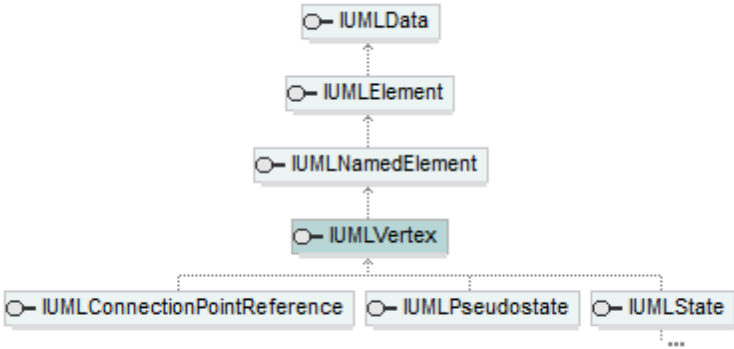
Operation **IUMLValueSpecificationAction::SetNewActionValue**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind return	in return	string IUMLValueSpecification			

UModelAPI - IUMLVertex

Interface **IUMLVertex**



hierarchy		
typedElements	Interface IUMLDataAll	Operation InsertSubVertexAt InsertTransitionAt TransitionSource TransitionTarget
	Interface IUMLRegion	Operation InsertSubVertexAt InsertTransitionAt
	Interface IUMLTransition	Operation TransitionSource TransitionTarget

Operation **IUMLVertex::Container**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLRegion			

Operation **IUMLVertex::Incomings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLTransition .					

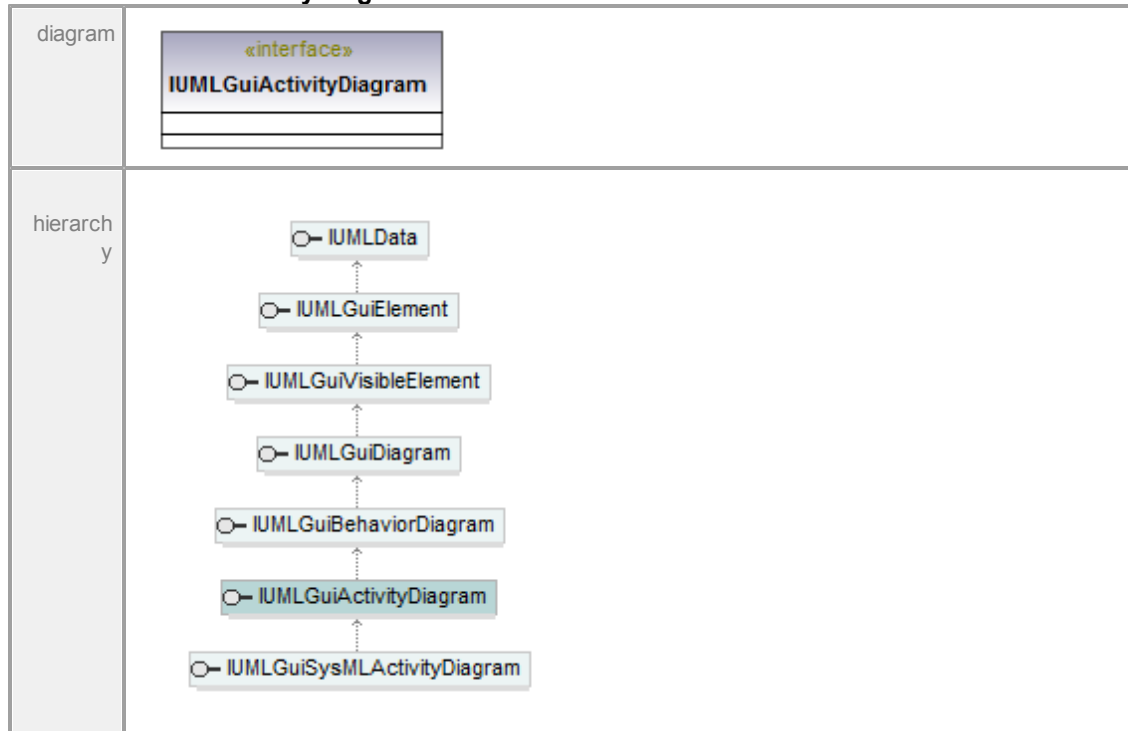
Operation **IUMLVertex::Outgoings**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	A list of elements of type IUMLTransition .					

IUMLGuiElement

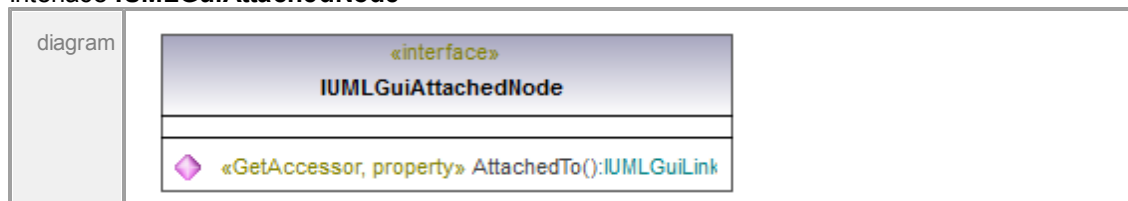
This is a list of elements from Altova for diagrams and to show [IUMLElements](#) on diagrams.

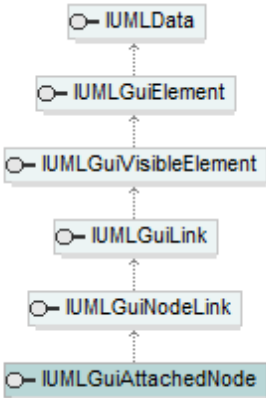
UModelAPI - IUMLGuiActivityDiagram

Interface **IUMLGuiActivityDiagram**UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiAttachedNode

Interface **IUMLGuiAttachedNode**

hierarchy	
documentation	This GUI element is a node (possibly without a linked IUMLElement) which is directly attached to a IUMLGuiNodeLink. It disappears and pops up based on data set in the element of the IUMLGuiNodeLink it is attached to. The user usually only has control of this element via styles or the node it is attached to. This node is used as graphical object on diagrams to represent Tagged Values for example.

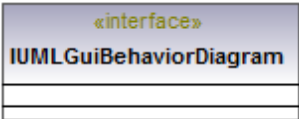
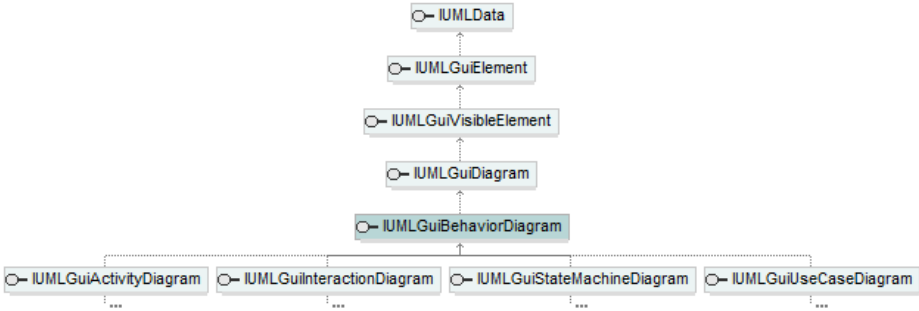
Operation **IUMLGuiAttachedNode::AttachedTo**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLGuiLink			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

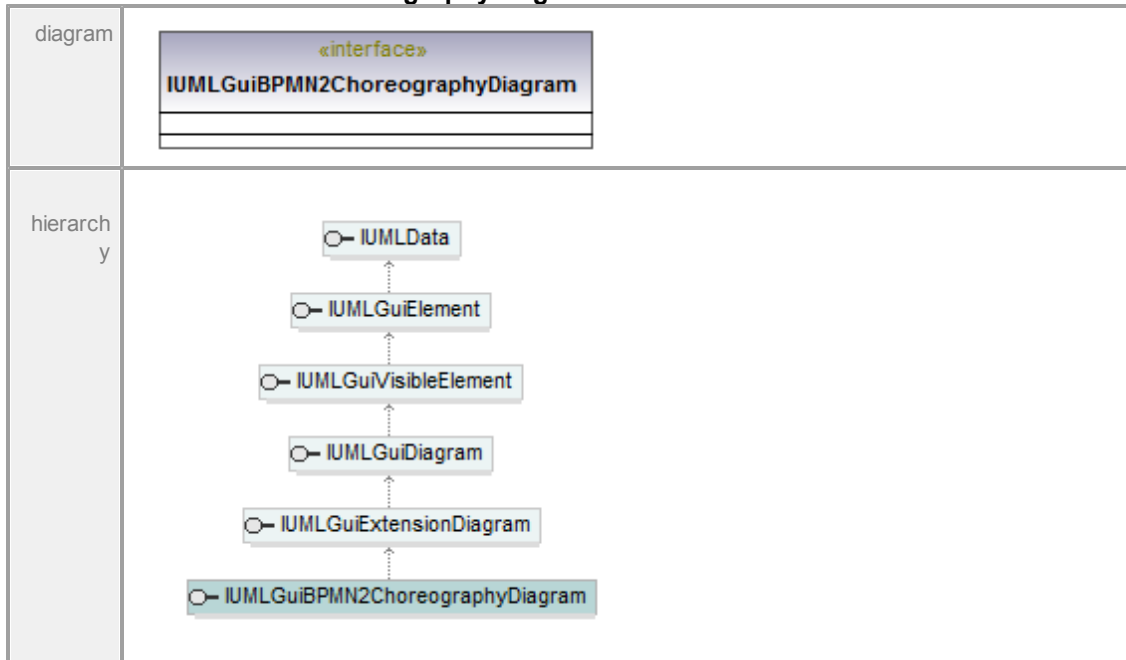
UModelAPI - IUMLGuiBehaviorDiagram

Interface **IUMLGuiBehaviorDiagram**

diagram	
hierarchy	

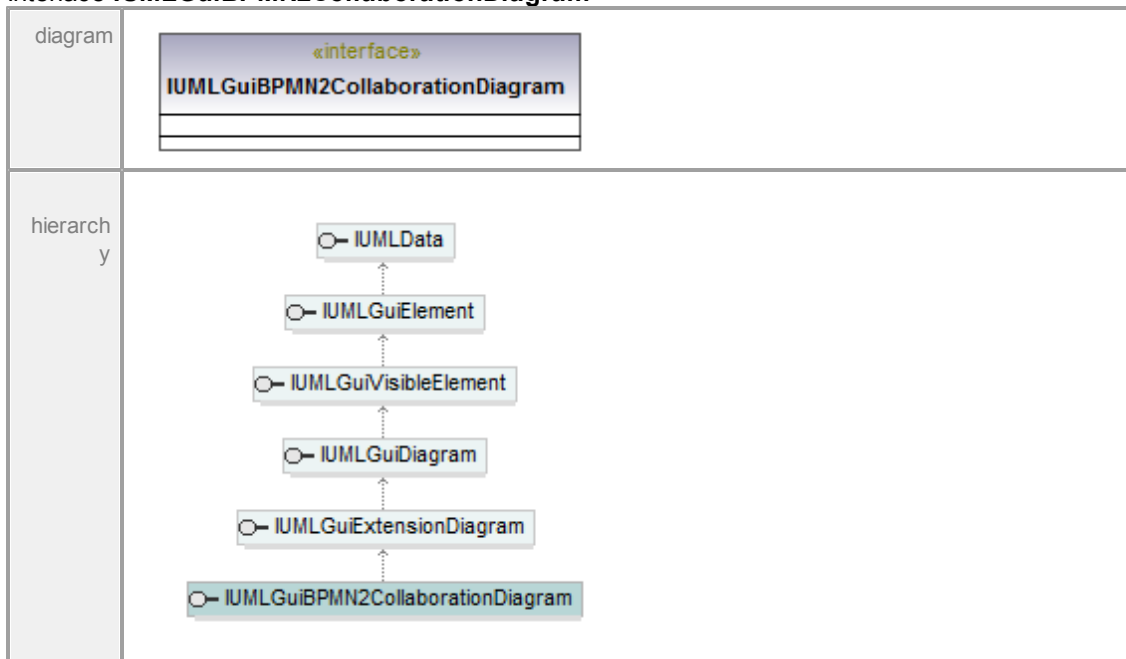
UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLGuiBPMN2ChoreographyDiagram

Interface **IUMLGuiBPMN2ChoreographyDiagram**UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

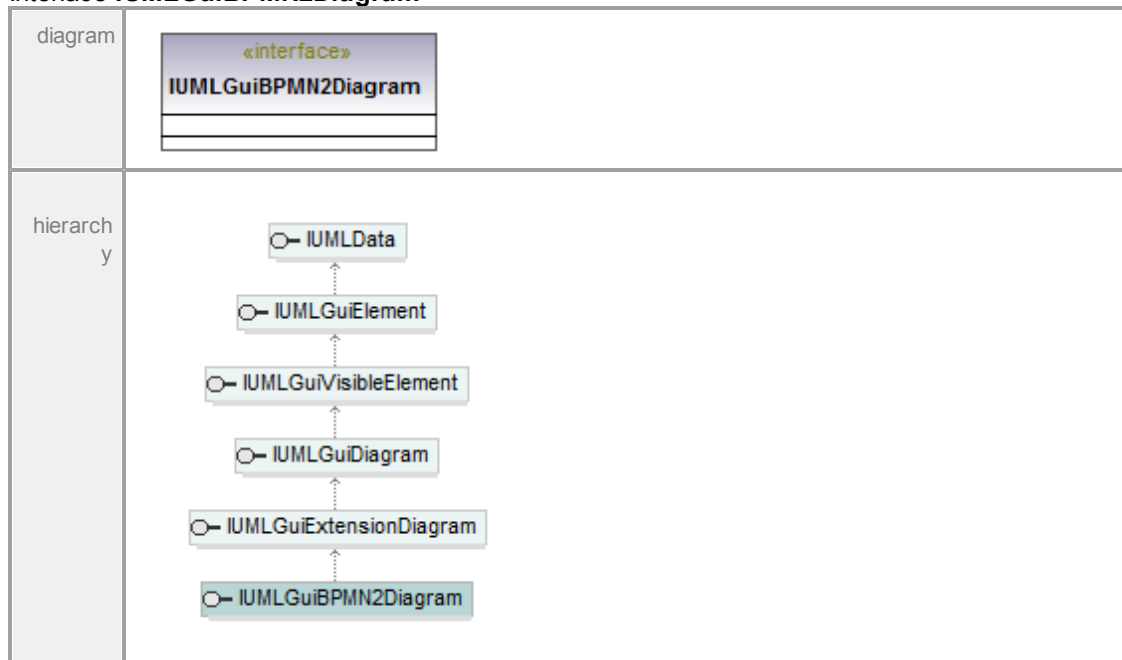
UModelAPI - IUMLGuiBPMN2CollaborationDiagram

Interface **IUMLGuiBPMN2CollaborationDiagram**

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

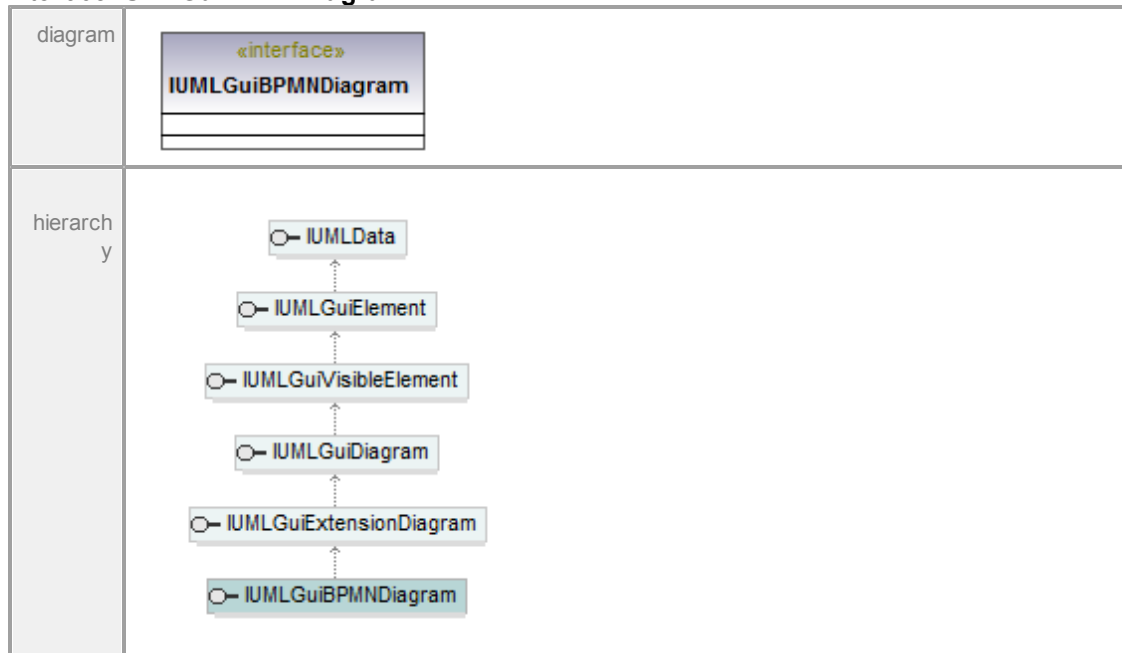
08/03/15 13:42:06

UModelAPI - IUMLGuiBPMN2Diagram

Interface **IUMLGuiBPMN2Diagram**UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

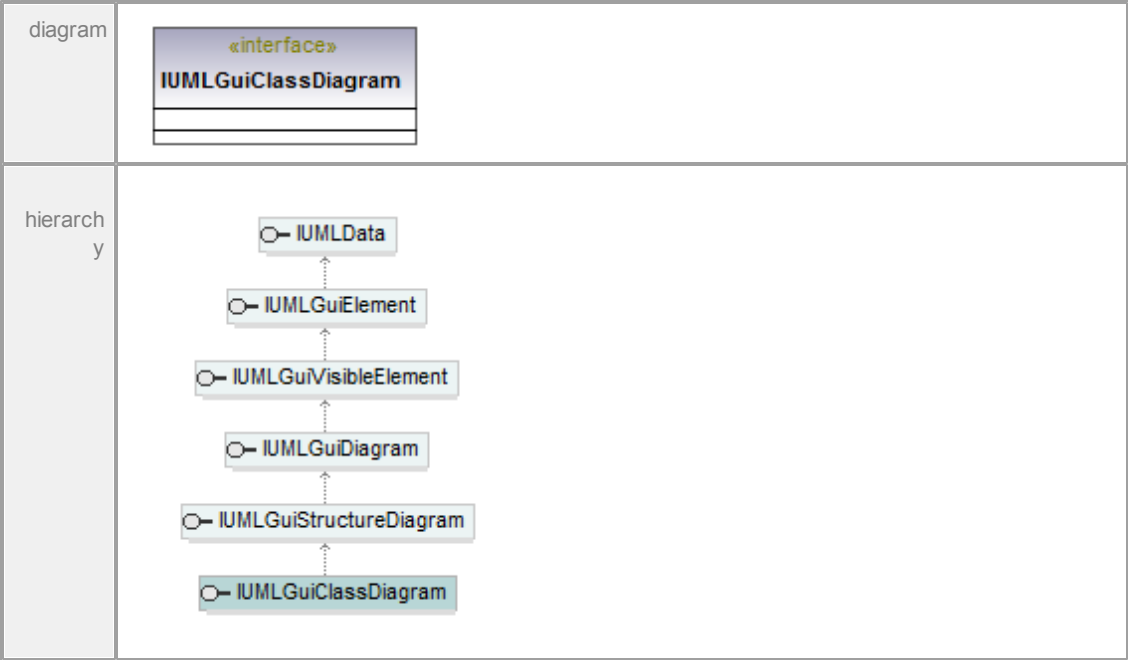
08/03/15 13:42:06

UModelAPI - IUMLGuiBPMNDiagram

Interface **IUMLGuiBPMNDiagram**

UModelAPI - IUMLGuiClassDiagram

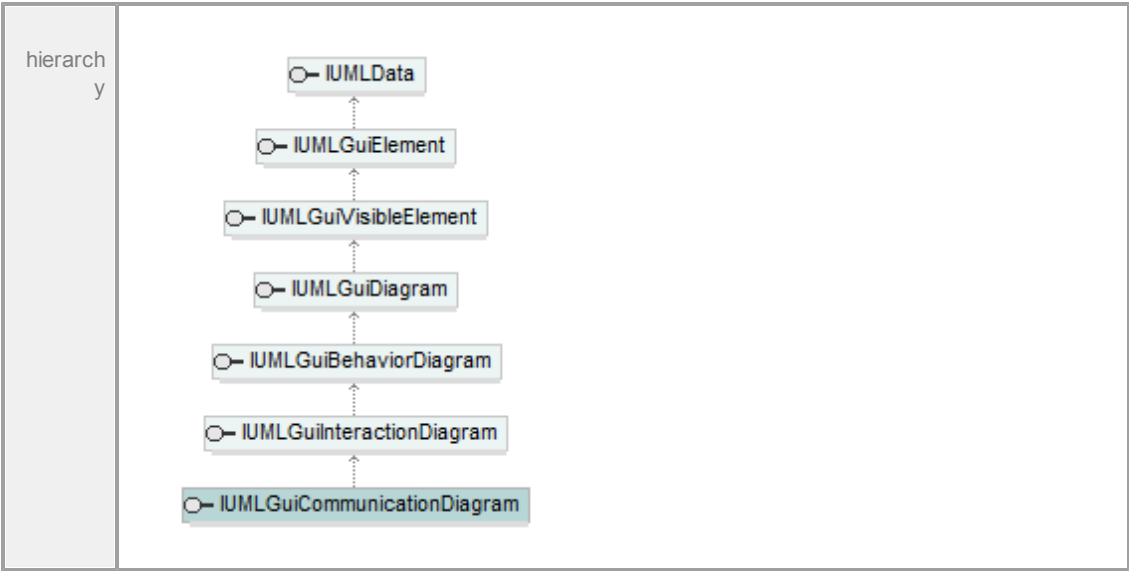
Interface **IUMLGuiClassDiagram**



UModelAPI - IUMLGuiCommunicationDiagram

Interface **IUMLGuiCommunicationDiagram**



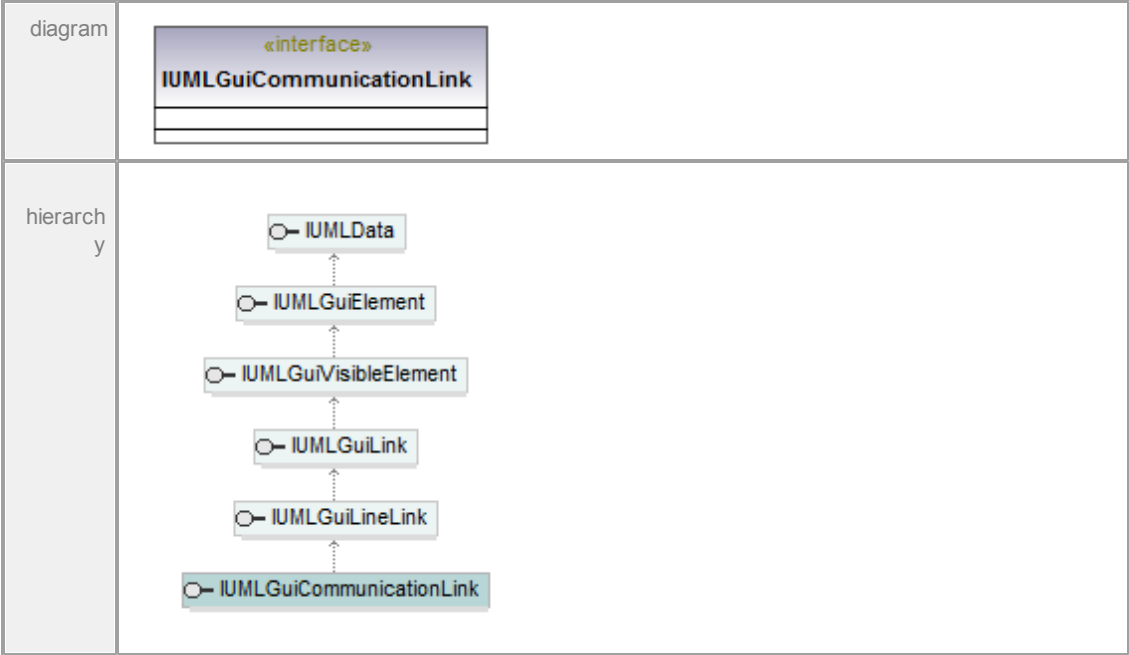


Operation **IUMLGuiCommunicationDiagram::AddUMLGuiCommunicationLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromLink	in	IUMLGuiLink			
	ipToLink	in	IUMLGuiLink			
	return	return	IUMLGuiCommunicationLink			

UModelAPI - IUMLGuiCommunicationLink

Interface **IUMLGuiCommunicationLink**

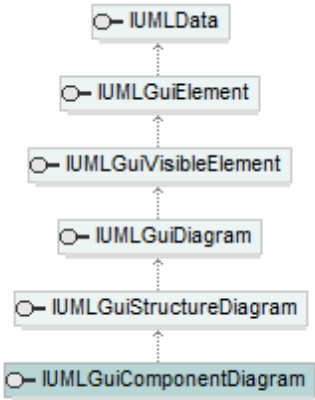


typedElements	Interface IUMLGuiCommunicationDiagram	Operation AddUMLGuiCommunicationLink
documentation	This line link is used on communication diagrams to provide a connection link for messages between lifelines.	

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLGuiComponentDiagram

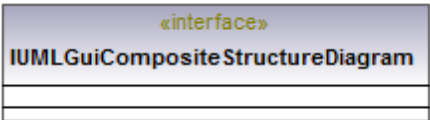
Interface IUMLGuiComponentDiagram

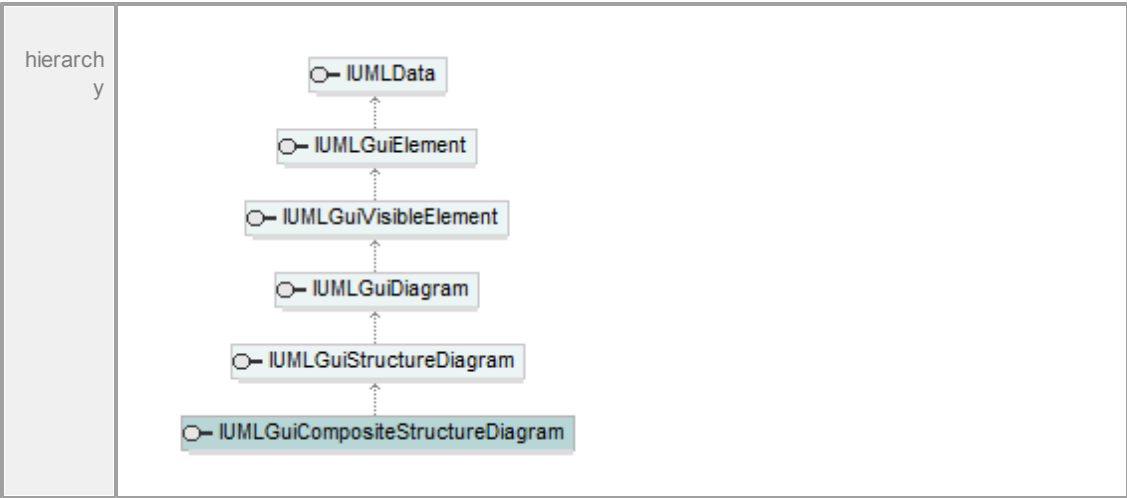
diagram	
hierarchy	

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLGuiCompositeStructureDiagram

Interface IUMLGuiCompositeStructureDiagram

diagram	



UML documentation generated by [UModel UML Editor](http://www.altova.com/umodel) <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiConstrainedNode

Interface IUMLGuiConstrainedNode

diagram	<pre>classDiagram class IUMLGuiConstrainedNode { <<interface>> +ConstrainingPointX():int +ConstrainingPointY():int +ConstrainingAreaIndex():int }</pre>
hierarchy	<pre>graph TD; IUMLData < > IUMLGuiElement; IUMLGuiElement < > IUMLGuiVisibleElement; IUMLGuiVisibleElement < > IUMLGuiLink; IUMLGuiLink < > IUMLGuiNodeLink; IUMLGuiNodeLink < > IUMLGuiConstrainedNode;</pre>
documentat	This node link is used to represent objects on diagrams w hich can be directly attached to a parent node link and placed relatively to and in special constraining areas of the parent. Used for example for Pins on Activiy Diagrams.

Operation IUMLGuiConstrainedNode::ConstrainingPointX

paramete	name	direction	type	type modifier	multiplicity	default

r	return	return	int
documentation	X coordinate relative to the upper left position of the containing area.		

Operation **IUMLGuiConstrainedNode::ConstrainingPointY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			
documentation	Y coordinate relative to the upper left position of the containing area.					

Operation **IUMLGuiConstrainedNode::ContainingAreaIndex**

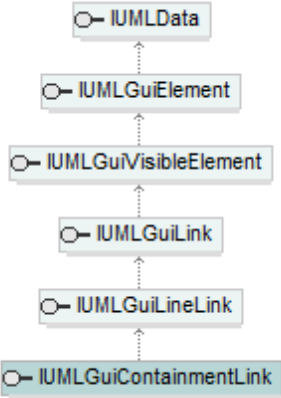
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			
documentation	Defines the index of the area where this node is currently in and to which its relative position has its origin.					

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiContainmentLink

Interface **IUMLGuiContainmentLink**

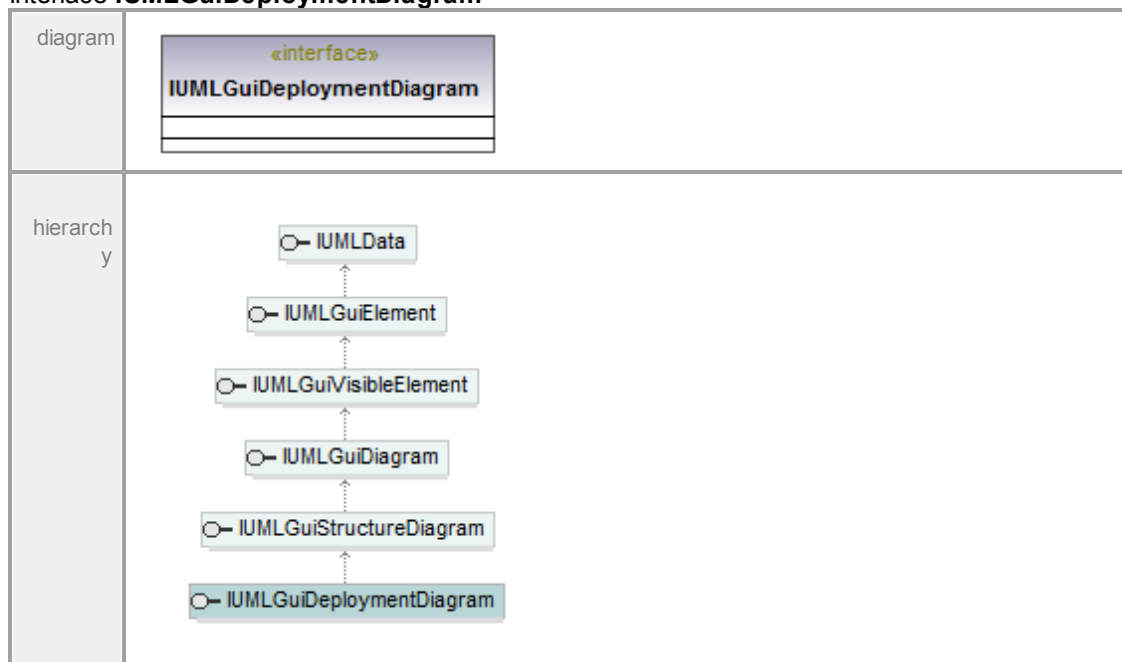
diagram		
hierarchy		
typedElements	Interface IUMLDataAll Interface IUMLGuiDiagram	Operation AddUMLGuiContainmentLink Operation AddUMLGuiContainmentLink

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiDeploymentDiagram

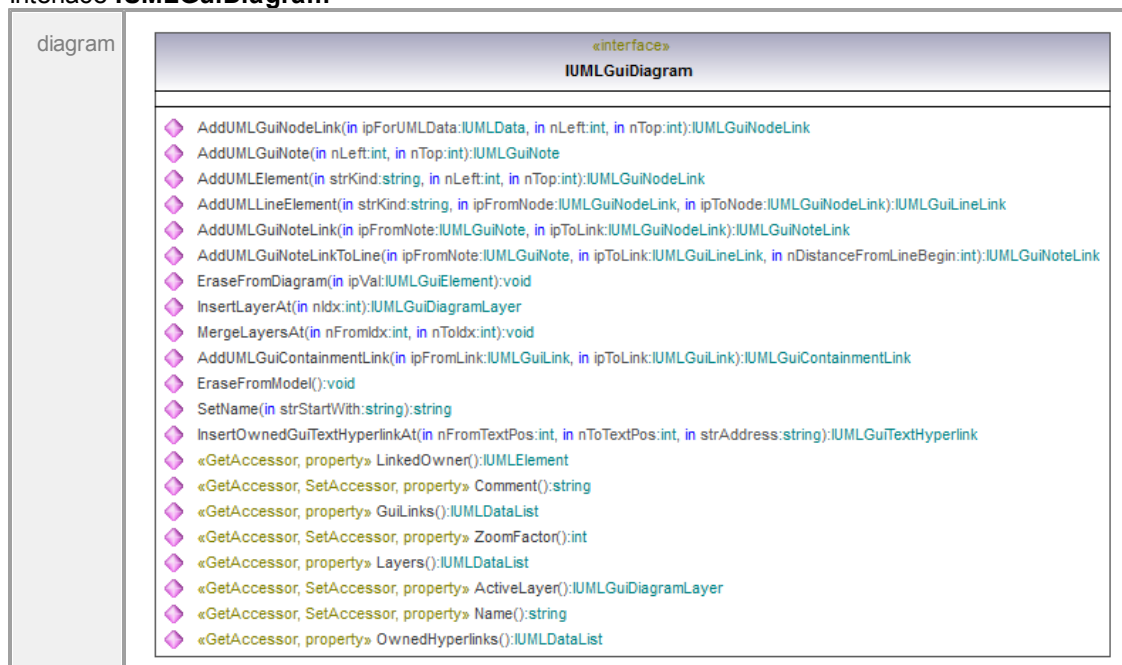
Interface IUMLGuiDeploymentDiagram

UML documentation generated by UModel UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiDiagram

Interface IUMLGuiDiagram



hierarchy		
typedElements	Interface IDiagramWindow Interface IDocument Interface IUMLDataAll Interface IModelGuiRootElement Interface IModelGuiSubDiagramNode	Operation Diagram Operation OpenDiagram Operation InsertOwnedDiagramAtReferencedDiagram Operation InsertOwnedDiagramAtReferencedDiagram Operation ReferencedDiagram
documentation	Represents an UML diagram and contains all layers, nodes (represented as IModelGuiNodeLink) and lines (represented as IModelGuiLineLink). Use the property GuiLinks to access these.	

Operation [IModelGuiDiagram::ActiveLayer](#)

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IModelGuiDiagramLayer			

Operation [IModelGuiDiagram::AddUMLElement](#)

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	in	string			
	nLeft	in	int			
	nTop	in	int			
	return	return	IModelGuiNodeLink			
documentation	Adds a new UML element (e.g. UMLClass , UMLPackage ,...) to the model and shows it with a new IModelGuiNodeLink on the diagram.					

Operation [IModelGuiDiagram::AddUMLGuiContainmentLink](#)

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromLink	in	IModelGuiLink			
	ipToLink	in	IModelGuiLink			
	return	return	IModelGuiContainmentLink			

Operation [IModelGuiDiagram::AddUMLGuiNodeLink](#)

parameter	name	direction	type	type modifier	multiplicity	default
	ipForUMLData	in	IModelData			
	nLeft	in	int			
	nTop	in	int			
	return	return	IModelGuiNodeLink			
documentation	Adds a new IModelGuiNodeLink for an existing UML element (e.g. UMLClass , UMLPackage ,...) on the diagram.					

Operation **IUMLGuiDiagram::AddUMLGuiNote**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	in	int			
	nTop	in	int			
	return	return	IUMLGuiNote			

Operation **IUMLGuiDiagram::AddUMLGuiNoteLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	in	IUMLGuiNote			
	ipToLink	in	IUMLGuiNodeLink			
	return	return	nk IUMLGuiNoteLink nk			

Operation **IUMLGuiDiagram::AddUMLGuiNoteLinkToLine**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	in	IUMLGuiNote			
	ipToLink	in	IUMLGuiLineLink nk			
	nDistanceFromLineBegin	in	int			
	return	return	IUMLGuiNoteLink nk			

Operation **IUMLGuiDiagram::AddUMLLineElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	in	string			
	ipFromNode	in	IUMLGuiNodeLink nk			
	ipToNode	in	IUMLGuiNodeLink nk			
	return	return	IUMLGuiLineLink nk			
documentation	Adds a new UML line element (e.g. IUMLGeneralization , IUMLAssociation ,...) to the model and shows it with a new IUMLGuiLineLink on the diagram.					

Operation **IUMLGuiDiagram::Comment**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLGuiDiagram::EraseFromDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
	ipVal	in	IUMLGuiElement nt			
	return	return	void			
documentation	Use this function to erase the element from the diagram only. Use IUMLElement::EraseFromModel to erase from the model and all diagrams.					

Operation **IUMLGuiDiagram::EraseFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IUMLGuiDiagram::GuiLinks**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of elements of type IUMLGuiLink which are displayed directly on this diagram. Usually, these are IUMLGuiNodeLinks and IUMLGuiLineLinks .					

Operation **IUMLGuiDiagram::InsertLayerAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLGuiDiagramLayer			

Operation **IUMLGuiDiagram::InsertOwnedGuiTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nFromTextPos	in	int			
	nToTextPos	in	int			
	strAddress	in	string			
	return	return	IUMLGuiTextHyperlink			

Operation **IUMLGuiDiagram::Layers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of all layers in the diagram. The list contains elements of type IUMLGuiDiagramLayer .					

Operation **IUMLGuiDiagram::LinkedOwner**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLElement			

Operation **IUMLGuiDiagram::MergeLayersAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nFromIdx	in	int			
	nToIdx	in	int			
	return	return	void			

Operation **IUMLGuiDiagram::Name**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLGuiDiagram::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLGuiDiagram::SetName**

parameter	name	direction	type	type modifier	multiplicity	default
	strStartWith return	in return	string string			

Operation **IUMLGuiDiagram::ZoomFactor**

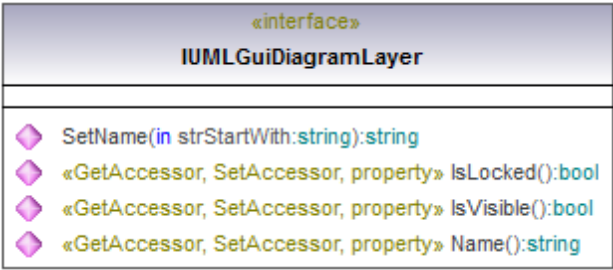
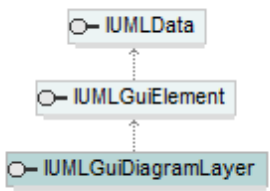
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiDiagramLayer

Interface **IUMLGuiDiagramLayer**

diagram	 <pre> «interface» IUMLGuiDiagramLayer +SetName(in strStartWith:string):string +«GetAccessor, SetAccessor, property» IsLocked():bool +«GetAccessor, SetAccessor, property» IsVisible():bool +«GetAccessor, SetAccessor, property» Name():string </pre>				
hierarchy	 <pre> classDiagram IUMLData < -- IUMLGuiElement IUMLGuiElement < -- IUMLGuiDiagramLayer </pre>				
typedElements	Interface IUMLDataAll Operation ActiveLayer InsertLayerAtLayer Interface IUMLGuiDiagram Operation ActiveLayer InsertLayerAtLayer Interface IUMLGuiLink Operation Layer				
documentation	Represents a layer on an IUMLGuiDiagram . Makes it possible to group elements on a diagram into categories, to lock/unlock them and to make them visible or invisible.				

Operation **IUMLGuiDiagramLayer::IsLocked**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLGuiDiagramLayer::IsVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLGuiDiagramLayer::Name**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLGuiDiagramLayer::SetName**

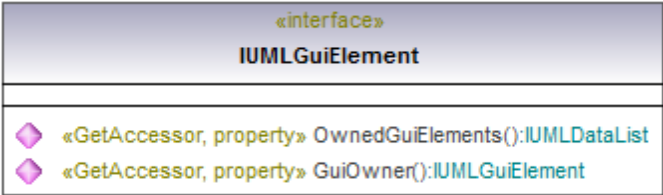
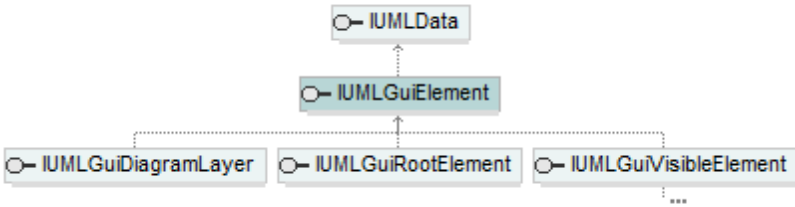
parameter	name	direction	type	type modifier	multiplicity	default
	strStartWith return	in return	string string			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiElement

Interface **IUMLGuiElement**

diagram		
hierarchy		
typedElements	Interface IUMLDataAll Interface IUMLGuiDiagram Interface IUMLGuiElement Interface IUMLGuiLineLink	Operation EraseFromDiagram GuiOwner LineBegin LineEnd Operation EraseFromDiagram Operation GuiOwner Operation LineBegin LineEnd
documentation	The base class for all graphical objects.	

Operation **IUMLGuiElement::GuiOwner**

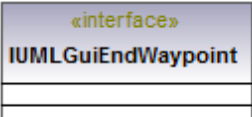
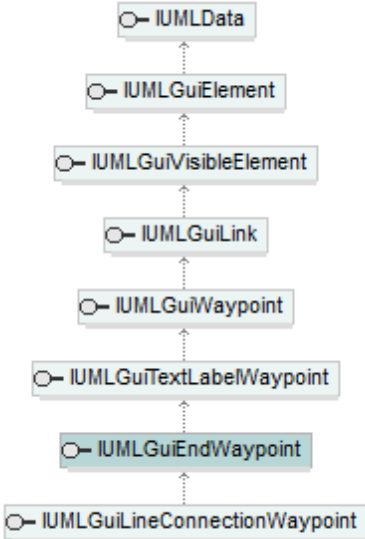
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLGuiElement			

Operation **IUMLGuiElement::OwnedGuiElements**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	Returns a derived list of all owned Gui elements. All elements in this list are a subtype if IUMLGuiElement .					

UModelAPI - IUMLGuiEndWaypoint

Interface **IUMLGuiEndWaypoint**

diagram	 The diagram shows a UML interface box for IUMLGuiEndWaypoint. The box is labeled with «interface» in green and the name IUMLGuiEndWaypoint in bold. It has two empty compartments for attributes and operations.
hierarch y	 The hierarchy diagram shows a vertical stack of class boxes connected by dashed arrows pointing upwards, indicating inheritance. The classes, from top to bottom, are: IUMLData, IUMLGuiElement, IUMLGuiVisibleElement, IUMLGuiLink, IUMLGuiWaypoint, IUMLGuiTextLabelWaypoint, IUMLGuiEndWaypoint (highlighted in light blue), and IUMLGuiLineConnectionWaypoint.
documen tation	A special waypoint w hich only occurs at the end or the begin of a line represented by a IUMLGuiLineLink .

UModelAPI - IUMLGuiExtensionDiagram

Interface **IUMLGuiExtensionDiagram**

diagram	 The diagram shows a UML interface box for IUMLGuiExtensionDiagram. The box is labeled with «interface» in green and the name IUMLGuiExtensionDiagram in bold. It has two empty compartments for attributes and operations.

hierarchy	<pre> classDiagram class IUMLData class IUMLGuiElement class IUMLGuiVisibleElement class IUMLGuiDiagram class IUMLGuiExtensionDiagram class IUMLGuiBPMNDiagram class IUMLGuiXMLSchemaDiagram class IUMLGuiBPMN2ChoreographyDiagram class IUMLGuiBPMN2CollaborationDiagram class IUMLGuiBPMN2Diagram IUMLData < -- IUMLGuiElement IUMLGuiElement < -- IUMLGuiVisibleElement IUMLGuiVisibleElement < -- IUMLGuiDiagram IUMLGuiDiagram < -- IUMLGuiExtensionDiagram IUMLGuiExtensionDiagram < -- IUMLGuiBPMNDiagram IUMLGuiExtensionDiagram < -- IUMLGuiXMLSchemaDiagram IUMLGuiExtensionDiagram < -- IUMLGuiBPMN2ChoreographyDiagram IUMLGuiExtensionDiagram < -- IUMLGuiBPMN2CollaborationDiagram IUMLGuiExtensionDiagram < -- IUMLGuiBPMN2Diagram </pre>
documentation	This diagram type is the base for all UModel specific extension diagrams (for example BPMN diagrams, XML Schema Diagrams) and not a diagram type for itself.

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiInteractionDiagram

Interface IUMLGuiInteractionDiagram

diagram	<pre> classDiagram class IUMLGuiInteractionDiagram { <<interface>> } </pre>
hierarchy	<pre> classDiagram class IUMLData class IUMLGuiElement class IUMLGuiVisibleElement class IUMLGuiDiagram class IUMLGuiBehaviorDiagram class IUMLGuiInteractionDiagram class IUMLGuiCommunicationDiagram class IUMLGuiInteractionOverviewDiagram class IUMLGuiSequenceDiagram class IUMLGuiTimingDiagram IUMLData < -- IUMLGuiElement IUMLGuiElement < -- IUMLGuiVisibleElement IUMLGuiVisibleElement < -- IUMLGuiDiagram IUMLGuiDiagram < -- IUMLGuiBehaviorDiagram IUMLGuiBehaviorDiagram < -- IUMLGuiInteractionDiagram IUMLGuiInteractionDiagram < -- IUMLGuiCommunicationDiagram IUMLGuiInteractionDiagram < -- IUMLGuiInteractionOverviewDiagram IUMLGuiInteractionDiagram < -- IUMLGuiSequenceDiagram IUMLGuiInteractionDiagram < -- IUMLGuiTimingDiagram </pre>

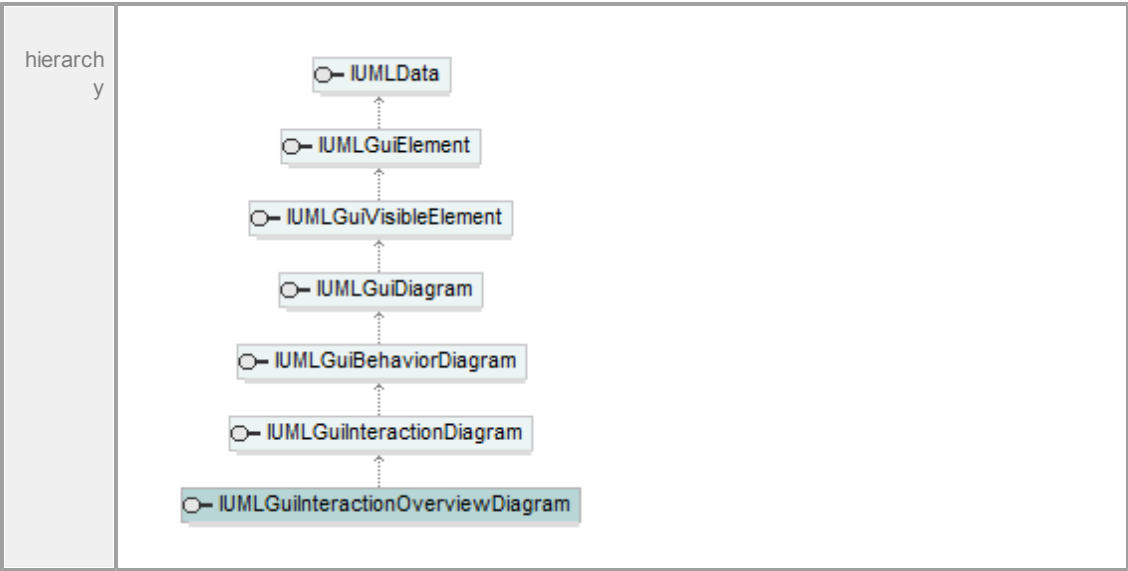
UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiInteractionOverviewDiagram

Interface IUMLGuiInteractionOverviewDiagram

diagram	<pre> classDiagram class IUMLGuiInteractionOverviewDiagram { <<interface>> } </pre>



UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiLabeledRelativeNodeLink

Interface IUMLGuiLabeledRelativeNodeLink

diagram	<pre>classDiagram; class IUMLGuiLabeledRelativeNodeLink { <<interface>> <<GetAccessor, SetAccessor, property>> Label():string;</pre>
hierarchy	<pre>graph TD; IUMLData < > IUMLGuiElement; IUMLGuiElement < > IUMLGuiVisibleElement; IUMLGuiVisibleElement < > IUMLGuiLink; IUMLGuiLink < > IUMLGuiRelativeNodeLink; IUMLGuiRelativeNodeLink < > IUMLGuiLabeledRelativeNodeLink;</pre>
documentation	This special gui link is used for elements w hich are relative to another node and have a label, for example for the names of Messages on Communication diagrams.

Operation IUMLGuiLabeledRelativeNodeLink::Label

paramete	name	direction	type	type modifier	multiplicity	default

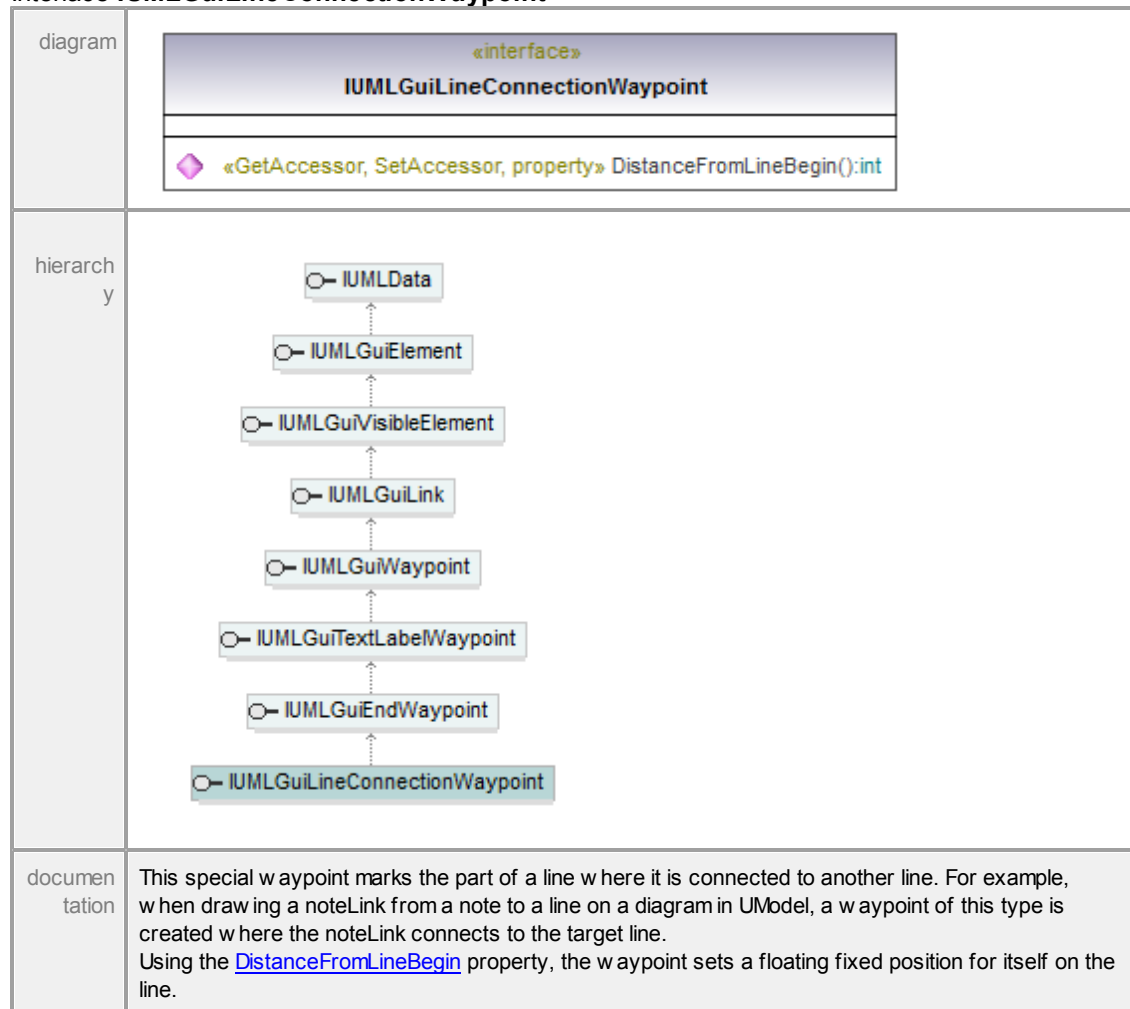
r	return	return	string

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiLineConnectionWaypoint

Interface IUMLGuiLineConnectionWaypoint



Operation IUMLGuiLineConnectionWaypoint::DistanceFromLineBegin

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiLineLink

Interface **IUMLGuiLineLink**

diagram	«interface» IUMLGuiLineLink ◆ InsertWaypointAt(in nldx:int):IUMLGuiWaypoint ◆ EraseWaypointAt(in nldx:int):void ◆ «GetAccessor, property» Waypoints():IUMLDataList ◆ «GetAccessor, property» LineConnectionWaypoints():IUMLDataList ◆ «GetAccessor, property» MiddleWaypoint():IUMLGuiMiddleWaypoint ◆ «GetAccessor, property» AllWaypoints():IUMLDataList ◆ «GetAccessor, property» LineBegin():IUMLGuiElement ◆ «GetAccessor, property» LineEnd():IUMLGuiElement	
hierarch y	<pre>graph BT IUMLData -- > IUMLGuiElement IUMLGuiElement -- > IUMLGuiVisibleElement IUMLGuiVisibleElement -- > IUMLGuiLink IUMLGuiLink -- > IUMLGuiLineLink IUMLGuiLineLink -.-> IUMLGuiCommunicationLink IUMLGuiLineLink -.-> IUMLGuiNoteLink IUMLGuiLineLink -.-> IUMLGuiTimingDiagramMessage IUMLGuiLineLink -.-> IUMLGuiContainmentLink</pre>	
typedE lements	Interface IUMLDataAll Interface IUMLGuiDiagram	Operation AddUMLGuiNoteLinkToLine AddUMLLineElement Operation AddUMLGuiNoteLinkToLine AddUMLLineElement
documen tation	This interface represents a line on a diagram. There are some special lines deriving from this interface available as well. A line is composed of multiple but at least 2 waypoints which are connected to each other, which can be accessed using the AllWaypoints property. Two of these waypoints are usually of type IUMLGuiEndWaypoint. There may be also a Middle waypoint accessible with the property MiddleWaypoint which can be used for example to access textlabels and a LineConnectionWaypoint when the line is connected to another line, like to a IUMLGuiNoteLink. The LineBegin property refers the first object and LineEnd property refers the second graphical object which the line connects.	

Operation **IUMLGuiLineLink::AllWaypoints**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documen tation	A derived list of all waypoints which are part of this line. All elements in this list are of type (or subtype of) IUMLGuiWaypoint .					

Operation **IUMLGuiLineLink::EraseWaypointAt**

paramete r	name	direction	type	type modifier	multiplicity	default
	nldx	in	int			

	return	return	void

Operation **IUMLGuiLineLink::InsertWaypointAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int IUMLGuiWaypoint int			

Operation **IUMLGuiLineLink::LineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiElement			
documentation	A reference to the first object, where the line starts.					

Operation **IUMLGuiLineLink::LineConnectionWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of all waypoints which connect the line with other lines. All elements in this list are of type (or subtype of) IUMLGuiWaypoint .					

Operation **IUMLGuiLineLink::LineEnd**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiElement			
documentation	A reference to the second object, where the line ends.					

Operation **IUMLGuiLineLink::MiddleWaypoint**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiMiddleWaypoint			

Operation **IUMLGuiLineLink::Waypoints**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	A list of all waypoints which form the vertices of this line. All elements in this list are of type (or subtype of) IUMLGuiWaypoint .					

UModelAPI - IUMLGuiLink

Interface **IUMLGuiLink**

diagram	«interface» IUMLGuiLink ◆ «GetAccessor, property» Element():IUMLElement ◆ «GetAccessor, property» AttachedNodes():IUMLDataList ◆ «GetAccessor, property» RelativeNodes():IUMLDataList ◆ «GetAccessor, SetAccessor, property» Layer():IUMLGuiDiagramLayer	
hierarchy		
types	Interface IDiagramWindow Interface IUMLDataAll Interface IUMLGuiAttachedNode Interface IUMLGuiCommunicationDiagram Interface IUMLGuiDiagram	Operation FocusedGuiElement ScrollToGuiElement SelectGuiElement Operation AddUMLGuiContainmentLink AttachedTo Operation AttachedTo Operation AddUMLGuiCommunicationLink Operation AddUMLGuiContainmentLink
documentation	A GuiLink represents a graphical object on a diagram which is connected to an element from the UML (like a Class, an Interface or a Lifeline). This connected object can be accessed using the Element property. IUMLGuiLinks further can have attached nodes (IUMLGuiAttachedNode) which appear when necessary, like Tagged Values and are associated with a Layer on the diagram.	

Operation **IUMLGuiLink::AttachedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	Returns a derived list of all attached nodes of this element. All elements in this list are of type (or subtype) of IUMLGuiAttachedNode .					

Operation **IUMLGuiLink::Element**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLElement			

Operation **IUMLGuiLink::Layer**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiDiagramLayer			

Operation **IUMLGuiLink::RelativeNodes**

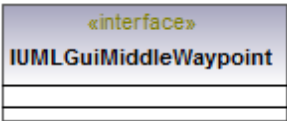
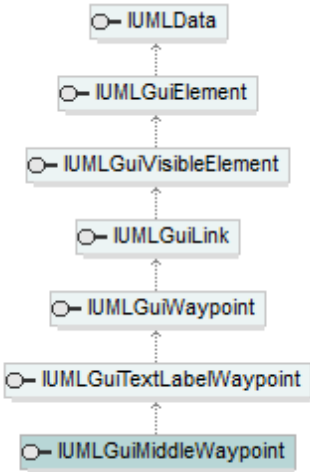
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	Returns a list of relative nodes to this gui link. The list contains only elements of type (or subtype of) IUMLGuiRelativeNodeLink .					

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiMiddleWaypoint

Interface **IUMLGuiMiddleWaypoint**

diagram	 <pre> classDiagram class IUMLGuiMiddleWaypoint { <<interface>> } </pre>	
hierarchy	 <pre> classDiagram IUMLData < .. IUMLGuiElement IUMLGuiElement < .. IUMLGuiVisibleElement IUMLGuiVisibleElement < .. IUMLGuiLink IUMLGuiLink < .. IUMLGuiWaypoint IUMLGuiWaypoint < .. IUMLGuiTextLabelWaypoint IUMLGuiTextLabelWaypoint < .. IUMLGuiMiddleWaypoint </pre>	
typedElements	Interface IUMLDataAll Interface IUMLGuiLineLink	Operation MiddleWaypoint Operation MiddleWaypoint
documentation	A middle waypoint is a special waypoint on a line (IUMLGuiLineLink) which appears in the center of the line and can have text labels attached to it.	

UModelAPI - IUMLGuiNodeLink

Interface IUMLGuiNodeLink

diagram	«interface» IUMLGuiNodeLink ◆ SetRect(in nLeft:int, in nTop:int, in nRight:int, in nBottom:int):void ◆ MoveTo(in nLeft:int, in nTop:int):void ◆ IsElementVisible(in ipElement:IUMLElement):bool ◆ SetElementVisible(in ipElement:IUMLElement, in bVisible:bool):void ◆ AddOwnedGuiNodeLink(in ipForUMLData:IUMLGuiNodeLink):void ◆ «GetAccessor, SetAccessor, property» Top():int ◆ «GetAccessor, SetAccessor, property» Left():int ◆ «GetAccessor, SetAccessor, property» Bottom():int ◆ «GetAccessor, SetAccessor, property» Right():int ◆ «GetAccessor, property» OwnedGuiNodeLinks():IUMLDataList ◆ «GetAccessor, property» OwningGuiNodeLink():IUMLGuiNodeLink
hierarchy	○ IUMLData ○ IUMLGuiElement ○ IUMLGuiVisibleElement ○ IUMLGuiLink ○ IUMLGuiNodeLink ○ IUMLGuiAttachedNode ○ IUMLGuiConstrainedNode ○ IUMLGuiNote ○ IUMLGuiSeparatedNodeLink ... ○ IUMLGuiSubDiagramNode ○ IUMLGuiTickMark ○ IUMLGuiTimingDiagramLifeline
typedElements	Interface IUMLDataAll Operation AddOwnedGuiNodeLink AddUMLElement AddUMLGuiNodeLink AddUMLGuiNoteLink AddUMLLineElement OwningGuiNodeLink Interface IUMLGuiDiagram Operation AddUMLElement AddUMLGuiNodeLink AddUMLGuiNoteLink AddUMLLineElement Interface IUMLGuiNodeLink Operation AddOwnedGuiNodeLink OwningGuiNodeLink
documentation	A GuiNodeLink represents a graphical object on a diagram w hich usually represents an element from the UML (for example a Class, an Interface or a Lifeline). It has a position defined by a rectangle w hich can be positioned freely on the diagram. A GuiNodeLink can itself contain other GuiNodeLinks, for example w hen displaying a big state in a state machine diagram w hich contains other, smaller substates. Additionally, if the GuiNodeLink displays cells on it, like for example operations and properties on a class, it can store for each element show n if the element should be visible or not. Use the SetElementVisible and IsElementVisible functions for this.

Operation IUMLGuiNodeLink::AddOwnedGuiNodeLink

paramete	name	direction	type	type modifier	multiplicity	default

r	ipForUMLData	in	IUMLGuiNodeLink			
	return	return	void			

Operation **IUMLGuiNodeLink::Bottom**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLGuiNodeLink::IsElementVisible**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipElement	in	IUMLElement			
	return	return	bool			

Operation **IUMLGuiNodeLink::Left**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLGuiNodeLink::MoveTo**

parameter	name	direction	type	type modifier	multiplicity	default
r	nLeft	in	int			
	nTop	in	int			
	return	return	void			

Operation **IUMLGuiNodeLink::OwnedGuiNodeLinks**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			
documentation	Returns a list of all owned gui node links, all nodes which are directly contained in this node. All elements in this list are of type (or subtype of) IUMLGuiLink .					

Operation **IUMLGuiNodeLink::OwningGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiNodeLink			

Operation **IUMLGuiNodeLink::Right**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLGuiNodeLink::SetElementVisible**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipElement	in	IUMLElement			
	bVisible	in	bool			
	return	return	void			

Operation **IUMLGuiNodeLink::SetRect**

parameter	name	direction	type	type modifier	multiplicity	default
r	nLeft	in	int			

	nTop	in	int
	nRight	in	int
	nBottom	in	int
	return	return	void

Operation **IUMLGuiNodeLink::Top**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

UML documentation generated by [UModel UML Editor http://www.altova.com/umodel](http://www.altova.com/umodel) 08/03/15 13:42:06

UModelAPI - IUMLGuiNote

Interface **IUMLGuiNote**

diagram	«interface» IUMLGuiNote - InsertOwnedGuiTextHyperlinkAt(in nFromTextPos:int, in nToTextPos:int, in strAddress:string):IUMLGuiTextHyperlink «GetAccessor, SetAccessor, property» NoteText():string «GetAccessor, property» OwnedHyperlinks():IUMLDataList	
hierarchy	<pre>graph TD IUMLData < -- IUMLGuiElement IUMLGuiElement < -- IUMLGuiVisibleElement IUMLGuiVisibleElement < -- IUMLGuiLink IUMLGuiLink < -- IUMLGuiNodeLink IUMLGuiNodeLink < -- IUMLGuiNote</pre>	
typedElements	Interface IUMLDataAll	Operation AddUMLGuiNote AddUMLGuiNodeLink AddUMLGuiNoteLinkToLine
	Interface IUMLGuiDiagram	Operation AddUMLGuiNote AddUMLGuiNodeLink AddUMLGuiNoteLinkToLine
documentation	A IUMLGuiNote is the graphical object resembling a note on UModel diagrams displaying a text comment. It provides access to the note text and a list of hyperlinks in this text. These hyperlinks are nothing more than a list of URLs together with an begin and end number referencing positions in the text. Any text between a such a begin and end position is displayed as hyperlink and triggers UModel to open the URL when clicked.	

Operation **IUMLGuiNote::InsertOwnedGuiTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nFromTextPos	in	int			
	nToTextPos	in	int			
	strAddress	in	string			
	return	return	IUMLGuiTextHyperlink			

Operation **IUMLGuiNote::NoteText**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLGuiNote::OwnedHyperlinks**

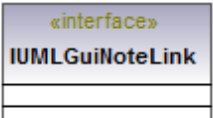
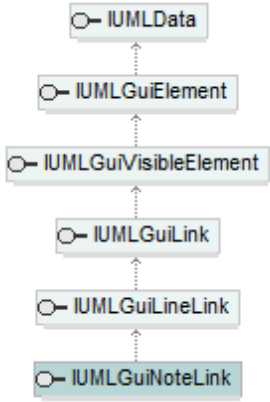
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiNoteLink

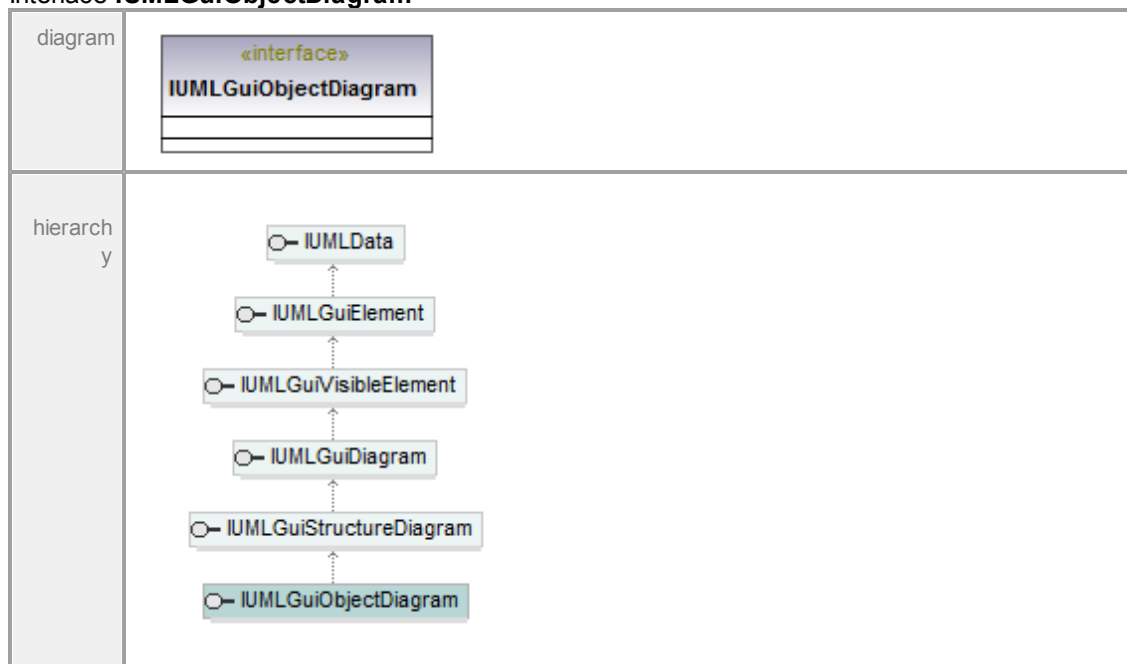
Interface **IUMLGuiNoteLink**

diagram		
hierarchy		
typedElements	Interface IUMLDataAll Interface IUMLGuiDiagram	Operation AddIUMLGuiNoteLink AddIUMLGuiNoteLinkToLine Operation AddIUMLGuiNoteLink AddIUMLGuiNoteLinkToLine
documentation	A notelink is a special IUMLGuiLineLink which connects a IUMLGuiNote with another IUMLGuiLink element. It is displayed as a dotted line.	

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

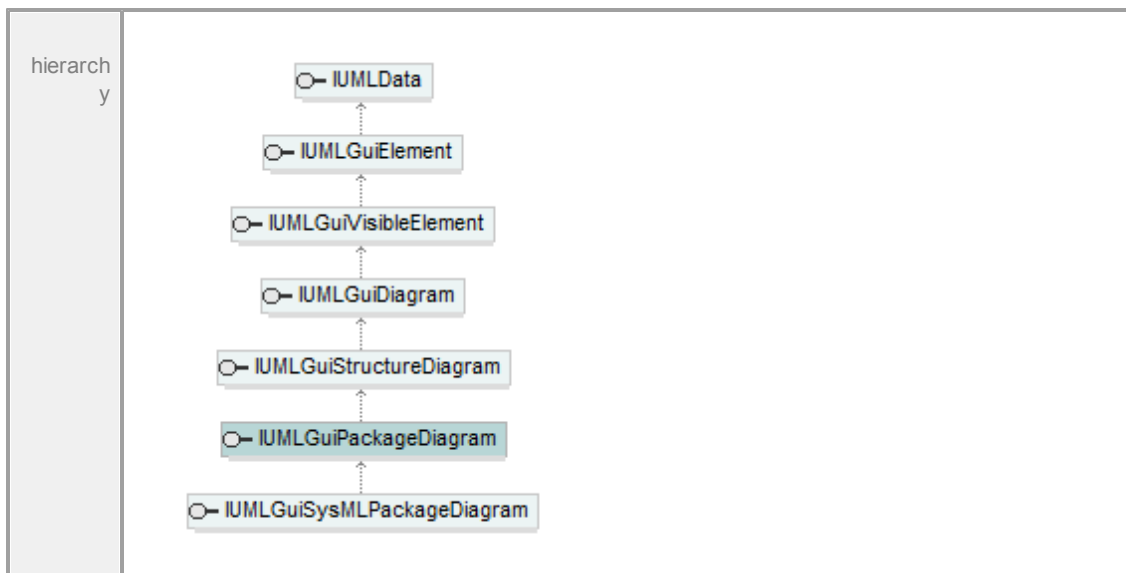
UModelAPI - IUMLGuiObjectDiagram

Interface **IUMLGuiObjectDiagram**UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiPackageDiagram

Interface **IUMLGuiPackageDiagram**

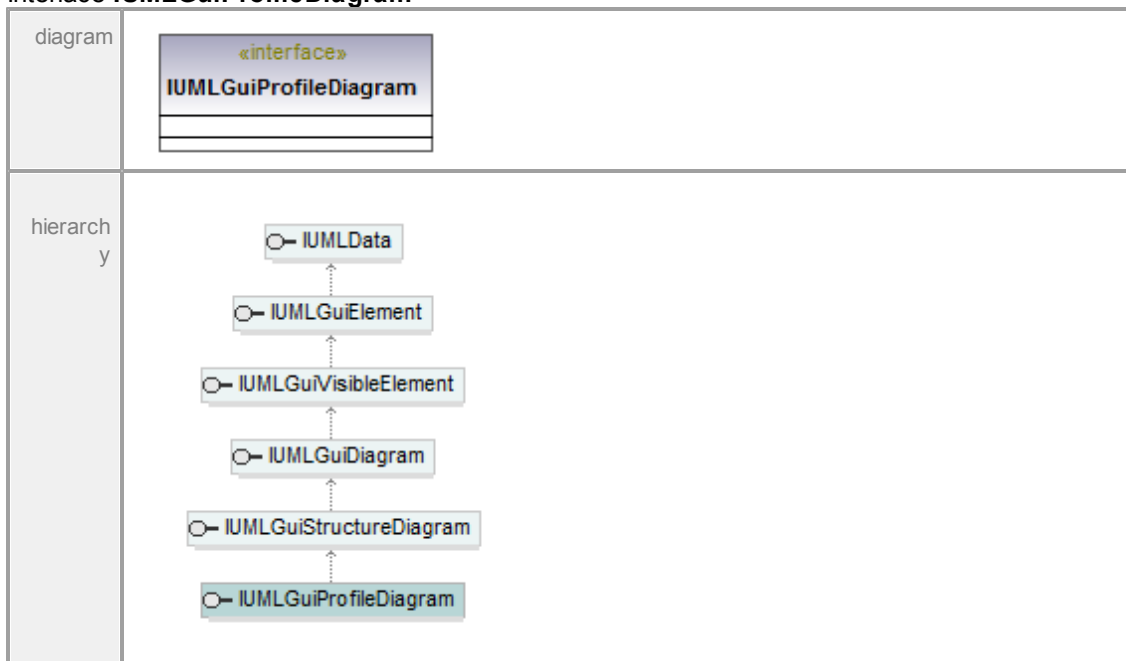


UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiProfileDiagram

Interface IUMLGuiProfileDiagram

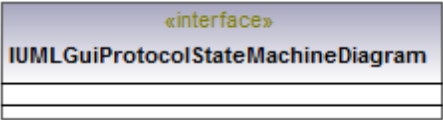
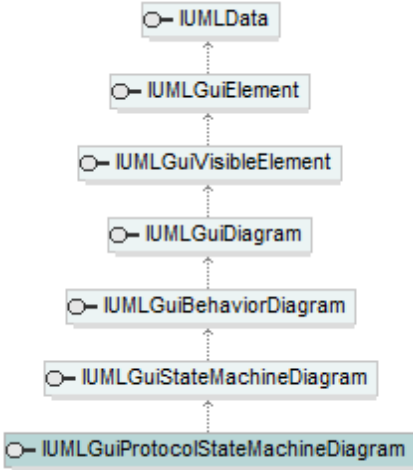


UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

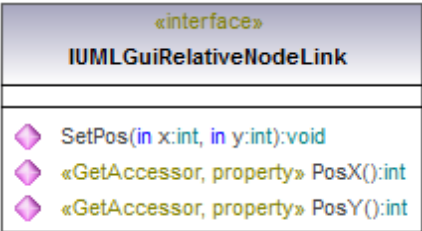
UModelAPI - IUMLGuiProtocolStateMachineDiagram

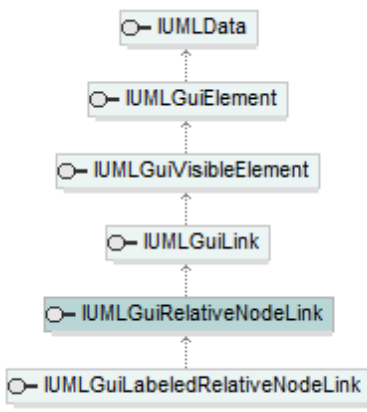
Interface **IUMLGuiProtocolStateMachineDiagram**

diagram	 UML diagram showing the IUMLGuiProtocolStateMachineDiagram interface. The interface is represented by a rectangle with a blue header containing the text «interface» and IUMLGuiProtocolStateMachineDiagram. Below the header is a single empty line.
hierarchy	 UML hierarchy diagram showing the inheritance relationships for IUMLGuiProtocolStateMachineDiagram. The hierarchy is as follows: <pre>graph BT; IUMLData -- > IUMLGuiElement; IUMLGuiElement -- > IUMLGuiVisibleElement; IUMLGuiVisibleElement -- > IUMLGuiDiagram; IUMLGuiDiagram -- > IUMLGuiBehaviorDiagram; IUMLGuiBehaviorDiagram -- > IUMLGuiStateMachineDiagram; IUMLGuiStateMachineDiagram -- > IUMLGuiProtocolStateMachineDiagram</pre> The diagram shows a vertical stack of boxes representing the hierarchy. From top to bottom: IUMLData, IUMLGuiElement, IUMLGuiVisibleElement, IUMLGuiDiagram, IUMLGuiBehaviorDiagram, IUMLGuiStateMachineDiagram, and IUMLGuiProtocolStateMachineDiagram. Each box has a small circle icon on the left. Dashed arrows point from each box to the one immediately below it, indicating inheritance.

UModelAPI - IUMLGuiRelativeNodeLink

Interface **IUMLGuiRelativeNodeLink**

diagram	 UML diagram showing the IUMLGuiRelativeNodeLink interface. The interface is represented by a rectangle with a blue header containing the text «interface» and IUMLGuiRelativeNodeLink. Below the header, the following methods are listed: ◆ SetPos(in x:int, in y:int):void ◆ «GetAccessor, property» PosX():int ◆ «GetAccessor, property» PosY():int

hierarchy	 <pre>graph TD IUMLData < .. IUMLGuiElement IUMLGuiElement < .. IUMLGuiVisibleElement IUMLGuiVisibleElement < .. IUMLGuiLink IUMLGuiLink < .. IUMLGuiRelativeNodeLink IUMLGuiRelativeNodeLink < .. IUMLGuiLabeledRelativeNodeLink</pre>
documentation	This gui link is used for elements which are positioned relative to another node. For example the names of Messages on Communication diagrams use a specialization of this interface.

Operation **IUMLGuiRelativeNodeLink::PosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLGuiRelativeNodeLink::PosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

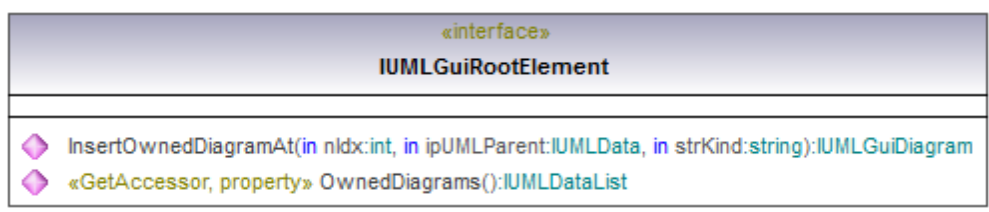
Operation **IUMLGuiRelativeNodeLink::SetPos**

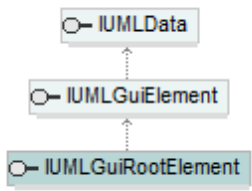
parameter	name	direction	type	type modifier	multiplicity	default
	x	in	int			
	y	in	int			
	return	return	void			

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLGuiRootElement

Interface **IUMLGuiRootElement**

diagram	 <pre>classDiagram class IUMLGuiRootElement { <<interface>> +InsertOwnedDiagramAt(in nldx:int, in ipUMLParent:IUMLData, in strKind:string):IUMLGuiDiagram +GetAccessor, property OwnedDiagrams():IUMLDataList }</pre>

hierarchy	 <pre>graph TD IUMLElement -- > IUMLElement IUMLElementRootElement -- > IUMLElement</pre>
typed Elements	Interface Document Operation GuiRoot
documentation	This is the root interface for all graphical objects and contains all diagrams which exists in the UModel project.

Operation **IUMLElementRootElement::InsertOwnedDiagramAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipUMLParent	in	IUMLElement			
	strKind	in	string			
	return	return	IUMLElementDiagram			

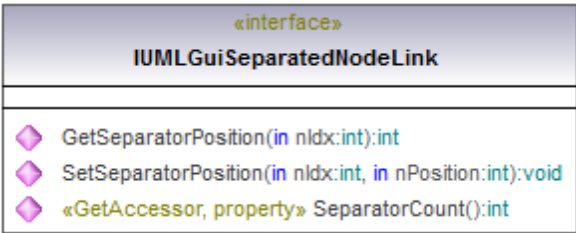
Operation **IUMLElementRootElement::OwnedDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLElementList			
documentation	Returns a list of all diagrams in this UModel project. All elements in this list are of type (or subtype of) IUMLElementDiagram .					

UML documentation generated by [UModel!](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLElementSeparatedNodeLink

Interface **IUMLElementSeparatedNodeLink**

diagram	 <pre>classDiagram class IUMLElementSeparatedNodeLink { <<interface>> GetSeparatorPosition(in nIdx:int):int SetSeparatorPosition(in nIdx:int, in nPosition:int):void «GetAccessor, property» SeparatorCount():int }</pre>

hierarchy	<pre> classDiagram IUMLData < -- IUMLGuiElement IUMLGuiElement < -- IUMLGuiVisibleElement IUMLGuiVisibleElement < -- IUMLGuiLink IUMLGuiLink < -- IUMLGuiNodeLink IUMLGuiNodeLink < -- IUMLGuiSeparatedNodeLink IUMLGuiSeparatedNodeLink < -- IUMLGuiSeparatedNodeLink2D </pre>
documentation	This node link represents a graphical object on a UModel diagram which can be separated into two or more parts by one or more either horizontal or vertical lines. For each line, the position of the separator is stored in this node. This node type is used for example by CombinedFragments, ActivityPartitions and States with regions.

Operation **IUMLGuiSeparatedNodeLink::GetSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	int			

Operation **IUMLGuiSeparatedNodeLink::SeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLGuiSeparatedNodeLink::SetSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	nPosition	in	int			
	return	return	void			

UModelAPI - IUMLGuiSeparatedNodeLink2D

Interface **IUMLGuiSeparatedNodeLink2D**

diagram	<pre> «interface» IUMLGuiSeparatedNodeLink2D +GetHSeparatorPosition(in nIdx:int):int +SetHSeparatorPosition(in nIdx:int, in nPosition:int):void +GetVSeparatorPosition(in nIdx:int):int +SetVSeparatorPosition(in nIdx:int, in nPosition:int):void +«GetAccessor, property» HSeparatorCount():int +«GetAccessor, property» VSeparatorCount():int </pre>
hierarchy	<pre> graph BT IUMLData -- > IUMLGuiElement IUMLGuiElement -- > IUMLGuiVisibleElement IUMLGuiVisibleElement -- > IUMLGuiLink IUMLGuiLink -- > IUMLGuiNodeLink IUMLGuiNodeLink -- > IUMLGuiSeparatedNodeLink IUMLGuiSeparatedNodeLink -- > IUMLGuiSeparatedNodeLink2D </pre>
documentation	This node link represents a graphical object on a UModel diagram which can be separated into parts by one or more horizontal or vertical lines, but in contrast to IUMLGuiSeparatedNodeLink, the node can be subdivided vertically and horizontally at the same time. For each vertical or horizontal separation line, the position of the separator is stored in this node. This node type is used for example by ActivityPartitions.

Operation **IUMLGuiSeparatedNodeLink2D::GetHSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	int			

Operation **IUMLGuiSeparatedNodeLink2D::GetVSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	int			

Operation **IUMLGuiSeparatedNodeLink2D::HSeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLGuiSeparatedNodeLink2D::SetHSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	ndx	in	int			
	nPosition	in	int			
	return	return	void			

Operation **IUMLGuiSeparatedNodeLink2D::SetVSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
	ndx	in	int			
	nPosition	in	int			
	return	return	void			

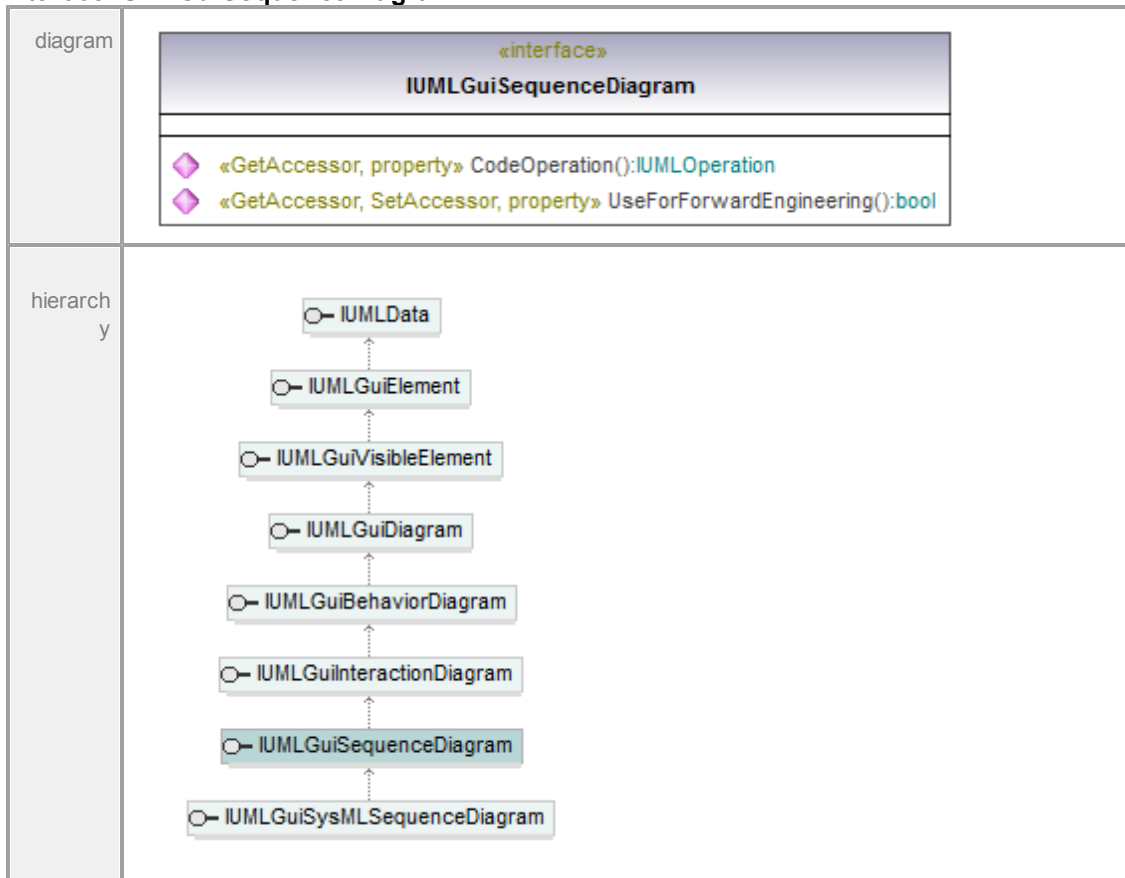
Operation **IUMLGuiSeparatedNodeLink2D::VSeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiSequenceDiagram

Interface **IUMLGuiSequenceDiagram**

Operation **IUMLGuiSequenceDiagram::CodeOperation**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLOperation			

Operation **IUMLGuiSequenceDiagram::UseForForwardEngineering**

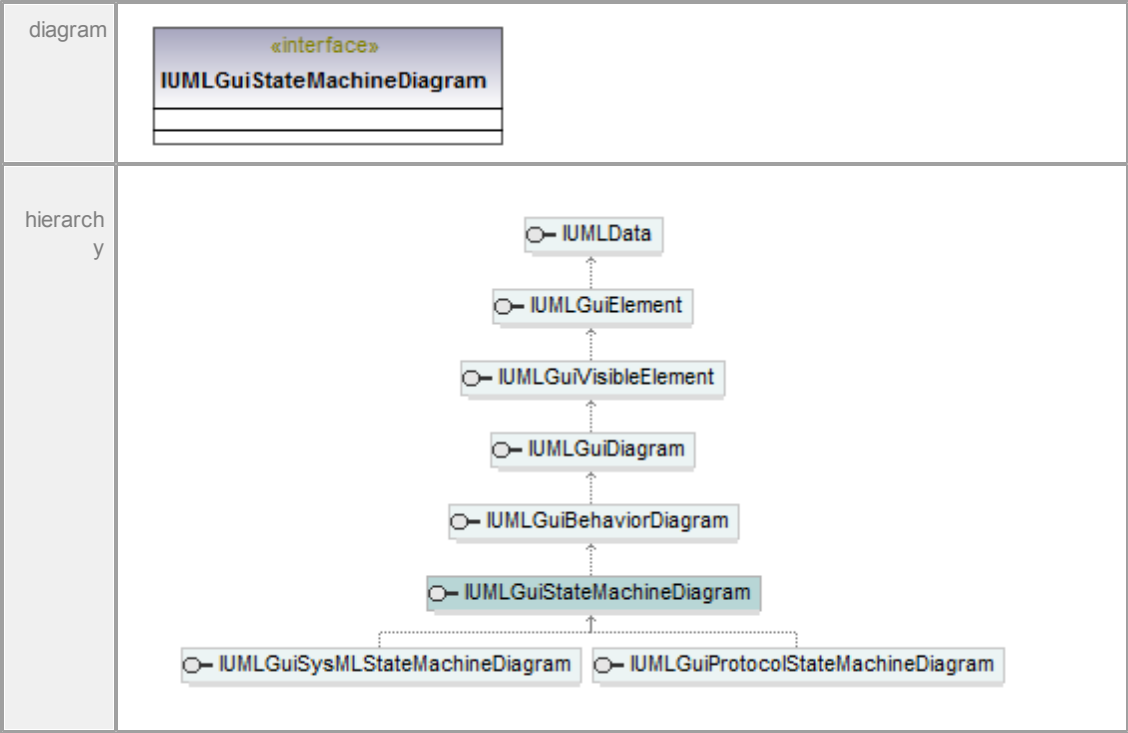
parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiStateMachineDiagram

Interface **IUMLGuiStateMachineDiagram**



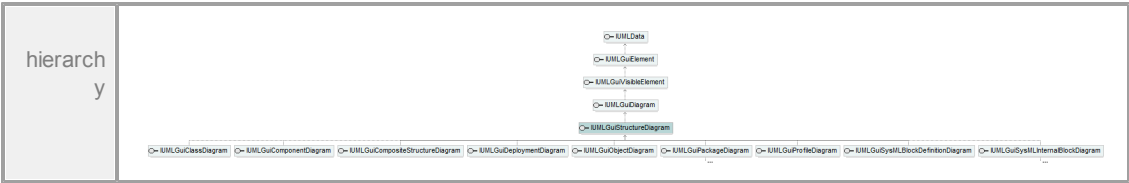
UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiStructureDiagram

Interface **IUMLGuiStructureDiagram**





UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLGuiStyle

Interface IUMLGuiStyle

diagram	«interface» IUMLGuiStyle «GetAccessor, property» Application():IDispatch «GetAccessor, property» Parent():IDispatch «GetAccessor, property» Kind():ENUMUMLGuiStyleKind «GetAccessor, property» Name():string «GetAccessor, SetAccessor, property» Value():string «GetAccessor, property» UsedValue():string
typedElements	Interface IUMLGuiStyles Operation GetStyle Item

Operation IUMLGuiStyle::Application

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation IUMLGuiStyle::Kind

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	ENUMUMLGuiStyleKind			

Operation IUMLGuiStyle::Name

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation IUMLGuiStyle::Parent

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

Operation IUMLGuiStyle::UsedValue

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation IUMLGuiStyle::Value

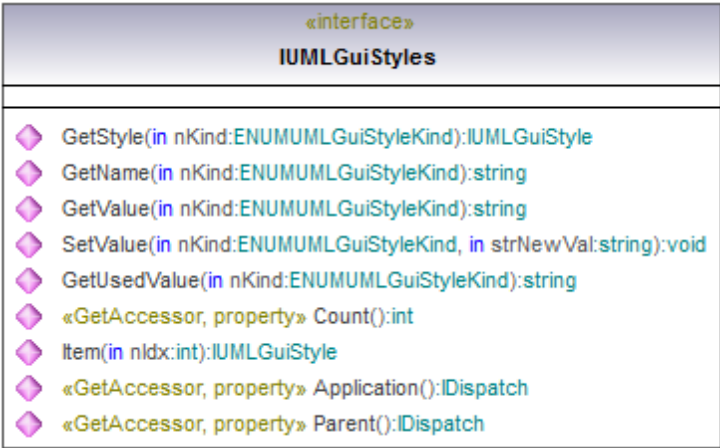
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiStyles

Interface IUMLGuiStyles

diagram					
typedElements	Interface IDocument	Operation ElementFamilyStyles			
		LineStyle			
		NodeStyles			
		ProjectStyles			
	Interface IUMLDataAll	Operation StereotypedElementStyles			
		Styles			
	Interface IUMLGuiVisibleElement	Operation Styles			
	Interface IUMLStereotype	Operation StereotypedElementStyles			

Operation IUMLGuiStyles::Application

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation IUMLGuiStyles::Count

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation IUMLGuiStyles::GetName

parameter	name	direction	type	type modifier	multiplicity	default
r	nKind	in	ENUMUMLGuiStyleKind			
	return	return	string			

Operation IUMLGuiStyles::GetStyle

parameter	name	direction	type	type modifier	multiplicity	default
r	nKind	in	ENUMUMLGuiStyleKind			

	return	return	yleKind IUMLGuiStyle			

Operation **IUMLGuiStyles::GetUsedValue**

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	in	ENUMUMLGuiSt yleKind			
	return	return	string			

Operation **IUMLGuiStyles::GetValue**

parameter	name	direction	type	type modifier	multiplicity	default
	nKind	in	ENUMUMLGuiSt yleKind			
	return	return	string			

Operation **IUMLGuiStyles::Item**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	IUMLGuiStyle			

Operation **IUMLGuiStyles::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IDispatch			

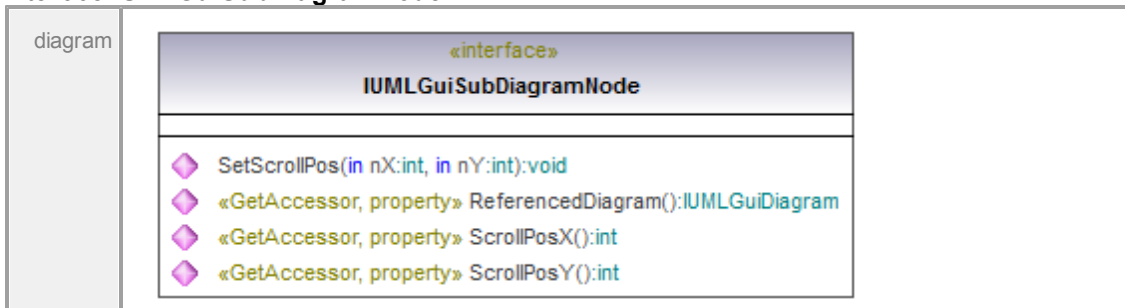
Operation **IUMLGuiStyles::SetValue**

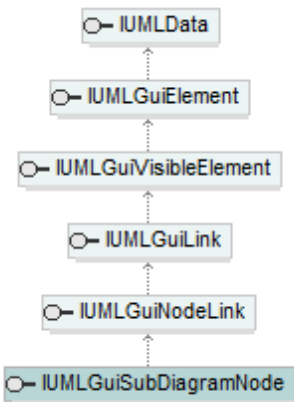
parameter	name	direction	type	type modifier	multiplicity	default
	nKind	in	ENUMUMLGuiSt yleKind			
	strNewVal	in	string			
	return	return	void			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiSubDiagramNode

Interface **IUMLGuiSubDiagramNode**

hierarch y	 <pre>graph TD IUMLData -- > IUMLGuiElement IUMLGuiElement -- > IUMLGuiVisibleElement IUMLGuiVisibleElement -- > IUMLGuiLink IUMLGuiLink -- > IUMLGuiNodeLink IUMLGuiNodeLink -- > IUMLGuiSubDiagramNode</pre>
documen tation	The sub diagram node represents a node link on a diagram w hich again includes another diagram. This is used for example on interaction overview diagrams to display sequence, communication and timing diagrams inside nodes. The property ReferencedDiagram controls the diagrams w hich is show n inside the node.

Operation **IUMLGuiSubDiagramNode::ReferencedDiagram**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLGuiDiagram m			

Operation **IUMLGuiSubDiagramNode::ScrollPosX**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLGuiSubDiagramNode::ScrollPosY**

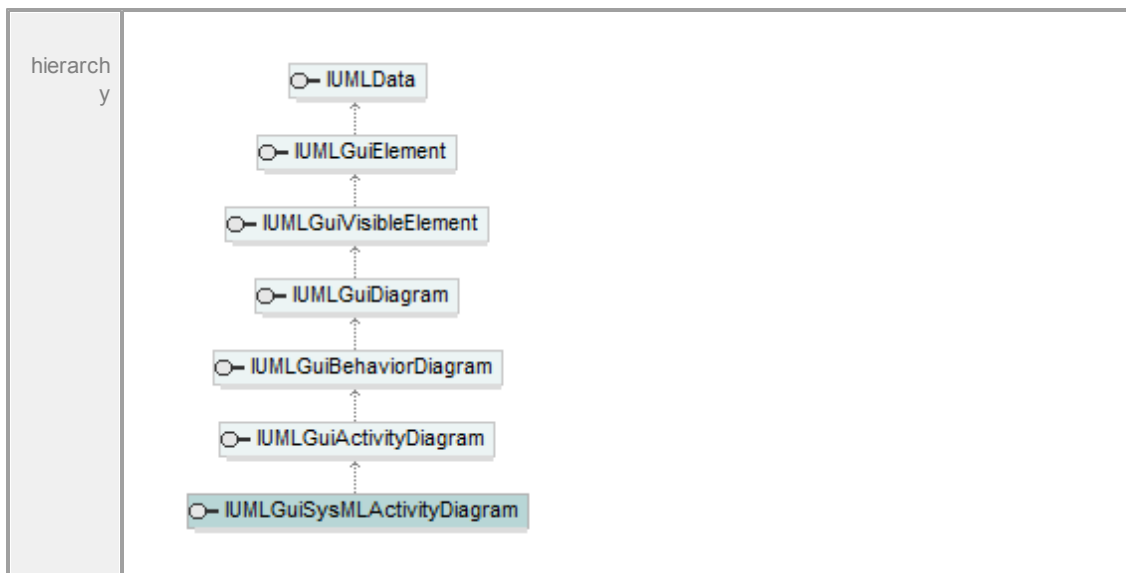
paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLGuiSubDiagramNode::SetScrollPos**

paramete r	name	direction	type	type modifier	multiplicity	default
	nX	in	int			
	nY	in	int			
	return	return	void			

Interface **IUMLGuiSysMLActivityDiagram**

diagram	 <pre>classDiagram class IUMLGuiSysMLActivityDiagram { <<interface>> }</pre>

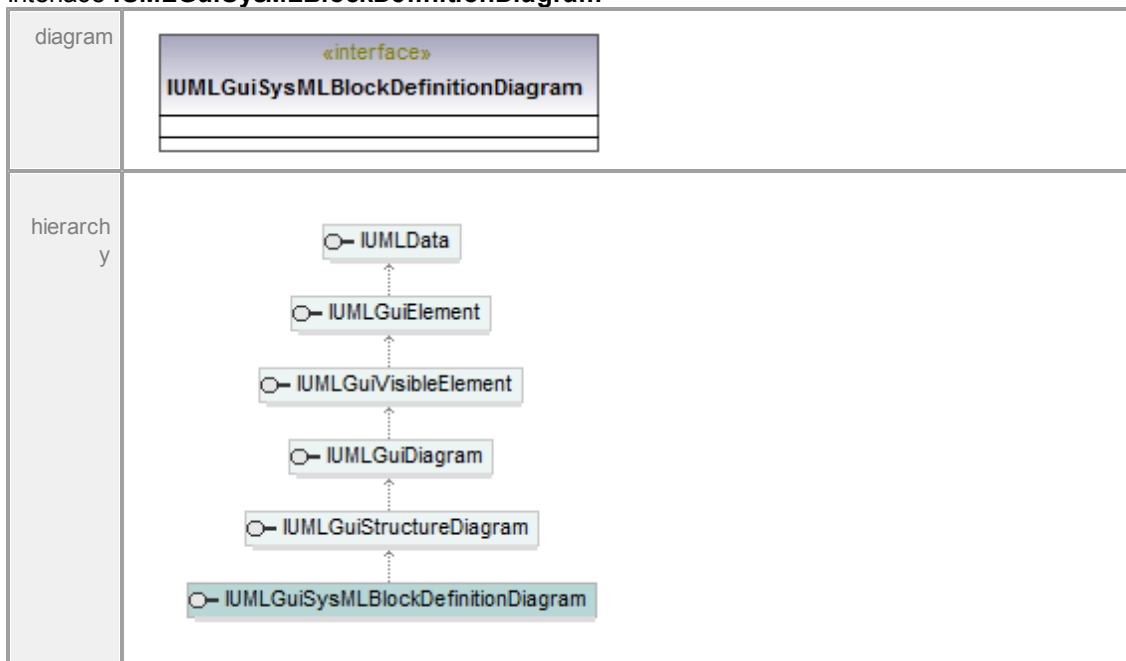


UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiSysMLBlockDefinitionDiagram

Interface IUMLGuiSysMLBlockDefinitionDiagram

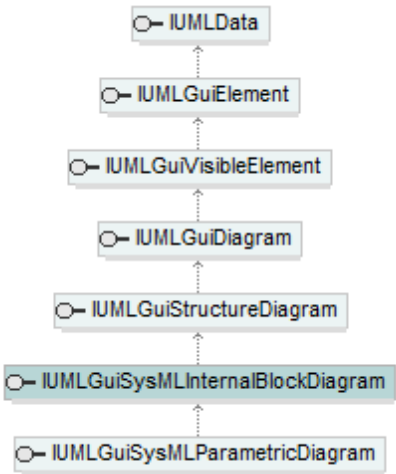


UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

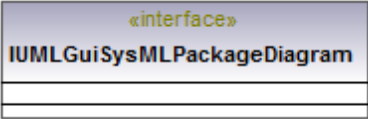
UModelAPI - IUMLGuiSysMLInternalBlockDiagram

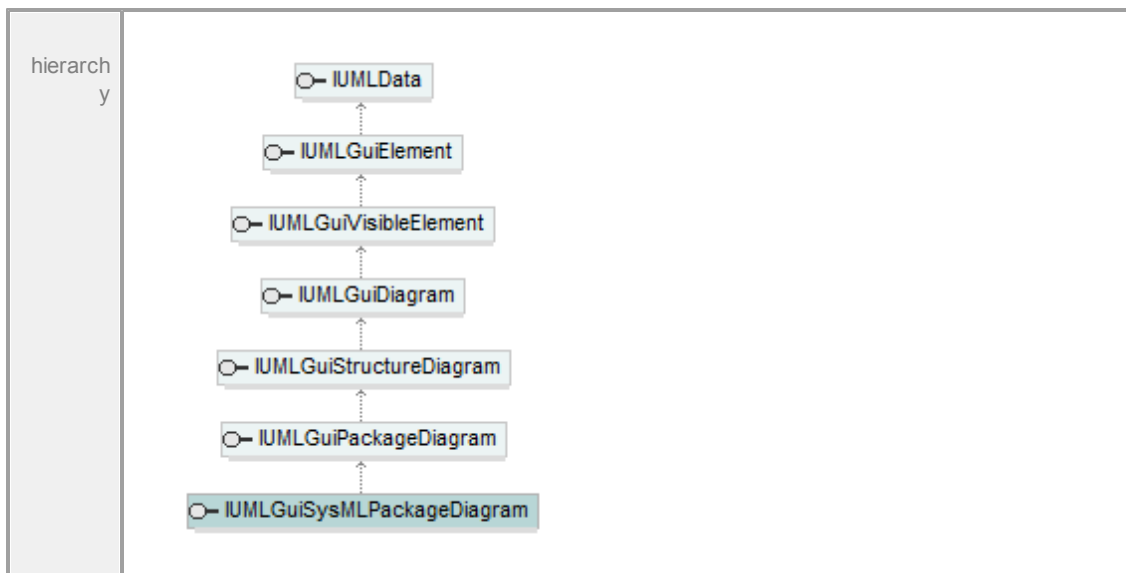
Interface **IUMLGuiSysMLInternalBlockDiagram**

diagram	 UML diagram showing the IUMLGuiSysMLInternalBlockDiagram interface. The diagram is a rectangle with a light blue header containing the text «interface» and IUMLGuiSysMLInternalBlockDiagram. Below the header are two empty lines for attributes or methods.
hierarchy	 UML hierarchy diagram showing inheritance. The diagram consists of a vertical stack of boxes connected by dashed arrows pointing downwards. The boxes are: IUMLData, IUMLGuiElement, IUMLGuiVisibleElement, IUMLGuiDiagram, IUMLGuiStructureDiagram, IUMLGuiSysMLInternalBlockDiagram, and IUMLGuiSysMLParametricDiagram. Each box has a small circle icon to its left.

UModelAPI - IUMLGuiSysMLPackageDiagram

Interface **IUMLGuiSysMLPackageDiagram**

diagram	 UML diagram showing the IUMLGuiSysMLPackageDiagram interface. The diagram is a rectangle with a light blue header containing the text «interface» and IUMLGuiSysMLPackageDiagram. Below the header are two empty lines for attributes or methods.

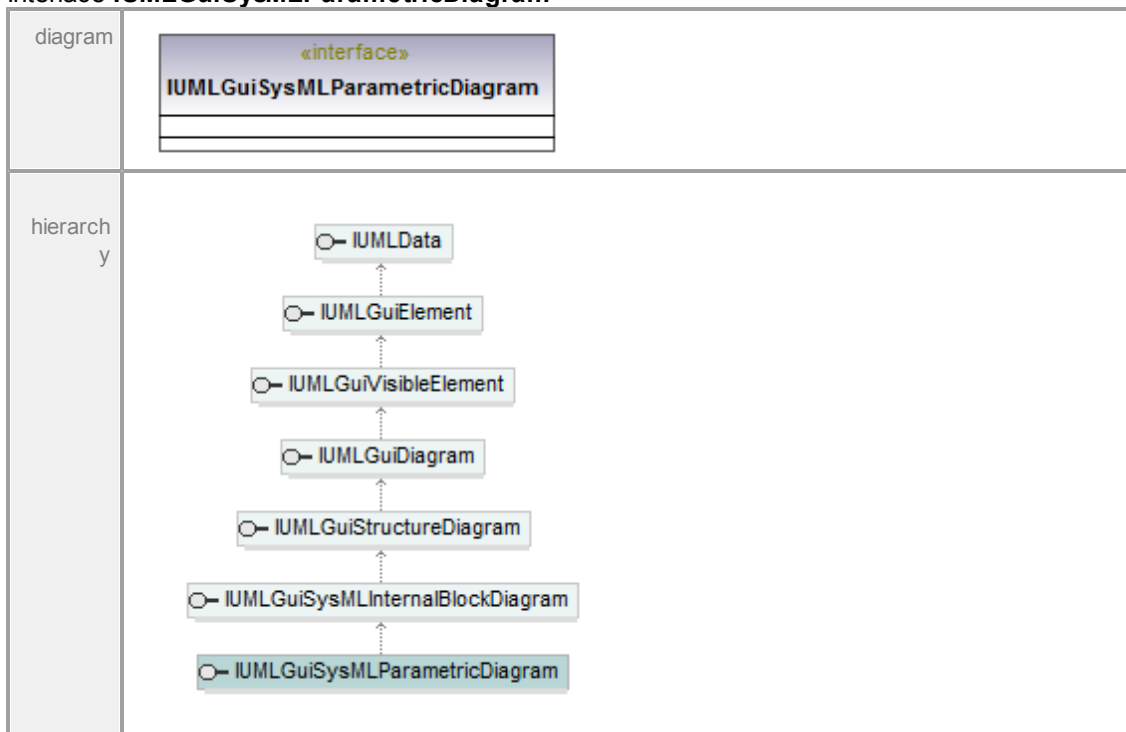


UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiSysMLParametricDiagram

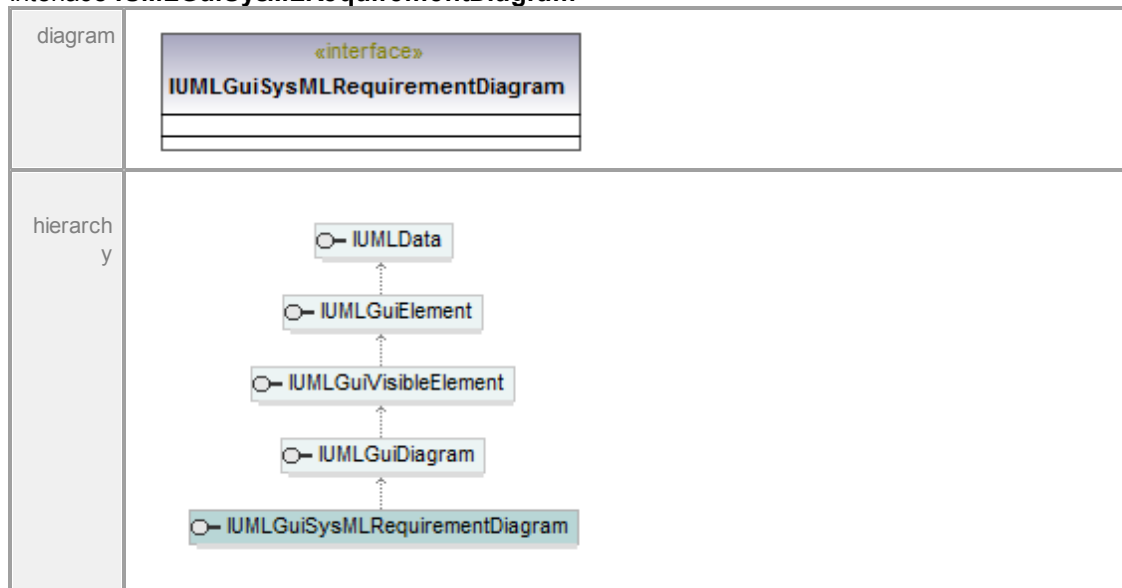
Interface IUMLGuiSysMLParametricDiagram



UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

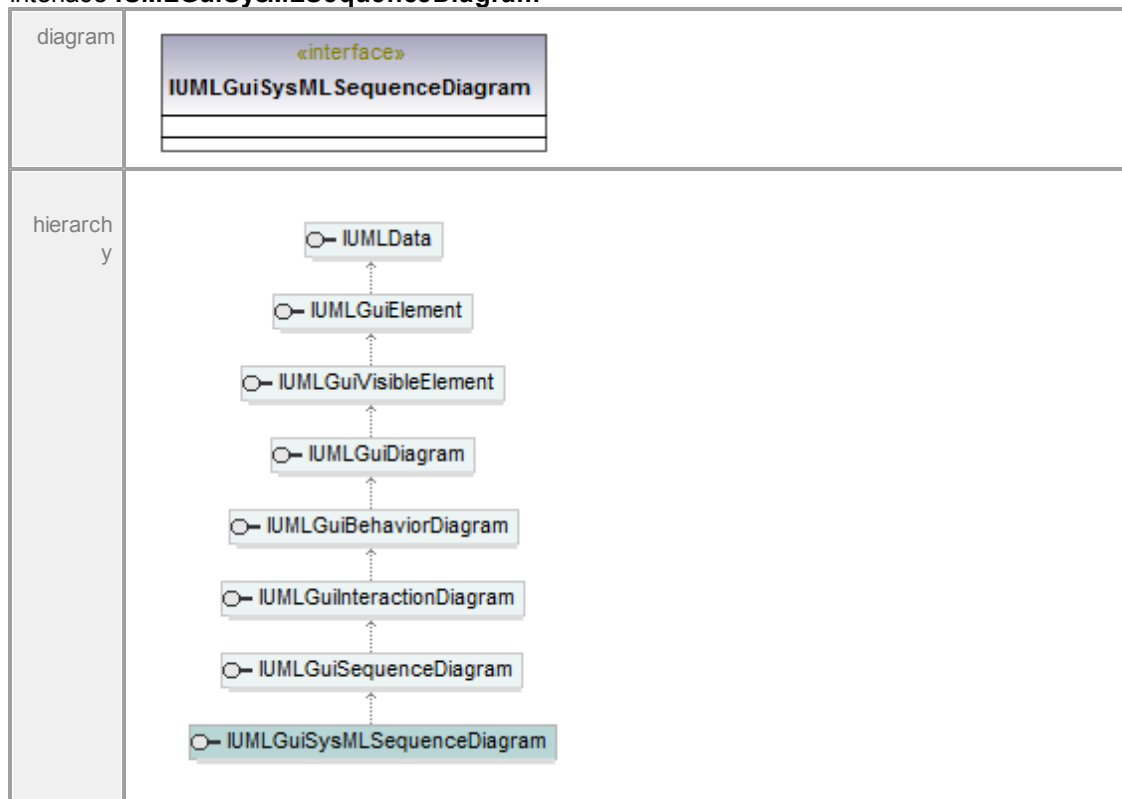
08/03/15 13:42:06

UModelAPI - IUMLGuiSysMLRequirementDiagram

Interface **IUMLGuiSysMLRequirementDiagram**UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

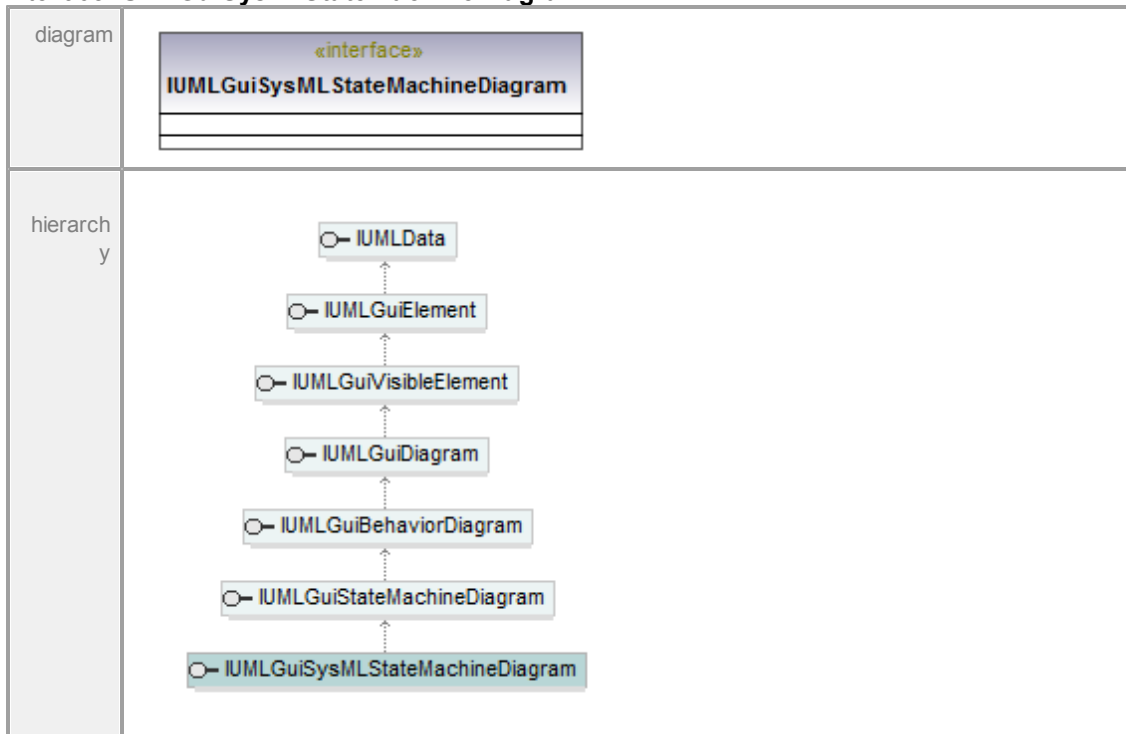
UModelAPI - IUMLGuiSysMLSequenceDiagram

Interface **IUMLGuiSysMLSequenceDiagram**

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

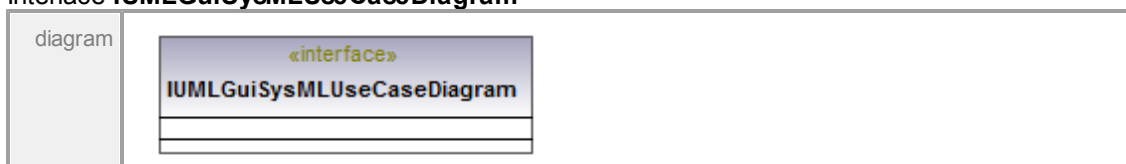
08/03/15 13:42:06

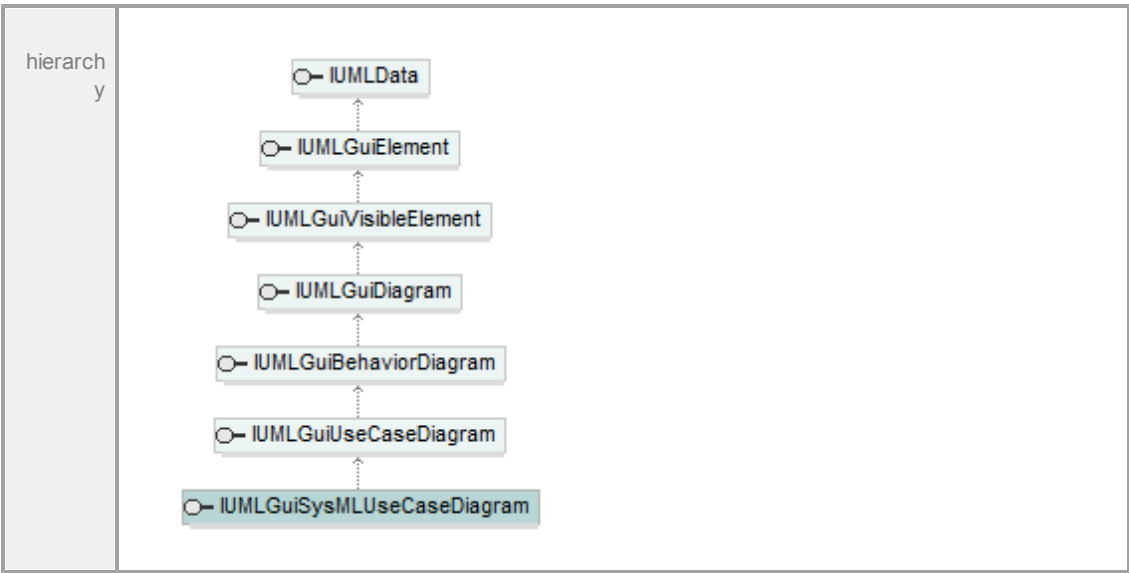
UModelAPI - IUMLGuiSysMLStateMachineDiagram

Interface **IUMLGuiSysMLStateMachineDiagram**UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiSysMLUseCaseDiagram

Interface **IUMLGuiSysMLUseCaseDiagram**



UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiTextHyperlink

Interface IUMLGuiTextHyperlink

diagram	«interface» IUMLGuiTextHyperlink ◆ SetHyperlinkGuiElementAddress(in ipLinkedGuiElement:IUMLGuiVisibleElement, in ipLinkedGuiElementCell:IUMLNamedElement):void ◆ SetHyperlinkModelElementAddress(in ipLinkedData:IUMLData):void ◆ SetHyperlinkFileAddress(in strFilePathOrUrl:string):void ◆ OpenLink():void ◆ «GetAccessor, property» NoteTextStartPos():int ◆ «GetAccessor, property» NoteTextEndPos():int ◆ «GetAccessor, property» LinkAddress():string	
hierarchy	<pre>graph TD; IUMLData < > IUMLGuiElement; IUMLGuiElement < > IUMLGuiVisibleElement; IUMLGuiVisibleElement < > IUMLGuiTextHyperlink;</pre>	
typedElements	Interface IUMLDataAll Interface IUMLGuiDiagram Interface IUMLGuiNote	Operation InsertOwnedGuiTextHyperlinkAt Operation InsertOwnedGuiTextHyperlinkAt Operation InsertOwnedGuiTextHyperlinkAt
documentation	Text Hyperlinks store an URL together with a begin and an end number referencing positions in some text. Any text between a such a begin and end position is displayed as hyperlink and triggers UModel	

to open the URL when clicked. This is used in [IUMLGuiNote](#) to create hyperlinks inside of the text comment for example.

Operation **IUMLGuiTextHyperlink::LinkAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLGuiTextHyperlink::NoteTextEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLGuiTextHyperlink::NoteTextStartPos**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLGuiTextHyperlink::OpenLink**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	void			

Operation **IUMLGuiTextHyperlink::SetHyperlinkFileAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	strFilePathOrUrl		string			
	return	return	void			

Operation **IUMLGuiTextHyperlink::SetHyperlinkGuiElementAddress**

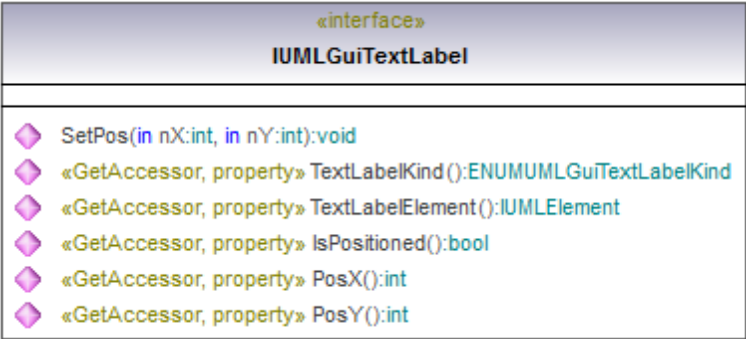
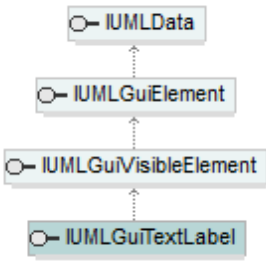
parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedGuiElement		IUMLGuiVisibleElement			
	ipLinkedGuiElementCell		IUMLNamedElement			
	return	return	void			

Operation **IUMLGuiTextHyperlink::SetHyperlinkModelElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
	ipLinkedData	in	IUMLData			
	return	return	void			

UModelAPI - IUMLGuiTextLabel

Interface **IUMLGuiTextLabel**

diagram		
hierarch y		
typedE lements	Interface IUMLDataAll Interface IUMLGuiTextLabelWaypoint	Operation GetTextLabelText IsTextLabelVisible SetTextLabelVisible Operation GetTextLabelText IsTextLabelVisible SetTextLabelVisible
documen tation	A text label is an graphical object displaying additional data at the begin, end or at the center of a line. The IUMLGuiTextLabel interface provides access to this element. The text label can reference an UML Element using the TextLabelElement property and has a TextLabelKind which affects what text is shown in the text label.	

Operation **IUMLGuiTextLabel::IsPositioned**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	bool			

Operation **IUMLGuiTextLabel::PosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLGuiTextLabel::PosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLGuiTextLabel::SetPos**

parameter	name	direction	type	type modifier	multiplicity	default

r	nX nY return	in in return	int int void

Operation **IUMLGuiTextLabel::TextLabelElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLElement			

Operation **IUMLGuiTextLabel::TextLabelKind**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLGuiTextLabelKind			

UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiTextLabelWaypoint

Interface **IUMLGuiTextLabelWaypoint**

Operation **IUMLGuiTextLabelWaypoint::GetTextLabelText**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	in	IUMLGuiTextLabel			
	return	return	string			

Operation **IUMLGuiTextLabelWaypoint::IsTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	in	IUMLGuiTextLabel			
	return	return	bool			

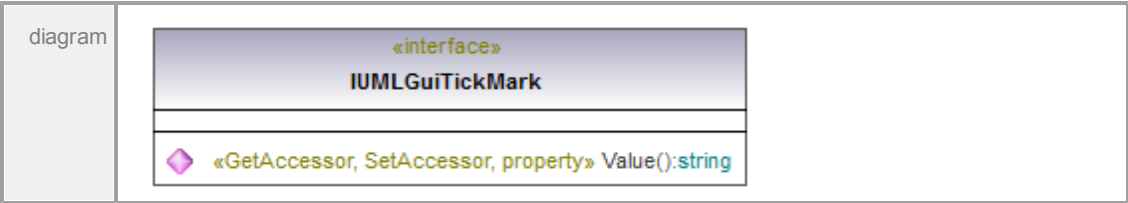
Operation **IUMLGuiTextLabelWaypoint::SetTextLabelVisible**

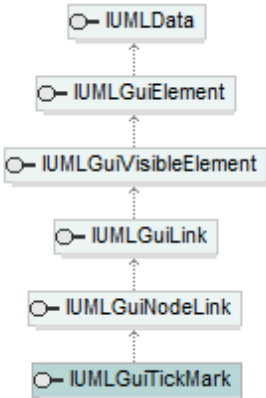
parameter	name	direction	type	type modifier	multiplicity	default
	ipTextLabel	in	IUMLGuiTextLabel			
	bVisible	in	bool			
	return	return	void			

Operation **IUMLGuiTextLabelWaypoint::TextLabels**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			
documentation	Returns a list of all text labels of this waypoint. Contains only elements of type (or subtype of) IUMLGuiTextLabel .					

Interface **IUMLGuiTickMark**



hierarchy	
typedElements	Interface IUMLGuiTimingDiagram Operation AddUMLGuiTickMark
documentation	A tick mark is a special graphical item appearing on the border of lifelines on timing diagrams. It represents a certain point in time and is displayed as short vertical line. It has a Value property which is displayed as text below the vertical line.

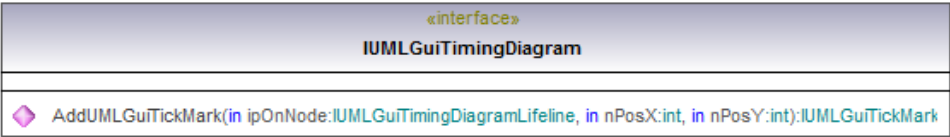
Operation **IUMLGuiTickMark::Value**

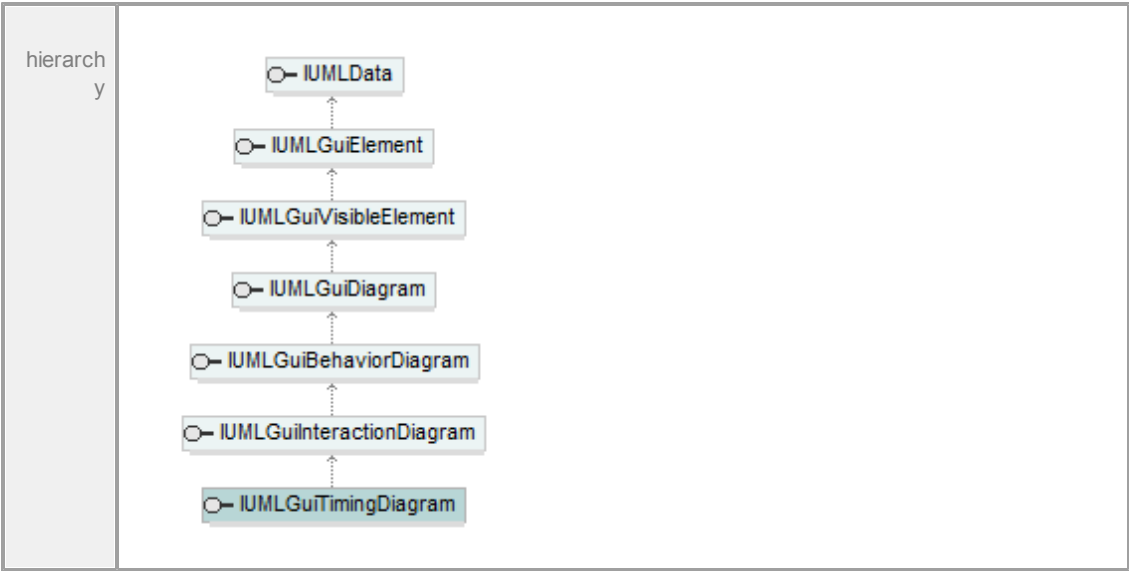
parameter	name return	direction return	type string	type modifier	multiplicity	default

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLGuiTimingDiagram

Interface **IUMLGuiTimingDiagram**

diagram	



Operation **IUMLGuiTimingDiagram::AddUMLGuiTickMark**

parameter	name	direction	type	type modifier	multiplicity	default
	ipOnNode	in	IUMLGuiTimingDiagramLifeLine			
	nPosX	in	int			
	nPosY	in	int			
	return	return	IUMLGuiTickMark			

UModelAPI - IUMLGuiTimingDiagramLifeline

Interface **IUMLGuiTimingDiagramLifeline**

diagram	«interface» IUMLGuiTimingDiagramLifeline ◆ GetTimeTickLength(in nIdx:int):int ◆ SetTimeTickLength(in nIdx:int, in nNewVal:int):void ◆ GetStateIndex(in nTimeTickIndex:int):int ◆ SetStateIndex(in nTimeTickIndex:int, in nNewVal:int):void ◆ SetStateIndexErased(in nTimeTickIndex:int):void ◆ GetVisualStatePosition(in nStateIndex:int):int ◆ SetVisualStatePosition(in nStateIndex:int, in nNewVal:int):void ◆ «GetAccessor, SetAccessor, property» NameCompartmentEndPos():int ◆ «GetAccessor, SetAccessor, property» GeneralValueLifelineNameCompartmentEndPos():int ◆ «GetAccessor, SetAccessor, property» StateCompartmentEndPos():int ◆ «GetAccessor, SetAccessor, property» IsShowAsGeneralValueLifeline():bool ◆ «GetAccessor, SetAccessor, property» TimeTickLengthCount():int ◆ «GetAccessor, SetAccessor, property» VisualStatePositionCount():int	
hierarc hy	<pre> graph BT IUMLData -- > IUMLGuiElement IUMLGuiElement -- > IUMLGuiVisibleElement IUMLGuiVisibleElement -- > IUMLGuiLink IUMLGuiLink -- > IUMLGuiNodeLink IUMLGuiNodeLink -- > IUMLGuiTimingDiagramLifeline </pre>	
typedE lements	Interface IUMLGuiTimingDiagram	Operation AddUMLGuiTickMark
docume ntation	A IUMLGuiTimingDiagramLifeline is the graphical representation of a lifeline on a timing diagram. This type of lifeline has several options to display its data and provides access to these through its numerous properties.	

Operation

IUMLGuiTimingDiagramLifeline::GeneralValueLifelineNameCompartmentEndPos

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLGuiTimingDiagramLifeline::GetStateIndex**

paramete	name	direction	type	type modifier	multiplicity	default

r	nTimeTickIndex	int				
	return	return	int			

Operation **IUMLGuiTimingDiagramLifetime::GetTimeTickLength**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	int			

Operation **IUMLGuiTimingDiagramLifetime::GetVisualStatePosition**

parameter	name	direction	type	type modifier	multiplicity	default
r	nStateIndex	in	int			
	return	return	int			

Operation **IUMLGuiTimingDiagramLifetime::IsShowAsGeneralValueLifetime**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLGuiTimingDiagramLifetime::NameCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLGuiTimingDiagramLifetime::SetStateIndex**

parameter	name	direction	type	type modifier	multiplicity	default
r	nTimeTickIndex		int			
	x					
	nNewVal	in	int			
	return	return	void			

Operation **IUMLGuiTimingDiagramLifetime::SetStateIndexErased**

parameter	name	direction	type	type modifier	multiplicity	default
r	nTimeTickIndex		int			
	x					
	return	return	void			

Operation **IUMLGuiTimingDiagramLifetime::SetTimeTickLength**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	nNewVal	in	int			
	return	return	void			

Operation **IUMLGuiTimingDiagramLifetime::SetVisualStatePosition**

parameter	name	direction	type	type modifier	multiplicity	default
r	nStateIndex	in	int			
	nNewVal	in	int			
	return	return	void			

Operation **IUMLGuiTimingDiagramLifetime::StateCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLGuiTimingDiagramLifeline::TimeTickLengthCount**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLGuiTimingDiagramLifeline::VisualStatePositionCount**

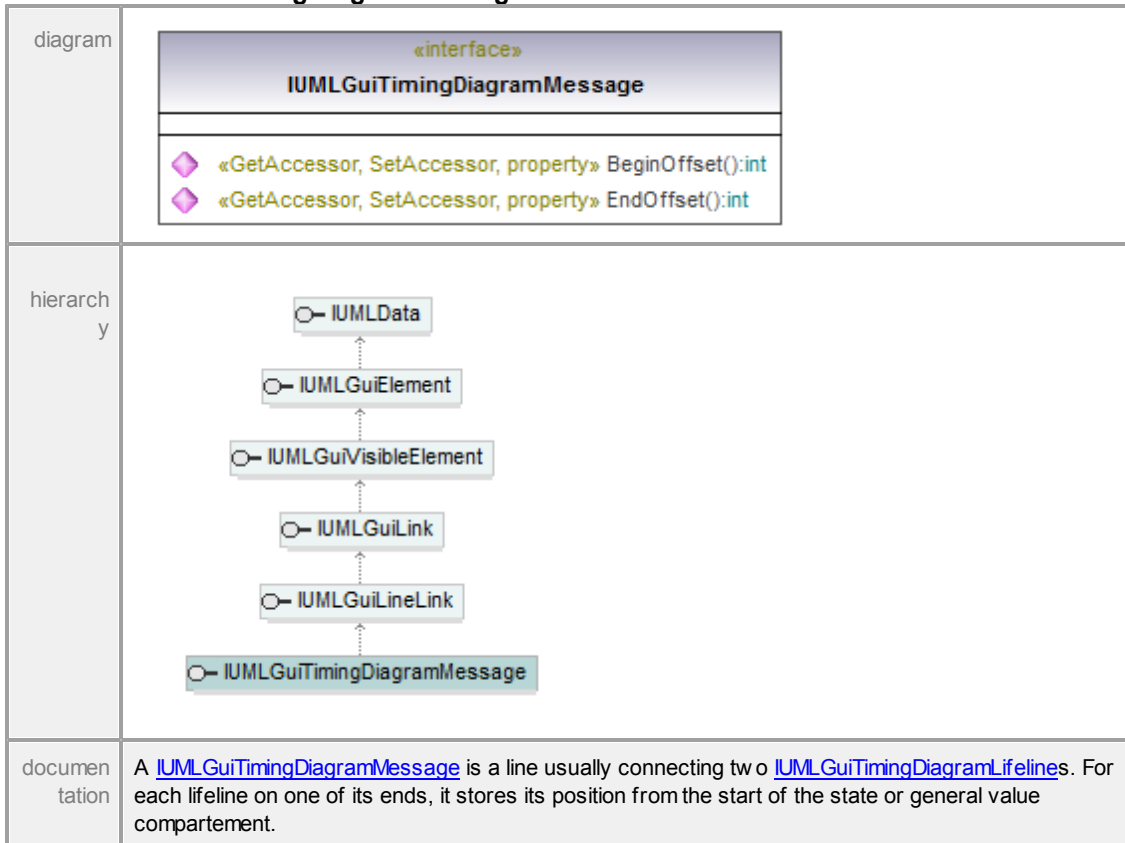
parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLGuiTimingDiagramMessage

Interface **IUMLGuiTimingDiagramMessage**



Operation **IUMLGuiTimingDiagramMessage::BeginOffset**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLGuiTimingDiagramMessage::EndOffset**

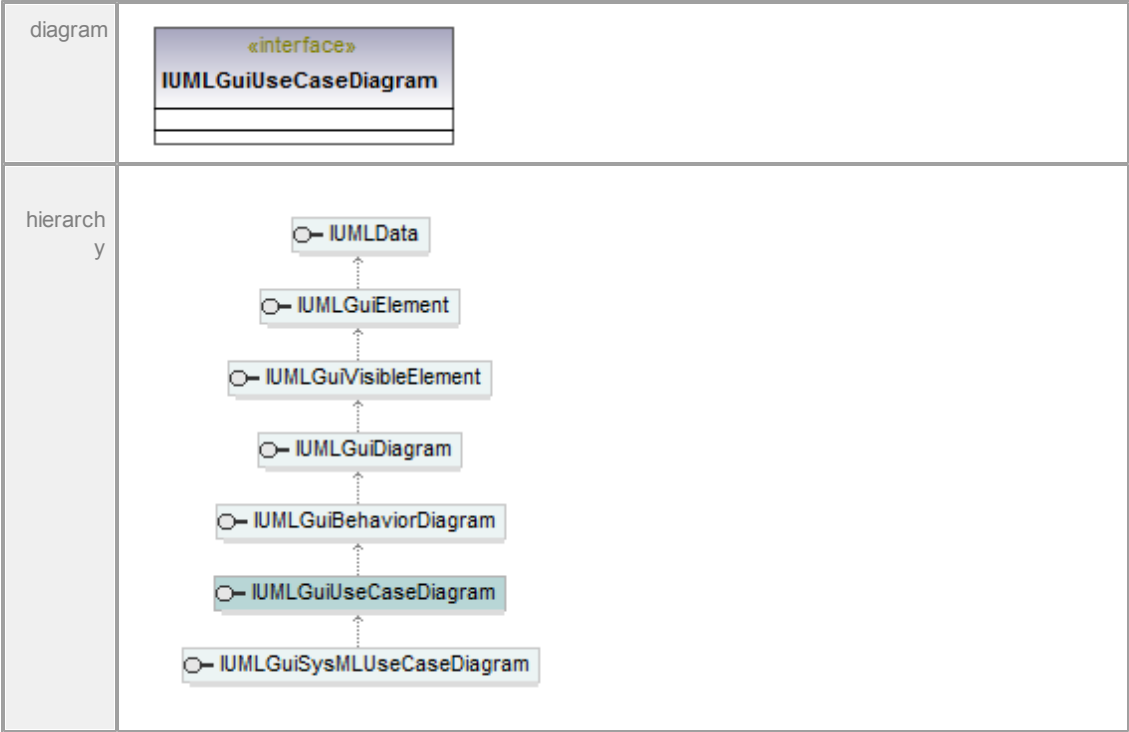
parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	int

UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLGuiUseCaseDiagram

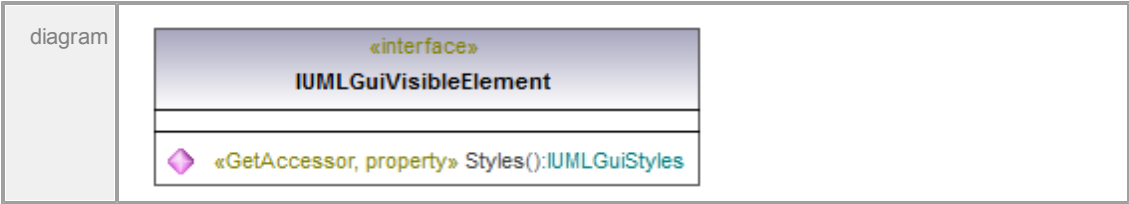
Interface IUMLGuiUseCaseDiagram



UML documentation generated by [UModel](http://www.altova.com/umodel) UML Editor <http://www.altova.com/umodel> 08/03/15 13:42:06

UModelAPI - IUMLGuiVisibleElement

Interface IUMLGuiVisibleElement



hierarchy	<pre> graph TD IUMData -- > IUMGuiElement IUMGuiElement -- > IUMGuiVisibleElement IUMGuiVisibleElement -- > IUMGuiDiagram IUMGuiVisibleElement -- > IUMGuiLink IUMGuiVisibleElement -- > IUMGuiTextHyperlink IUMGuiVisibleElement -- > IUMGuiTextLabel </pre>	
typedElements	Interface IUMCommentTextHyperlink Interface IUMDataAll Interface IUMGuiTextHyperlink Interface IUMHyperlink2GuiElement Interface IUMNamedElement	Operation SetHyperlinkGuiElementAddress Operation InsertOwnedHyperlink2GuiElementAt LinkedGuiElement Operation SetHyperlinkGuiElementAddress Operation SetHyperlinkGuiElementAddress Operation LinkedGuiElement Operation InsertOwnedHyperlink2GuiElementAt
documentation	The IUMGuiVisibleElement is the base interface for most visible elements which can be placed on UModel diagrams. Visible elements have a style with which it is possible to influence its color and/or shape, depending on the actual type of the visible element.	

Operation [IUMGuiVisibleElement::Styles](#)

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMGuiStyles			

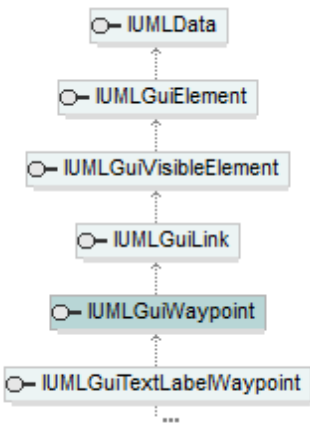
UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMGuiWaypoint

Interface [IUMGuiWaypoint](#)

diagram	<pre> classDiagram class IUMGuiWaypoint { <<interface>> SetPos(in x:int, in y:int):void «GetAccessor, property» PosX():int «GetAccessor, property» PosY():int «GetAccessor, property» LineLinks():IUMDataList } </pre>

hierarchy	 <pre>graph TD IUMLData < .. IUMLGuiElement IUMLGuiElement < .. IUMLGuiVisibleElement IUMLGuiVisibleElement < .. IUMLGuiLink IUMLGuiLink < .. IUMLGuiWaypoint IUMLGuiWaypoint < .. IUMLGuiTextLabelWaypoint IUMLGuiTextLabelWaypoint <</pre>	
typedElements	Interface IUMLDataAll Interface IUMLGuiLineLink	Operation InsertWaypointAt Operation InsertWaypointAt
documentation	A IUMLGuiWaypoint is a part of a IUMLGuiLineLink which defines the position of one point of a line. There are several subtypes of this interface which for example make it possible to attach text labels or lines to a waypoint or line.	

Operation **IUMLGuiWaypoint::LineLinks**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

Operation **IUMLGuiWaypoint::PosX**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

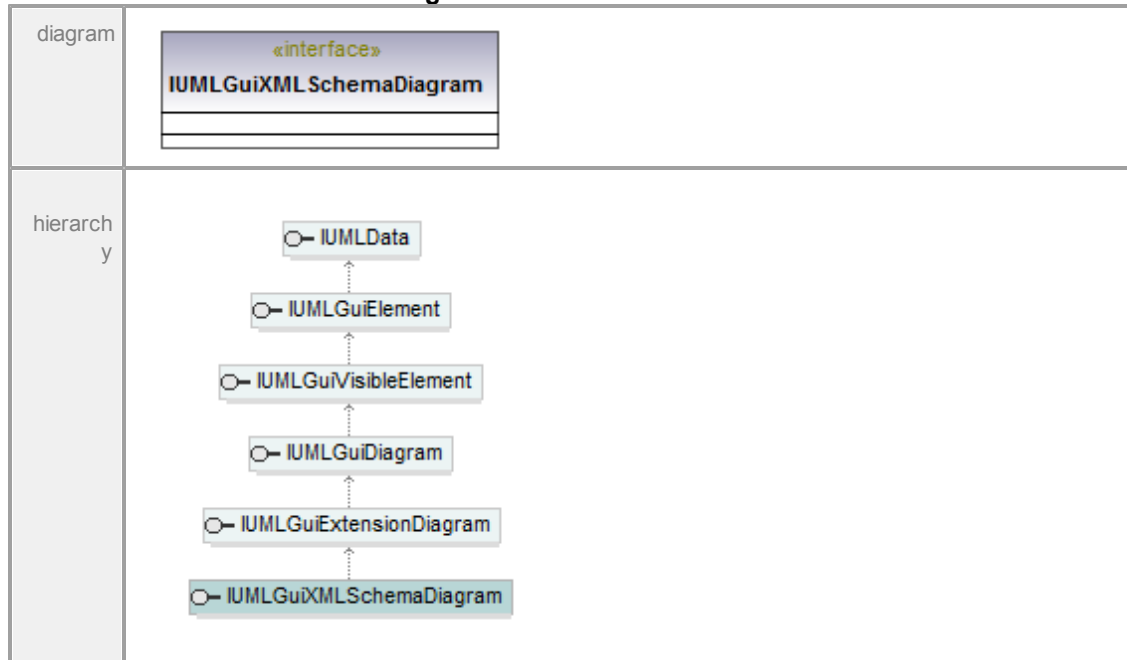
Operation **IUMLGuiWaypoint::PosY**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLGuiWaypoint::SetPos**

parameter	name	direction	type	type modifier	multiplicity	default
	x	in	int			
	y	in	int			
	return	return	void			

UModelAPI - IUMLGuiXMLSchemaDiagram

Interface **IUMLGuiXMLSchemaDiagram**UML documentation generated by **UModel** UML Editor <http://www.altova.com/umodel>

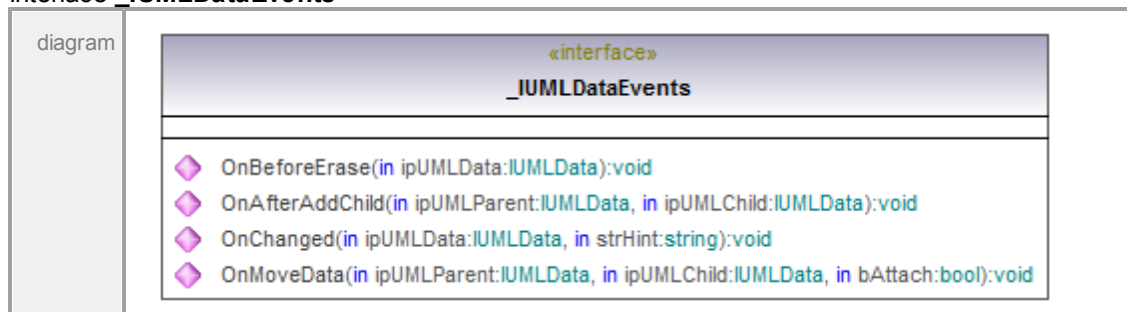
08/03/15 13:42:06

Events

This is a list of all events sent by the UModel API on UMLData level.

Also see [UMLData events and eventfilters](#).

UModelAPI - _IUMLDataEvents

Interface **_IUMLDataEvents**Operation **_IUMLDataEvents::OnAfterAddChild**

parameter	name	direction	type	type modifier	multiplicity	default
	ipUMLParent	in	IUMLData			
	ipUMLChild	in	IUMLData			
	return	return	void			

Operation **_IUMLDataEvents::OnBeforeErase**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipUMLData	in	IUMMLData			
	return	return	void			

Operation **IUMMLDataEvents::OnChanged**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipUMLData	in	IUMMLData			
	strHint	in	string			
	return	return	void			
documentation	strHint is for future use only!					

Operation **IUMMLDataEvents::OnMoveData**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipUMLParent	in	IUMMLData			
	ipUMLChild	in	IUMMLData			
	bAttach	in	bool			
	return	return	void			

Enumerations

This is a list of all enumerations used by the UModel API on UMLData level. If your scripting environment does not support enumerations use the number-values instead.

.

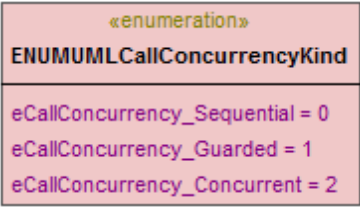
UModelAPI - ENUMUMLAggregationKind

Enumeration **ENUMUMLAggregationKind**

diagram	«enumeration» ENUMUMLAggregationKind eAggregation_None = 0 eAggregation_Shared = 1 eAggregation_Composite = 2				
typedElements	Interface IUMMLDataAll		Operation Aggregation		
	Interface IUMMLProperty		Operation Aggregation		

UModelAPI - ENUMUMLConcurrencyKind

Enumeration **ENUMUMLConcurrencyKind**

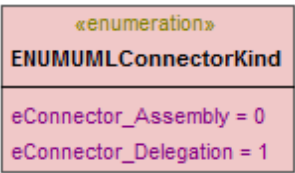
diagram		
typedElements	Interface IUMLBehavioralFeature Interface IUMLDataAll	Operation Concurrency Operation Concurrency

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMUMLConnectorKind

Enumeration **ENUMUMLConnectorKind**

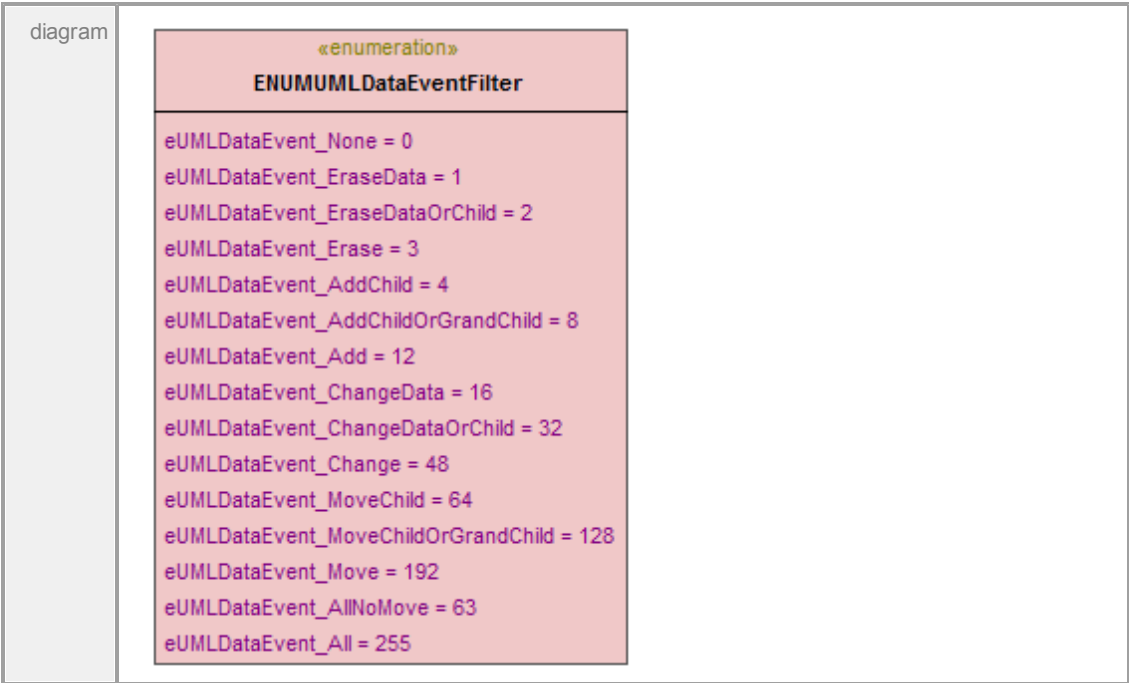
diagram		
typedElements	Interface IUMLConnector Interface IUMLDataAll	Operation ConnectorKind Operation ConnectorKind

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMUMLDataEventFilter

Enumeration **ENUMUMLDataEventFilter**

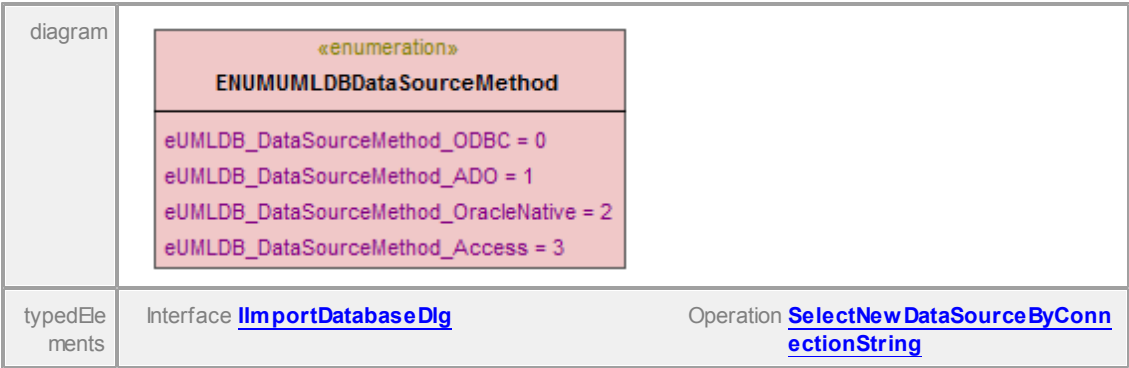


UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMUMLDBDataSourceMethod

Enumeration **ENUMUMLDBDataSourceMethod**

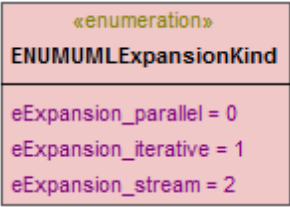


UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMUMLExpansionKind

Enumeration **ENUMUMLExpansionKind**

diagram		
typedElements	Interface IUMLDataAll Interface IUMLExpansionRegion	Operation Mode Operation Mode

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMUMLGuiStyleKind

Enumeration **ENUMUMLGuiStyleKind**

diagram	<i>[The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (http://manual.altova.com/UModel/umodelenterprise/).]</i>	
typedElements	Interface IUMLGuiStyle Interface IUMLGuiStyles	Operation Kind Operation GetName GetStyle GetUsedValue GetValue SetValue

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMUMLGuiTextLabelKind

Enumeration **ENUMUMLGuiTextLabelKind**

diagram	<pre> «enumeration» ENUMUMLGuiTextLabelKind eTextLabel_Element_Stereotype = 0 eTextLabel_Association_Name = 1 eTextLabel_Property_Name = 2 eTextLabel_Property_Multiplicity = 3 eTextLabel_Property_Constraint = 4 eTextLabel_Link_Name = 5 eTextLabel_LinkBegin_PropertyName = 6 eTextLabel_LinkEnd_PropertyName = 7 eTextLabel_Dependency = 8 eTextLabel_Usage = 9 eTextLabel_Manifestation = 10 eTextLabel_Deployment = 11 eTextLabel_Include = 12 eTextLabel_Extend = 13 eTextLabel_ProfileApplication = 14 eTextLabel_MessageString = 15 eTextLabel_Element_Constraint = 16 eTextLabel_ActivityEdge_Name = 17 eTextLabel_ActivityEdge_Guard = 18 eTextLabel_ActivityEdge_Weight = 19 eTextLabel_ObjectFlow_MultiCast = 20 eTextLabel_ObjectFlow_MultiReceive = 21 eTextLabel_ExceptionHandler_ExceptionType = 22 eTextLabel_Transition_Expression = 23 eTextLabel_DependencyRoleBinding_RoleName = 24 eTextLabel_Connector_Name = 25 eTextLabel_PackageMerge = 26 eTextLabel_PackageImport = 27 eTextLabel_ElementImport = 28 eTextLabel_BPMNConditionExpression = 29 eTextLabel_Abstraction = 30 eTextLabel_MemberEnd_Stereotype = 31 eTextLabel_ObjectFlow_DecisionInput = 32 eTextLabel_InformationFlow = 33 eTextLabel_DotNetPropertyName = 34 </pre>	
typedElements	Interface IUMLDataAll Interface IUMLGuiTextLabel	Operation TextLabelKind Operation TextLabelKind

UModelAPI - ENUMUMLInteractionOperatorKind

Enumeration **ENUMUMLInteractionOperatorKind**

diagram	<pre> «enumeration» ENUMUMLInteractionOperatorKind eInteractionOperator_Seq = 0 eInteractionOperator_Alt = 1 eInteractionOperator_Opt = 2 eInteractionOperator_Break = 3 eInteractionOperator_Par = 4 eInteractionOperator_Strict = 5 eInteractionOperator_Loop = 6 eInteractionOperator_Critical = 7 eInteractionOperator_Neg = 8 eInteractionOperator_Assert = 9 eInteractionOperator_Ignore = 10 eInteractionOperator_Consider = 11 </pre>	
typedElements	Interface IUMLCombinedFragment Interface IUMLDataAll	Operation InteractionOperator Operation InteractionOperator

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMUMLMessageKind

Enumeration **ENUMUMLMessageKind**

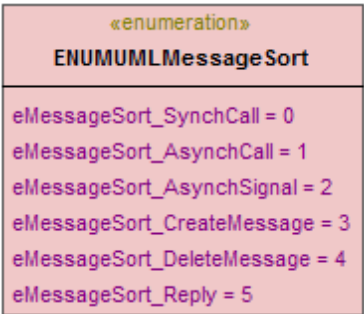
diagram	<pre> «enumeration» ENUMUMLMessageKind eMessage_Complete = 0 eMessage_Lost = 1 eMessage_Found = 2 eMessage_Unknown = 3 </pre>	
typedElements	Interface IUMLDataAll Interface IUMLMessage	Operation MessageKind Operation MessageKind

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMUMLMessageSort

Enumeration **ENUMUMLMessageSort**

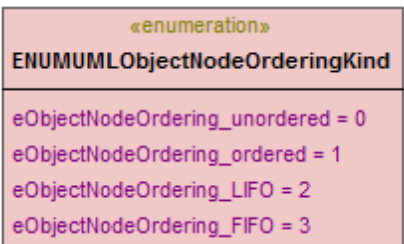
diagram		
typedElements	Interface IUMLDataAll Interface IUMLMessage	Operation MessageSort Operation MessageSort

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMUMLObjectNodeOrderingKind

Enumeration **ENUMUMLObjectNodeOrderingKind**

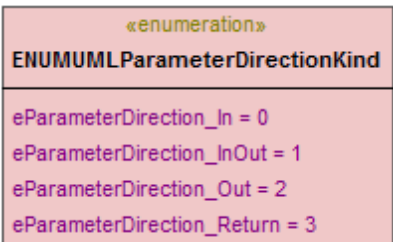
diagram		
typedElements	Interface IUMLDataAll Interface IUMLObjectNode	Operation Ordering Operation Ordering

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMUMLParameterDirectionKind

Enumeration **ENUMUMLParameterDirectionKind**

diagram		
typedElements	Interface IUMLDataAll Interface IUMLParameter	Operation Direction Operation Direction

UModelAPI - ENUMUMLPredefinedElement

Enumeration **ENUMUMLPredefinedElement**

diagram	<i>[The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (http://manual.altova.com/UModel/umodelenterprise/).]</i>		
typedElements	Interface IUMIDataAll	Operation	ApplyPredefinedStereotype FindPredefinedOwnedElement GetStereotypeApplicationForPredefinedStereotype IsPredefinedStereotypeApplied SetPredefinedTaggedValueAt
	Interface IUMLElement	Operation	UnapplyPredefinedStereotype ApplyPredefinedStereotype FindPredefinedOwnedElement GetStereotypeApplicationForPredefinedStereotype IsPredefinedStereotypeApplied UnapplyPredefinedStereotype
	Interface IUMLStereotypeApplication	Operation	SetPredefinedTaggedValueAt
documentation	Deprecated: ePredefined_Java_finalStereotypeOfProperty ePredefined_Java_finalStereotypeOfOperation ePredefined_Java_finalStereotypeOfClass		

UModelAPI - ENUMUMLPseudostateKind

Enumeration **ENUMUMLPseudostateKind**

diagram	«enumeration» ENUMUMLPseudostateKind ePseudostate_Initial = 0 ePseudostate_DeepHistory = 1 ePseudostate_ShallowHistory = 2 ePseudostate_Join = 3 ePseudostate_Fork = 4 ePseudostate_Junction = 5 ePseudostate_Choice = 6 ePseudostate_EntryPoint = 7 ePseudostate_ExitPoint = 8 ePseudostate_Terminate = 9	
typedElements	Interface IUMLDDataAll Interface IUMLPseudostate	Operation PseudostateKind Operation PseudostateKind

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - ENUMUMLTransitionKind

Enumeration **ENUMUMLTransitionKind**

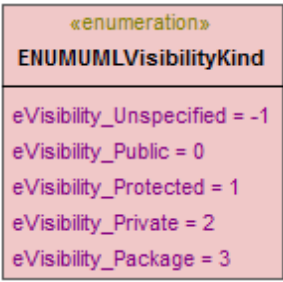
diagram	«enumeration» ENUMUMLTransitionKind eTransition_Internal = 0 eTransition_Local = 1 eTransition_External = 2	
typedElements	Interface IUMLDDataAll Interface IUMLTransition	Operation TransitionKind Operation TransitionKind

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

09/04/12 14:02:42

UModelAPI - ENUMUMLVisibilityKind

Enumeration **ENUMUMLVisibilityKind**

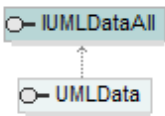
diagram		
typedElements	Interface ILocalOptionsEditing Interface IUMLDataAll Interface IUMLElementImport Interface IUMLNamedElement Interface IUMLPackageImport	Operation OperationsDefaultVisibility Operation PropertiesDefaultVisibility Operation Visibility Operation Visibility Operation Visibility Operation Visibility

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - IUMLDataAll

Interface **IUMLDataAll**

diagram	[The diagram is not included because of page size constraints; however, it is available in the HTML version of the manual (http://manual.altova.com/UModel/umodelenterprise/).]	
hierarchy		

Operation **IUMLDataAll::Abstraction**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLComponent			

Operation **IUMLDataAll::ActionContext**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLClassifier			

Operation **IUMLDataAll::ActionInputPin**

parameter	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLActionInputPin			

Operation **IUMLDataAll::ActionTriggers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ActionValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLValueSpecification			

Operation **IUMLDataAll::ActiveLayer**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiDiagramLayer			

Operation **IUMLDataAll::Activity**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLActivity			

Operation **IUMLDataAll::ActivityEdges**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ActivityGroups**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ActivityPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Actual**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLParameterableElement			

Operation **IUMLDataAll::ActualGates**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Addition**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLUseCase			

Operation **IUMLDataAll::AddOwnedGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipForUMLData	in	IUMLGuiNodeLink			
	return	return	void			

Operation **IUMLDataAll::AddUMLElement**

parameter	name	direction	type	type modifier	multiplicity	default
	strKind	in	string			
	nLeft	in	int			
	nTop	in	int			
	return	return	IUMLGuiNodeLink			

Operation **IUMLDataAll::AddUMLGuiContainmentLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromLink	in	IUMLGuiLink			
	ipToLink	in	IUMLGuiLink			
	return	return	IUMLGuiContainmentLink			

Operation **IUMLDataAll::AddUMLGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipForUMLData	in	IUMLData			
	nLeft	in	int			
	nTop	in	int			
	return	return	IUMLGuiNodeLink			

Operation **IUMLDataAll::AddUMLGuiNote**

parameter	name	direction	type	type modifier	multiplicity	default
	nLeft	in	int			
	nTop	in	int			
	return	return	IUMLGuiNote			

Operation **IUMLDataAll::AddUMLGuiNoteLink**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	in	IUMLGuiNote			
	ipToLink	in	IUMLGuiNodeLink			
	return	return	IUMLGuiNoteLink			

Operation **IUMLDataAll::AddUMLGuiNoteLinkToLine**

parameter	name	direction	type	type modifier	multiplicity	default
	ipFromNote	in	IUMLGuiNote			
	ipToLink	in	IUMLGuiLineLink			
	nDistanceFromLineBegin	in	int			
	return	return	IUMLGuiNoteLink			

Operation **IUMLDataAll::AddUMLLineElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	ipFromNode	in	IUMLGuiNodeLink			
	ipToNode	in	IUMLGuiNodeLink			
	return	return	IUMLGuiLineLink			

Operation **IUMLDataAll::Aggregation**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLAggregationKind			

Operation **IUMLDataAll::Alias**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::AllApplicableStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::AllowSubstitutable**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::AllWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::AnnotatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Application**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **IUMLDataAll::AppliedElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLElement			

Operation **IUMLDataAll::AppliedProfile**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLProfile			

Operation **IUMLDataAll::AppliedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ApplyingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLPackage			

Operation **IUMLDataAll::ApplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
r	nStereotype	in	ENUMUMLPredefinedElement			
	return	return	IUMLStereotypeApplication			

Operation **IUMLDataAll::ApplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipStereotype	in	IUMLStereotype			
	return	return	IUMLStereotypeApplication			

Operation **IUMLDataAll::Arguments**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Association**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLAssociation			

Operation **IUMLDataAll::AssociationEnd**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLProperty			

Operation **IUMLDataAll::AttachedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::AttachedTo**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiLink			

Operation **IUMLDataAll::Attributes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::BaseClass**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::BeginOffset**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::Behavior**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLDataAll::BehaviorExecution**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLDataAll::BehaviorSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehaviora IFeature			

Operation **IUMLDataAll::Body**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::BooleanValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::Bottom**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::BoundElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTemplate ableElement			

Operation **IUMLDataAll::CallOperation**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLOperation			

Operation **IUMLDataAll::CallTarget**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	IUMLInputPin			

Operation **IUMLDataAll::ChangeExpression**

paramete r	name return	direction return	type IUMLValueSpec ification	type modifier	multiplicity	default

Operation **IUMLDataAll::Class**

paramete r	name return	direction return	type IUMLClass	type modifier	multiplicity	default

Operation **IUMLDataAll::Classifier**

paramete r	name return	direction return	type IUMLClassifier	type modifier	multiplicity	default

Operation **IUMLDataAll::ClientDependencies**

paramete r	name return	direction return	type IUMLDataList	type modifier	multiplicity	default

Operation **IUMLDataAll::Clients**

paramete r	name return	direction return	type IUMLDataList	type modifier	multiplicity	default

Operation **IUMLDataAll::CodeFileNameCount**

paramete r	name return	direction return	type int	type modifier	multiplicity	default

Operation **IUMLDataAll::CodeLang**

paramete r	name return	direction return	type ENUMCodeLang	type modifier	multiplicity	default

Operation **IUMLDataAll::CodeLangVersion**

paramete r	name return	direction return	type ENUMCodeLangVersion	type modifier	multiplicity	default

Operation **IUMLDataAll::CodeOperation**

paramete r	name return	direction return	type IUMLOperation	type modifier	multiplicity	default

Operation **IUMLDataAll::CodeProjectFileOrDirectory**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **IUMLDataAll::CollaborationRoles**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::CollaborationType**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLCollaboration			

Operation **IUMLDataAll::CollaborationUses**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Comment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::Concurrency**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLConcurrencyKind			

Operation **IUMLDataAll::ConnectionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Connections**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ConnectorKind**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLConnectorKind			

Operation **IUMLDataAll::ConnectorType**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLAssociation			

Operation **IUMLDataAll::ConstrainedElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ConstrainingClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ConstrainingPointX**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::ConstrainingPointY**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::ContainedEdges**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ContainedNodes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Container**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLRegion			

Operation **IUMLDataAll::Context**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLNamespace			

Operation **IUMLDataAll::Contract**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLInterface			

Operation **IUMLDataAll::ConstrainingAreaIndex**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::Conveyed**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Covered**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLLifetime			

Operation **IUMLDataAll::Datatype**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataType			

Operation **IUMLDataAll::DecisionInput**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLDataAll::Default**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::DefaultLinkName**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::DefaultParamValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::DefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLValueSpecification			

Operation **IUMLDataAll::DefiningFeature**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLStructuralFeature			

Operation **IUMLDataAll::DeployedElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Deployments**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Direction**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLParameterDirectionKind			

Operation **IUMLDataAll::DistanceFromLineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::DoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLDataAll::Edges**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Effect**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLDataAll::Element**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLElement			

Operation **IUMLDataAll::ElementImports**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::EndOffset**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::EndTypes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Entries**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Entry**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLDataAll::Enumeration**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEnumeration			

Operation **IUMLDataAll::EraseAnnotatedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	idx return	in return	int void			

Operation **IUMLDataAll::EraseCodeFileNameAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseCollaborationRoleAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseConstrainingClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseCoveredByAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseExitAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseFromDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipVal	in	IUMLGuiElement			
	return	return	void			

Operation **IUMLDataAll::EraseFromModel**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	void			

Operation **IUMLDataAll::EraseInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	void			

Operation **IUMLDataAll::EraseNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int void			

Operation **IUMLDataAll::EraseObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int void			

Operation **IUMLDataAll::EraseOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int void			

Operation **IUMLDataAll::EraseRaisedExceptionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int void			

Operation **IUMLDataAll::EraseRealizingConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int void			

Operation **IUMLDataAll::EraseSubjectAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int void			

Operation **IUMLDataAll::EraseWaypointAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int void			

Operation **IUMLDataAll::Event**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEvent			

Operation **IUMLDataAll::EventActionResults**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::EventFilter**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::ExceptionHandlers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ExceptionInput**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLObjectNode			

Operation **IUMLDataAll::ExceptionTypes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ExecutionSpecificationFinish**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLExecutionSpecification			

Operation **IUMLDataAll::ExecutionSpecificationStart**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLExecutionSpecification			

Operation **IUMLDataAll::Exit**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLDataAll::Exits**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Expr**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLValueSpecification			

Operation **IUMLDataAll::Expression**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLExpression			

Operation **IUMLDataAll::ExtendedCase**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLUseCase			

Operation **IUMLDataAll::Extends**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Extension**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLUseCase			

Operation **IUMLDataAll::ExtensionLocations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ExtensionPoints**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Features**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::FeaturingClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::FileName**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::FindOwnedMemberWithQualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default
r	strName	in	string			
	return	return	IUMLNamedElement			

Operation **IUMLDataAll::FindPredefinedOwnedElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	nElement	in	ENUMUMLPredefinedElement			
	bRecursive	in	bool			
	return	return	IUMLData			

Operation **IUMLDataAll::Finish**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLOccurrenceSpecification			

Operation **IUMLDataAll::Formal**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTemplateParameter			

Operation **IUMLDataAll::FormalGates**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Fragments**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::General**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLClassifier			

Operation **IUMLDataAll::Generalizations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Generals**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::GeneralValueLifelineNameCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::GetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	string			

Operation **IUMLDataAll::GetCodeFilePath**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	string			

Operation **IUMLDataAll::GetHSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	int			

Operation **IUMLDataAll::GetMultiplicity**

parameter	name	direction	type	type modifier	multiplicity	default
r	bWithBrackets	in	bool			

	return	return	string			

Operation **IUMLDataAll::GetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLOperation			

Operation **IUMLDataAll::GetOwnedElementsOfKind**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	bRecursive	in	bool			
	return	return	IUMLDataList			

Operation **IUMLDataAll::GetSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	int			

Operation **IUMLDataAll::GetSourceLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLLifeline			

Operation **IUMLDataAll::GetStateIndex**

parameter	name	direction	type	type modifier	multiplicity	default
r	nTimeTickIndex		int			
	return	return	int			

Operation **IUMLDataAll::GetStereotypeApplicationForPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
r	nElement	in	ENUMUMLPredefinedElement			
	return	return	IUMLStereotypeApplication			

Operation **IUMLDataAll::GetStereotypeApplicationForStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipStereotype	in	IUMLStereotype			
	return	return	IUMLStereotypeApplication			

Operation **IUMLDataAll::GetTargetLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLLifeline			

Operation **IUMLDataAll::GetTextLabelText**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipTextLabel	in	IUMLGuiTextLabel			
	return	return	string			

Operation **IUMLDataAll::GetTimeTickLength**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	int			

Operation **IUMLDataAll::GetVisualStatePosition**

parameter	name	direction	type	type modifier	multiplicity	default
r	nStateIndex	in	int			
	return	return	int			

Operation **IUMLDataAll::GetVSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	int			

Operation **IUMLDataAll::Guard**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::GuiLinks**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::GuiOwner**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiElement			

Operation **IUMLDataAll::HandlerBody**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLExecutableNode			

Operation **IUMLDataAll::Handlers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::HSeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::IconFileName**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::ImplementingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavioralClassifier			

Operation **IUMLDataAll::ImportedElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLPackageableElement			

Operation **IUMLDataAll::ImportedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ImportedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLPackage			

Operation **IUMLDataAll::ImportingNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLNamespace			

Operation **IUMLDataAll::InActivity**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLActivity			

Operation **IUMLDataAll::Includes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::IncludingCase**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLUseCase			

Operation **IUMLDataAll::IncomingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Incomings**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::InformationFlowRealizations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::InformationSources**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::InformationTargets**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::InheritedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::InputElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::InputPins**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::InputValues**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::InsertActionTriggerAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLTrigger			

Operation **IUMLDataAll::InsertActivityEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	strKind	in	string			
	ipFrom	in	IUMLActivityNode			
	ipTo	in	IUMLActivityNode			
	return	return	IUMLActivityEdge			

Operation **IUMLDataAll::InsertActivityGroupAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			

	strKind return	in return	string IUMLEActivityGroup			

Operation **IUMLEDataAll::InsertActivityNodeAt**

parameter	name nIdx strKind return	direction in in return	type int string IUMLEActivityNode	type modifier	multiplicity	default

Operation **IUMLEDataAll::InsertActualGateAt**

parameter	name nIdx return	direction in return	type int IUMLEGate	type modifier	multiplicity	default

Operation **IUMLEDataAll::InsertAnnotatedElementAt**

parameter	name nIdx ipVal return	direction in in return	type int IUMLEElement void	type modifier	multiplicity	default

Operation **IUMLEDataAll::InsertArgumentAt**

parameter	name nIdx return	direction in return	type int IUMLEInputPin	type modifier	multiplicity	default

Operation **IUMLEDataAll::InsertCodeFileNameAt**

parameter	name nIdx strNewVal return	direction in in return	type int string void	type modifier	multiplicity	default

Operation **IUMLEDataAll::InsertCollaborationRoleAt**

parameter	name nIdx ipVal return	direction in in return	type int IUMLEConnectableElement void	type modifier	multiplicity	default

Operation **IUMLEDataAll::InsertCollaborationUseAt**

parameter	name nIdx return	direction in return	type int IUMLECollaborationUse	type modifier	multiplicity	default

Operation **IUMLEDataAll::InsertConnectionAt**

parameter	name nIdx	direction in	type int	type modifier	multiplicity	default

	return	return	IUMLConnectionPointReference			

Operation **IUMLDataAll::InsertConnectionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	IUMLPseudostate			

Operation **IUMLDataAll::InsertConstrainedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLElement			
	return	return	void			

Operation **IUMLDataAll::InsertConstrainingClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLClassifier			
	return	return	void			

Operation **IUMLDataAll::InsertConveyedAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLClassifier			
	return	return	void			

Operation **IUMLDataAll::InsertCoveredByAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLInteractionFragment			
	return	return	void			

Operation **IUMLDataAll::InsertDeploymentAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipDeployedArtifact	in	IUMLDeployedArtifact			
	return	return	IUMLDeployment			

Operation **IUMLDataAll::InsertEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipEdge	in	IUMLActivityEdge			
	return	return	void			

Operation **IUMLDataAll::InsertElementImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipImportedElement	in	IUMLPackageableElement			
	return	return	IUMLElementImport			

Operation **IUMLDataAll::InsertEntryAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLPseudostate			
	return	return	void			

Operation **IUMLDataAll::InsertEventActionResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	return	return	IUMLOutputPin			

Operation **IUMLDataAll::InsertExceptionTypeAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLClassifier			
	return	return	void			

Operation **IUMLDataAll::InsertExitAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipVal	in	IUMLPseudostate			
	return	return	void			

Operation **IUMLDataAll::InsertExtendAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipExtendedCase	in	IUMLUseCase			
	return	return	IUMLExtend			

Operation **IUMLDataAll::InsertExtensionLocationAt**

parameter	name	direction	type	type modifier	multiplicity	default
	nIdx	in	int			
	ipExtensionLocation	in	IUMLExtensionPoint			
	return	return	void			

Operation **IUMLDataAll::InsertExtensionPointAt**

parameter	name	direction	type	type modifier	multiplicity	default

r	nidx return	in return	int IUMLExtension Point			

Operation **IUMLDataAll::InsertFormalGateAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int IUMLEGate			

Operation **IUMLDataAll::InsertFragmentAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx strKind return	in in return	int string IUMLInteraction Fragment			

Operation **IUMLDataAll::InsertGeneralizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx ipGeneral return	in in return	int IUMLClassifier IUMLGeneraliz ation			

Operation **IUMLDataAll::InsertHandlerAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int IUMLException Handler			

Operation **IUMLDataAll::InsertIncludeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx ipIncludingCase return	in in return	int IUMLUseCase IUMLInclude			

Operation **IUMLDataAll::InsertInformationFlowRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx ipVal return	in in return	int IUMLRelationship void			

Operation **IUMLDataAll::InsertInformationSourceAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx ipVal return	in in return	int IUMLNamedElement void			

Operation **IUMLDataAll::InsertInformationTargetAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipVal	in	IUMLNamedElement			
	return	return	void			

Operation **IUMLDataAll::InsertInputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipNode	in	IUMLExpansionNode			
	return	return	void			

Operation **IUMLDataAll::InsertInputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLInputPin			

Operation **IUMLDataAll::InsertInStateAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipVal	in	IUMLState			
	return	return	void			

Operation **IUMLDataAll::InsertInterfaceRealizationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipContract	in	IUMLInterface			
	return	return	IUMLInterfaceRealization			

Operation **IUMLDataAll::InsertInterruptingEdgeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipEdge	in	IUMLActivityEdge			
	return	return	void			

Operation **IUMLDataAll::InsertLayerAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLGuiDiagramLayer			

Operation **IUMLDataAll::InsertLifelineAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLLifeline			

Operation **IUMLDataAll::InsertLocalPostConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLConstraint			

Operation **IUMLDataAll::InsertLocalPreConditionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLConstraint			

Operation **IUMLDataAll::InsertLowerUpperValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	strLower	in	string			
	strUpper	in	string			
	return	return	void			

Operation **IUMLDataAll::InsertManifestationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipUtilizedElement	in	IUMLPackageableElement			
	return	return	IUMLManifestation			

Operation **IUMLDataAll::InsertMessageAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLMessage			

Operation **IUMLDataAll::InsertNestedArtifactAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLArtifact			

Operation **IUMLDataAll::InsertNestedClassifierAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	strKind	in	string			
	return	return	IUMLClassifier			

Operation **IUMLDataAll::InsertNestedNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	strKind	in	string			
	return	return	IUMLNode			

Operation **IUMLDataAll::InsertNodeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipVal	in	UMLActivityNode			
	return	return	void			

Operation **IUMLDataAll::InsertObservationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipVal	in	UMLObservation			
	return	return	void			

Operation **IUMLDataAll::InsertOperandAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	UMLInteractionOperand			
			void			

Operation **IUMLDataAll::InsertOutputElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipNode	in	UMLExpansionNode			
	return	return	void			

Operation **IUMLDataAll::InsertOutputValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	UMLOutputPin			
			void			

Operation **IUMLDataAll::InsertOwnedArgumentAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	strKind	in	string			
	return	return	UMLValueSpecification			

Operation **IUMLDataAll::InsertOwnedAttributeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	UMLProperty			
			void			

Operation **IUMLDataAll::InsertOwnedBehaviorAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	strKind	in	string			
	return	return	UMLBehavior			

Operation **IUMLDataAll::InsertOwnedCommentAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLComment			

Operation **IUMLDataAll::InsertOwnedCommentTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nFromTextPos	in	int			
	nToTextPos	in	int			
	strAddress	in	string			
	return	return	IUMLComment TextHyperlink			

Operation **IUMLDataAll::InsertOwnedConnectorAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipFrom	in	IUMLConnectableElement			
	ipTo	in	IUMLConnectableElement			
	return	return	IUMLConnector			

Operation **IUMLDataAll::InsertOwnedDiagramAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipUMLParent	in	IUMLData			
	strKind	in	string			
	return	return	IUMLGuiDiagram			

Operation **IUMLDataAll::InsertOwnedGuiTextHyperlinkAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nFromTextPos	in	int			
	nToTextPos	in	int			
	strAddress	in	string			
	return	return	IUMLGuiTextHyperlink			

Operation **IUMLDataAll::InsertOwnedHyperlink2FileAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	strFilePathOrUrl	in	string			
	return	return	IUMLHyperlink2File			

Operation **IUMLDataAll::InsertOwnedHyperlink2GuiElementAt**

parameter	name	direction	type	type modifier	multiplicity	default

r	nidx	in	int			
	ipLinkedGuiElement		IUMLGuiVisibleElement			
	ipLinkedGuiElementCell		IUMLNamedElement			
	return	return	IUMLHyperlink2GuiElement			

Operation **IUMLDataAll::InsertOwnedHyperlink2ModelAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	ipLinkedData	in	IUMLData			
	return	return	IUMLHyperlink2Model			

Operation **IUMLDataAll::InsertOwnedLiteralAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	IUMLEnumerationLiteral			

Operation **IUMLDataAll::InsertOwnedOperationAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	IUMLOperation			

Operation **IUMLDataAll::InsertOwnedParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	IUMLParameter			

Operation **IUMLDataAll::InsertOwnedPortAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	IUMLPort			

Operation **IUMLDataAll::InsertOwnedRuleAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	return	return	IUMLConstraint			

Operation **IUMLDataAll::InsertOwnedTemplateBindingAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx	in	int			
	ipSignature	in	IUMLTemplateSignature			
	return	return	IUMLTemplateBinding			

Operation **IUMLDataAll::InsertOwnedTemplateParameterAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int IUMLClassifier TemplateParameter			

Operation **IUMLDataAll::InsertOwnedUseCaseAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx return	in return	int IUMLUseCase			

Operation **IUMLDataAll::InsertPackagedElementAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx strKind return	in in return	int string IUMLPackageableElement			

Operation **IUMLDataAll::InsertPackagedElementRelationshipAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx strKind ipFrom ipTo return	in in in in return	int string IUMLElement IUMLElement IUMLPackageableElement			

Operation **IUMLDataAll::InsertPackageImportAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx ipImportedPackage return	in in return	int IUMLPackage IUMLPackageImport			

Operation **IUMLDataAll::InsertPackageMergeAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx ipMergedPackage return	in in return	int IUMLPackage IUMLPackageMerge			

Operation **IUMLDataAll::InsertParameterSubstitutionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nidx ipFormalParameter ipActualParameter	in in in	int IUMLTemplateParameter IUMLParameter			

	eter return	return	tableElement UMLTemplate ParameterSubstitution			

Operation **IUMLDataAll::InsertPostconditionAt**

parameter	name nidx return	direction in return	type int IUMLConstraint	type modifier	multiplicity	default

Operation **IUMLDataAll::InsertPreconditionAt**

parameter	name nidx return	direction in return	type int IUMLConstraint	type modifier	multiplicity	default

Operation **IUMLDataAll::InsertProfileApplicationAt**

parameter	name nidx ipAppliedProfile return	direction in return	type int IUMLProfile IUMLProfileApplication	type modifier	multiplicity	default

Operation **IUMLDataAll::InsertQualifierAt**

parameter	name nidx return	direction in return	type int IUMLProperty	type modifier	multiplicity	default

Operation **IUMLDataAll::InsertRaisedExceptionAt**

parameter	name nidx ipVal return	direction in in return	type int IUMLType void	type modifier	multiplicity	default

Operation **IUMLDataAll::InsertRealizationAt**

parameter	name nidx ipRealizingClassifier return	direction in in return	type int IUMLClassifier IUMLComponentRealization	type modifier	multiplicity	default

Operation **IUMLDataAll::InsertRealizingConnectorAt**

parameter	name nidx ipVal return	direction in in return	type int IUMLConnector void	type modifier	multiplicity	default

Operation **IUMLDataAll::InsertRegionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLRegion			

Operation **IUMLDataAll::InsertResultAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	return	return	IUMLOutputPin			

Operation **IUMLDataAll::InsertSlotAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipDefiningFeature	in	IUMLStructuralFeature			
	return	return	IUMLSlot			

Operation **IUMLDataAll::InsertSlotInstanceValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipInstance	in	IUMLInstanceSpecification			
	return	return	IUMLInstanceValue			

Operation **IUMLDataAll::InsertSubjectAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipSubject	in	IUMLClassifier			
	return	return	void			

Operation **IUMLDataAll::InsertSubPartitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	strKind	in	string			
	return	return	IUMLActivityPartition			

Operation **IUMLDataAll::InsertSubVertexAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	strKind	in	string			
	return	return	IUMLVertex			

Operation **IUMLDataAll::InsertTransitionAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipSource	in	IUMLVertex			

	ipTarget return	in return	IUMLVertex IUMLTransition			

Operation **IUMLDataAll::InsertTriggerAt**

paramete r	name nIdx return	direction in return	type int IUMLTrigger	type modifier	multiplicity	default

Operation **IUMLDataAll::InsertValueAt**

paramete r	name nIdx strKind return	direction in return	type int string IUMLValueSpeci fication	type modifier	multiplicity	default

Operation **IUMLDataAll::InsertWaypointAt**

paramete r	name nIdx return	direction in return	type int IUMLGuiWaypo int	type modifier	multiplicity	default

Operation **IUMLDataAll::Instance**

paramete r	name return	direction return	type IUMLInstanceS pecification	type modifier	multiplicity	default

Operation **IUMLDataAll::InStates**

paramete r	name return	direction return	type IUMLDataList	type modifier	multiplicity	default

Operation **IUMLDataAll::IntegerValue**

paramete r	name return	direction return	type int	type modifier	multiplicity	default

Operation **IUMLDataAll::InteractionOperator**

paramete r	name return	direction return	type ENUMUMLInter actionOperator Kind	type modifier	multiplicity	default

Operation **IUMLDataAll::Interface**

paramete r	name return	direction return	type IUMLInterface	type modifier	multiplicity	default

Operation **IUMLDataAll::InterfaceRealizations**

paramete r	name return	direction return	type IUMLDataList	type modifier	multiplicity	default

Operation **IUMLDataAll::InterruptibleActivityRegions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::InterruptingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Invariant**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLConstraint			

Operation **IUMLDataAll::IsAbstract**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsActive**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsActivityReadOnly**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsBehavior**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsCodeLangNamespace**

parameter	name	direction	type	type modifier	multiplicity	default
r	nCodeLang	in	ENUMCodeLang			
	return	return	bool			

Operation **IUMLDataAll::IsCodeLangNamespaceRoot**

parameter	name	direction	type	type modifier	multiplicity	default
r	nCodeLang	in	ENUMCodeLang			
	return	return	bool			

Operation **IUMLDataAll::IsCodeProjectFile**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsCombineDuplicate**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsComposite**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsComputable**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsConjugated**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsControl**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsControlType**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsDerived**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsDerivedUnion**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsDimension**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsEditable**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsElementVisible**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipElement return	in return	<u>IUMLElement</u> bool			

Operation **IUMLDataAll::IsExternal**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsFinalSpecialization**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsFirstEvent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsHorizontal**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsIndirectlyInstantiated**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsKindOf**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string bool			

Operation **IUMLDataAll::IsLeaf**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsLocallyReentrant**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsLocked**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsMultiCast**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsMultiReceive**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsNavigable**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsNull**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsOrdered**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsOrthogonal**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsOwnedEnd**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsPositioned**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsPredefinedStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
r	nStereotype	in	ENUMUMLPredefinedElement			
	return	return	bool			

Operation **IUMLDataAll::IsQuery**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsReadOnly**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsReentrant**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsRelative**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsSameUMLData**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipUMLDataToC in compare		IUMLData			
	return	return	bool			

Operation **IUMLDataAll::IsService**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsShared**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsShowAsGeneralValueLifeline**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsSimple**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsStatic**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsStereotypeApplied**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipStereotype in		IUMLStereotype			
	return	return	bool			

Operation **IUMLDataAll::IsSubmachineState**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsSubstitutable**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsSynchronous**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::IsTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipTextLabel in		IUMLGuiTextLabel			

	return	return	bel bool			

Operation **IUMLDataAll::IsUnique**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IUMLDataAll::IsUnmarshall**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IUMLDataAll::IsUseForCodeEngineering**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IUMLDataAll::IsVarArgList**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IUMLDataAll::IsVisible**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IUMLDataAll::JoinSpec**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **IUMLDataAll::KindName**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **IUMLDataAll::Label**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **IUMLDataAll::Language**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **IUMLDataAll::Layer**

paramete r	name return	direction return	type IUMLGuiDiagramLayer	type modifier	multiplicity	default

Operation **IUMLDataAll::Layers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Left**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::Lifelines**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::LineBegin**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiElement			

Operation **IUMLDataAll::LineConnectionWaypoints**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::LineEnd**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiElement			

Operation **IUMLDataAll::LineLinks**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::LinkAddress**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::LinkedGuiElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiVisibleElement			

Operation **IUMLDataAll::LinkedGuiElementCell**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLNamedElement			

Operation **IUMLDataAll::LinkedModelElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLData			

Operation **IUMLDataAll::LinkedOwner**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLElement			

Operation **IUMLDataAll::LocalPostConditions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::LocalPreConditions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Location**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDeploymentTarget			

Operation **IUMLDataAll::LowerValues**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Manifestations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Mapping**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLOpaqueExpression			

Operation **IUMLDataAll::Max**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLValueSpecification			

Operation **IUMLDataAll::MaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLValueSpecification			

Operation **IUMLDataAll::MemberEnds**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Members**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::MergedPackage**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLPackage			

Operation **IUMLDataAll::MergeLayersAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nFromIdx	in	int			
	nToIdx	in	int			
	return	return	void			

Operation **IUMLDataAll::Message**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLMessage			

Operation **IUMLDataAll::MessageKind**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLMessageKind			

Operation **IUMLDataAll::Messages**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::MessageSort**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLMessageSort			

Operation **IUMLDataAll::MetaClass**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::Methods**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::MiddleWaypoint**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiMiddleWaypoint			

Operation **IUMLDataAll::Min**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	IUMLV alueSpecification			

Operation **IUMLDataAll::MinInt**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLV alueSpecification			

Operation **IUMLDataAll::Mode**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLExpansionKind			

Operation **IUMLDataAll::MoveTo**

parameter	name	direction	type	type modifier	multiplicity	default
r	nLeft	in	int			
	nTop	in	int			
	return	return	void			

Operation **IUMLDataAll::MustIsolate**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::Name**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::NameCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::Namespace**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLNamespace			

Operation **IUMLDataAll::NavigableOwnedEnds**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::NestedArtifacts**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::NestedClassifiers**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	IUMLDataList			

Operation **IUMLDataAll::NestedNodes**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

Operation **IUMLDataAll::NestedPackages**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

Operation **IUMLDataAll::NestingInterface**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLInterface			

Operation **IUMLDataAll::NestingPackage**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLPackage			

Operation **IUMLDataAll::Nodes**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

Operation **IUMLDataAll::NoteText**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	string			

Operation **IUMLDataAll::NoteTextEndPos**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLDataAll::NoteTextStartPos**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	int			

Operation **IUMLDataAll::Observations**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLDataList			

Operation **IUMLDataAll::OccurringEvent**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	IUMLEvent			

Operation **IUMLDataAll::OpenLink**

paramete r	name	direction	type	type modifier	multiplicity	default
	return	return	void			

Operation **IUMLDataAll::OperandGuard**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLInteractionConstraint			

Operation **IUMLDataAll::Operands**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Operation**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLOperation			

Operation **IUMLDataAll::Opposite**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLProperty			

Operation **IUMLDataAll::Ordering**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLObjectNodeOrderingKind			

Operation **IUMLDataAll::OutgoingEdges**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Outgoings**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OutputElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OutputPins**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OutputValues**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedActual**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLParameterableElement			

Operation **IUMLDataAll::OwnedArguments**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedAttributes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedBehaviors**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedComments**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedConnectors**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedDiagrams**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedDocComment**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLComment			

Operation **IUMLDataAll::OwnedDocCommentBody**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::OwnedElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedEnds**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedGuiElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedGuiNodeLinks**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedHyperlinks**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedLiterals**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedMembers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedOperations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLParameterableElement			

Operation **IUMLDataAll::OwnedParameters**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedPorts**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedRules**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedStereotypes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedTemplateBindings**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedTemplateParameters**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTemplateSignature			

Operation **IUMLDataAll::OwnedTypes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::OwnedUseCases**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Owner**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLElement			

Operation **IUMLDataAll::OwningAssociation**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLAssociation			

Operation **IUMLDataAll::OwningConstraint**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLConstraint			

Operation **IUMLDataAll::OwningElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLElement			

Operation **IUMLDataAll::OwningGuiNodeLink**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiNodeLink			

Operation **IUMLDataAll::OwningInstance**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLInstanceS			

	Specification					

Operation **IUMLDataAll::OwningInstanceSpec**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLInstanceSpecification			

Operation **IUMLDataAll::OwningLower**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLMultiplicityElement			

Operation **IUMLDataAll::OwningPackage**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLPackage			

Operation **IUMLDataAll::OwningParameter**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLParameter			

Operation **IUMLDataAll::OwningProperty**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLProperty			

Operation **IUMLDataAll::OwningSignal**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLSignal			

Operation **IUMLDataAll::OwningSlot**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLSlot			

Operation **IUMLDataAll::OwningState**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLState			

Operation **IUMLDataAll::OwningTemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTemplateParameter			

Operation **IUMLDataAll::OwningTransition**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLProtocolTransition			

Operation **IUMLDataAll::OwningUpper**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLMultiplicityElement			

Operation **IUMLDataAll::Package**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLPackage			

Operation **IUMLDataAll::PackagedElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::PackageImports**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::PackageMerges**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Parameter**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLParameter			

Operation **IUMLDataAll::ParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLParameterableElement			

Operation **IUMLDataAll::ParameterSignature**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTemplateSignature			

Operation **IUMLDataAll::ParameterSubstitutions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Parent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IDispatch			

Operation **IUMLDataAll::PinValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::PostCondition**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLConstraint			

Operation **IUMLDataAll::Postconditions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::PostTypeModifier**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::PosX**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::PosY**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::PreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLConstraint			

Operation **IUMLDataAll::Preconditions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ProfileApplications**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ProtectedNode**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLExecutableNode			

Operation **IUMLDataAll::Protocol**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLProtocolStateMachine			

Operation **IUMLDataAll::PseudostateKind**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLPseudostateKind			

Operation **IUMLDataAll::QualifiedName**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::Qualifiers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::RaisedExceptions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Realizations**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::RealizingClassifier**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLClassifier			

Operation **IUMLDataAll::RealizingConnectors**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ReceiveEvent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLMessageEnd			

Operation **IUMLDataAll::ReceivingPackage**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLPackage			

Operation **IUMLDataAll::ReferencedDiagram**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiDiagram			

Operation **IUMLDataAll::Referred**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	IUMList			

Operation **IUMList::RefersTo**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMInteraction			

Operation **IUMList::RegionAsInput**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMExpansionRegion			

Operation **IUMList::RegionAsOutput**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMExpansionRegion			

Operation **IUMList::Regions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMList			

Operation **IUMList::RelatedElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMList			

Operation **IUMList::RelativeNodes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMList			

Operation **IUMList::Represents**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMConnectableElement			

Operation **IUMList::Result**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMOutputPin			

Operation **IUMList::Results**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMList			

Operation **IUMList::Right**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::Role**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLConnectableElement			

Operation **IUMLDataAll::RoleBindings**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::ScrollPosX**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::ScrollPosY**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::Selection**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLDataAll::Selector**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLValueSpecification			

Operation **IUMLDataAll::SendEvent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLMessageEnd			

Operation **IUMLDataAll::SendSignal**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLSignal			

Operation **IUMLDataAll::SeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::SetCodeFileName**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	strNewVal	in	string			
	return	return	void			

Operation **IUMLDataAll::SetElementVisible**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipElement	in	IUMLElement			
	bVisible	in	bool			
	return	return	void			

Operation **IUMLDataAll::SetHSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	nPosition	in	int			
	return	return	void			

Operation **IUMLDataAll::SetHyperlinkFileAddress**

parameter	name	direction	type	type modifier	multiplicity	default
r	strNewVal	in	string			
	return	return	void			

Operation **IUMLDataAll::SetHyperlinkGuiElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipLinkedGuiElement		IUMLGuiVisibleElement			
	ipLinkedGuiElementCell		IUMLNamedElement			
	return	return	void			

Operation **IUMLDataAll::SetHyperlinkModelElementAddress**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipLinkedData	in	IUMLData			
	return	return	void			

Operation **IUMLDataAll::SetMultiplicity**

parameter	name	direction	type	type modifier	multiplicity	default
r	strNewVal	in	string			
	return	return	void			

Operation **IUMLDataAll::SetName**

parameter	name	direction	type	type modifier	multiplicity	default
r	strStartWith	in	string			
	return	return	string			

Operation **IUMLDataAll::SetNewActionValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	return	return	IUMLValueSpecification			

Operation **IUMLDataAll::SetNewCallTarget**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	return	return	IUMLInputPin			

Operation **IUMLDataAll::SetNewChangeExpression**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	return	return	IUMLValueSpecification			

Operation **IUMLDataAll::SetNewDefaultValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	return	return	IUMLValueSpecification			

Operation **IUMLDataAll::SetNewDefaultValueInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipInstance	in	IUMLInstanceSpecification			
	return	return	IUMLInstanceValue			

Operation **IUMLDataAll::SetNewDefaultValueLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
r	strNewVal	in	string			
	return	return	IUMLLiteralString			

Operation **IUMLDataAll::SetNewDoActivity**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	return	return	IUMLBehavior			

Operation **IUMLDataAll::SetNewEffect**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	return	return	IUMLBehavior			

Operation **IUMLDataAll::SetNewEntry**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	return	return	IUMLBehavior			

Operation **IUMLDataAll::SetNewExit**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind	in	string			
	return	return	IUMLBehavior			

Operation **IUMLDataAll::SetNewExpr**

parameter	name	direction	type	type modifier	multiplicity	default

r	strKind return	in return	string IUMLV alueSpecification

Operation **IUMLDataAll::SetNewInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLConstraint			

Operation **IUMLDataAll::SetNewMapping**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLOpaqueExpression			

Operation **IUMLDataAll::SetNewMax**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLV alueSpecification			

Operation **IUMLDataAll::SetNewMaxInt**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLV alueSpecification			

Operation **IUMLDataAll::SetNewMin**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLV alueSpecification			

Operation **IUMLDataAll::SetNewMinInt**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLV alueSpecification			

Operation **IUMLDataAll::SetNewOperandGuard**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLInteractionConstraint			

Operation **IUMLDataAll::SetNewOwnedParameteredElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLParameterableElement			

Operation **IUMLDataAll::SetNewPostCondition**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLConstrai t			

Operation **IUMLDataAll::SetNewPreCondition**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLConstrai t			

Operation **IUMLDataAll::SetNewProtocol**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLProtocolS tateMachine			

Operation **IUMLDataAll::SetNewSelector**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLValueSpe cification			

Operation **IUMLDataAll::SetNewSignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLInputPin			

Operation **IUMLDataAll::SetNewSpecification**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLValueSpe cification			

Operation **IUMLDataAll::SetNewSpecificationInstanceValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipInstance return	in return	IUMLInstanceS pecification IUMLInstanceV alue			

Operation **IUMLDataAll::SetNewSpecificationLiteralString**

parameter	name	direction	type	type modifier	multiplicity	default
r	strNewVal return	in return	string IUMLLiteralStri ng			

Operation **IUMLDataAll::SetNewStateInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLConstrai t			

Operation **IUMLDataAll::SetNewTemplateSignature**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTemplate Signature			

Operation **IUMLDataAll::SetNewTransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
r	strKind return	in return	string IUMLConstrai t			

Operation **IUMLDataAll::SetNewWhen**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTimeExpr ession			

Operation **IUMLDataAll::SetOperation**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipVal return	in return	IUMLOperation void			

Operation **IUMLDataAll::SetPos**

parameter	name	direction	type	type modifier	multiplicity	default
r	x y return	in in return	int int void			

Operation **IUMLDataAll::SetPredefinedTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx nProperty strNewValue return	in in in return	int ENUMUMLPred efinedElement string IUMLValueSpe cification			

Operation **IUMLDataAll::SetRect**

parameter	name	direction	type	type modifier	multiplicity	default
r	nLeft nTop nRight nBottom return	in in in in return	int int int int void			

Operation **IUMLDataAll::SetScrollPos**

parameter	name	direction	type	type modifier	multiplicity	default
r	nX	in	int			
	nY	in	int			
	return	return	void			

Operation **IUMLDataAll::SetSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	nPosition	in	int			
	return	return	void			

Operation **IUMLDataAll::SetSlotInstanceValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipForDefiningFeature	in	IUMLStructuralFeature			
	ipInstance	in	IUMLInstanceSpecification			
	return	return	IUMLInstanceValue			

Operation **IUMLDataAll::SetSlotValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipForDefiningFeature	in	IUMLStructuralFeature			
	strNewValue	in	string			
	return	return	IUMLValueSpecification			

Operation **IUMLDataAll::SetStateIndex**

parameter	name	direction	type	type modifier	multiplicity	default
r	nTimeTickIndex	in	int			
	nNewValue	in	int			
	return	return	void			

Operation **IUMLDataAll::SetStateIndexErased**

parameter	name	direction	type	type modifier	multiplicity	default
r	nTimeTickIndex	in	int			
	return	return	void			

Operation **IUMLDataAll::SetTaggedValueAt**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	ipDefiningFeature	in	IUMLStructuralFeature			
	strNewValue	in	string			
	return	return	IUMLValueSpecification			

Operation **IUMLDataAll::SetTextLabelVisible**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipTextLabel	in	IUMLGuiTextLabel			
	bVisible	in	bool			
	return	return	void			

Operation **IUMLDataAll::SetTimeTickLength**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	nNewVal	in	int			
	return	return	void			

Operation **IUMLDataAll::SetVisualStatePosition**

parameter	name	direction	type	type modifier	multiplicity	default
r	nStateIndex	in	int			
	nNewVal	in	int			
	return	return	void			

Operation **IUMLDataAll::SetVSeparatorPosition**

parameter	name	direction	type	type modifier	multiplicity	default
r	nIdx	in	int			
	nPosition	in	int			
	return	return	void			

Operation **IUMLDataAll::Signal**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLSignal			

Operation **IUMLDataAll::SignalTarget**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLInputPin			

Operation **IUMLDataAll::Signature**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTemplateSignature			

Operation **IUMLDataAll::SingleExecution**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::Slots**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Source**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEActivityNode			

Operation **IUMLEDataAll::Sources**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEDataList			

Operation **IUMLEDataAll::Specific**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEClassifier			

Operation **IUMLEDataAll::Specification**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEValueSpecification			

Operation **IUMLEDataAll::Specifics**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEDataList			

Operation **IUMLEDataAll::Start**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEOccurrenceSpecification			

Operation **IUMLEDataAll::State**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEState			

Operation **IUMLEDataAll::StateCompartmentEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLEDataAll::StateInvariant**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEConstraint			

Operation **IUMLEDataAll::StateMachine**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLEStateMachine			

Operation **IUMLEDataAll::Stereotype**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	IUMLStereotype			

Operation **IUMLDataAll::StereotypeApplications**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::StereotypedElementStyles**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiStyles			

Operation **IUMLDataAll::StringValue**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::StructuredActivityNodes**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Styles**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLGuiStyles			

Operation **IUMLDataAll::SubGroups**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Subjects**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Submachine**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLStateMachine			

Operation **IUMLDataAll::SubmachineStates**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::SubPartitions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::SubVertices**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::SuperClasses**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::SuperGroup**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLActivityGroup			

Operation **IUMLDataAll::SuperPartition**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLActivityPartition			

Operation **IUMLDataAll::SupplierDependencies**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Suppliers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Symbol**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::Target**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLActivityNode			

Operation **IUMLDataAll::Targets**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Template**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTemplateableElement			

Operation **IUMLDataAll::TemplateBinding**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTemplate			

	Binding					

Operation **IUMLDataAll::TemplateParameter**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTemplateParameter			

Operation **IUMLDataAll::TextEndPos**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::TextLabelElement**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLElement			

Operation **IUMLDataAll::TextLabelKind**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLGuiTextLabelKind			

Operation **IUMLDataAll::TextLabels**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::TextStartPos**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::TimeObservationEvent**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLNamedElement			

Operation **IUMLDataAll::TimeTickLengthCount**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::Top**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::Transformation**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLBehavior			

Operation **IUMLDataAll::TransitionGuard**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLConstraint			

Operation **IUMLDataAll::TransitionKind**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLTransitionKind			

Operation **IUMLDataAll::Transitions**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::TransitionSource**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLVertex			

Operation **IUMLDataAll::TransitionTarget**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLVertex			

Operation **IUMLDataAll::Triggers**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Type**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLType			

Operation **IUMLDataAll::TypedElements**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::UnapplyPredefinedStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
r	nStereotype	in	ENUMUMLPredefinedElement			
	return	return	void			

Operation **IUMLDataAll::UnapplyStereotype**

parameter	name	direction	type	type modifier	multiplicity	default
r	ipStereotype	in	IUMLStereotype			
	return	return	void			

Operation **IUMLDataAll::UnlimitedValue**

parameter	name	direction	type	type modifier	multiplicity	default

r	return	return	int			

Operation **IUMLDataAll::UpperBound**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **IUMLDataAll::UpperValues**

paramete r	name return	direction return	type IUMLDataList	type modifier	multiplicity	default

Operation **IUMLDataAll::UseCase**

paramete r	name return	direction return	type IUMLUseCase	type modifier	multiplicity	default

Operation **IUMLDataAll::UseCases**

paramete r	name return	direction return	type IUMLDataList	type modifier	multiplicity	default

Operation **IUMLDataAll::UseForForwardEngineering**

paramete r	name return	direction return	type bool	type modifier	multiplicity	default

Operation **IUMLDataAll::UserDefinedLinkName**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **IUMLDataAll::UtilizedElement**

paramete r	name return	direction return	type IUMLPackageableElement	type modifier	multiplicity	default

Operation **IUMLDataAll::UUID**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **IUMLDataAll::Value**

paramete r	name return	direction return	type string	type modifier	multiplicity	default

Operation **IUMLDataAll::Values**

paramete r	name return	direction return	type IUMLDataList	type modifier	multiplicity	default

Operation **IUMLDataAll::Viewpoint**

paramete	name	direction	type	type modifier	multiplicity	default

r	return	return	string			

Operation **IUMLDataAll::Visibility**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	ENUMUMLVisibilityKind			

Operation **IUMLDataAll::VisualStatePositionCount**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::VSeparatorCount**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

Operation **IUMLDataAll::WasUsedForCodeSynchronization**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	bool			

Operation **IUMLDataAll::Waypoints**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLDataList			

Operation **IUMLDataAll::Weight**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	string			

Operation **IUMLDataAll::When**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	IUMLTimeExpression			

Operation **IUMLDataAll::ZoomFactor**

parameter	name	direction	type	type modifier	multiplicity	default
r	return	return	int			

UML documentation generated by [UModel](#) UML Editor <http://www.altova.com/umodel>

08/03/15 13:42:06

UModelAPI - UMLData

Interface **UMLData**



Chapter 21

License Information

21 License Information

This section contains:

- Information about the [distribution of this software product](#)
- Information about [software activation and license metering](#)
- Information about the [intellectual property rights](#) related to this software product
- The [End-User License Agreement](#) governing the use of this software product

Please read this information carefully. It is binding upon you since you agreed to these terms when you installed this software product.

21.1 Electronic Software Distribution

This product is available through electronic software distribution, a distribution method that provides the following unique benefits:

- You can evaluate the software free-of-charge before making a purchasing decision.
 - Once you decide to buy the software, you can place your order online at the [Altova website](#) and immediately get a fully licensed product within minutes.
 - When you place an online order, you always get the latest version of our software.
 - The product package includes a comprehensive integrated onscreen help system. The latest version of the user manual is available at www.altova.com (i) in HTML format for online browsing, and (ii) in PDF format for download (and to print if you prefer to have the documentation on paper).
-

30-day evaluation period

After downloading this product, you can evaluate it for a period of up to 30 days free of charge. About 20 days into this evaluation period, the software will start to remind you that it has not yet been licensed. The reminder message will be displayed once each time you start the application. If you would like to continue using the program after the 30-day evaluation period, you have to purchase an [Altova Software License Agreement](#), which is delivered in the form of a key-code that you enter into the Software Activation dialog to unlock the product. You can purchase your license at the online shop at the [Altova website](#).

Helping Others within Your Organization to Evaluate the Software

If you wish to distribute the evaluation version within your company network, or if you plan to use it on a PC that is not connected to the Internet, you may only distribute the Setup programs, provided that they are not modified in any way. Any person that accesses the software installer that you have provided, must request their own 30-day evaluation license key code and after expiration of their evaluation period, must also purchase a license in order to be able to continue using the product.

For further details, please refer to the [Altova Software License Agreement](#) at the end of this section.

21.2 Software Activation and License Metering

As part of Altova's Software Activation, the software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova-operated license server and validating the authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the software and to improve customer service. Activation is based on the exchange of license related data such as operating system, IP address, date/time, software version, and computer name, along with other information between your computer and an Altova license server.

Your Altova product has a built-in license metering module that further helps you avoid any unintentional violation of the End User License Agreement. Your product is licensed either as a single-user or multi-user installation, and the license-metering module makes sure that no more than the licensed number of users use the application concurrently.

This license-metering technology uses your local area network (LAN) to communicate between instances of the application running on different computers.

Single license

When the application starts up, as part of the license metering process, the software sends a short broadcast datagram to find any other instance of the product running on another computer in the same network segment. If it doesn't get any response, it will open a port for listening to other instances of the application.

Multi license

If more than one instance of the application is used within the same LAN, these instances will briefly communicate with each other on startup. These instances exchange key-codes in order to help you to better determine that the number of concurrent licenses purchased is not accidentally violated. This is the same kind of license metering technology that is common in the Unix world and with a number of database development tools. It allows Altova customers to purchase reasonably-priced concurrent-use multi-user licenses.

We have also designed the applications so that they send few and small network packets so as to not put a burden on your network. The TCP/IP ports (2799) used by your Altova product are officially registered with the IANA (see [the IANA website \(http://www.iana.org/\)](http://www.iana.org/) for details) and our license-metering module is tested and proven technology.

If you are using a firewall, you may notice communications on port 2799 between the computers that are running Altova products. You are, of course, free to block such traffic between different groups in your organization, as long as you can ensure by other means, that your license agreement is not violated.

You will also notice that, if you are online, your Altova product contains many useful functions; these are unrelated to the license-metering technology.

21.3 Intellectual Property Rights

The Altova Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and all accompanying printed materials. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

Altova software contains certain Third Party Software that is also protected by intellectual property laws, including without limitation applicable copyright laws as described in detail at http://www.altova.com/legal_3rdparty.html.

All other names or trademarks are the property of their respective owners.

21.4 Altova End User License Agreement

THIS IS A LEGAL DOCUMENT -- RETAIN FOR YOUR RECORDS

ALTOVA® END USER LICENSE AGREEMENT

Licensors:
Altova GmbH
Rudolfsplatz 13a/9
A-1010 Wien
Austria

Important - Read Carefully. Notice to User:

This End User License Agreement ("Agreement") is a legal document between you and Altova GmbH ("Altova"). It is important that you read this document before using the Altova-provided software ("Software") and any accompanying documentation, including, without limitation printed materials, 'online' files, or electronic documentation ("Documentation"). By clicking the "I accept" and "Next" buttons below, or by installing, or otherwise using the Software, you agree to be bound by the terms of this Agreement as well as the Altova Privacy Policy ("Privacy Policy") including, without limitation, the warranty disclaimers, limitation of liability, data use and termination provisions below, whether or not you decide to purchase the Software. You agree that this agreement is enforceable like any written agreement negotiated and signed by you. If you do not agree, you are not licensed to use the Software, and you must destroy any downloaded copies of the Software in your possession or control. You may print a copy of this Agreement as part of the installation process at the time of acceptance. Alternatively, a copy of this Agreement may be found at <http://www.altova.com/eula> and a copy of the Privacy Policy may be found at <http://www.altova.com/privacy>.

1. SOFTWARE LICENSE

(a) License Grant.

(i) Upon your acceptance of this Agreement Altova grants you a non-exclusive, non-transferable (except as provided below), limited license, without the right to grant sublicenses, to install and use a copy of the Software on one compatible personal computer or workstation in the same local area network (LAN) up to the Permitted Number of computers. Subject to the limitations set forth in Section 1(c), you may install and use a copy of the Software on more than one of your compatible personal computers or workstations if you have purchased a Named-User license. Subject to the limitations set forth in Sections 1(d) and 1(e), users may use the software concurrently on a network. The Permitted Number of computers and/or users and the type of license, e.g. Installed, Named-Users, and Concurrent-User, shall be determined and specified at such time as you elect to purchase the Software. Installed user licenses are intended to be fixed and not concurrent. In other words, you cannot uninstall the Software on one machine in order to reinstall that license to a different machine and then uninstall and reinstall back to the original machine. Installations should be static. Notwithstanding the foregoing, permanent uninstallations and redeployments are acceptable in limited circumstances such as if an employee leaves the company or the machine is permanently decommissioned. During the evaluation period, hereinafter defined, only a single user may install and use the software on one (1) personal computer or workstation. If you have licensed the Software as part of a suite of Altova software products (collectively, the "Suite") and have not installed each product individually, then the Agreement governs your use of all of the software included in the Suite.

(ii) If you have licensed SchemaAgent, then the terms and conditions of this Agreement apply to your use of the SchemaAgent server software ("SchemaAgent Server") included therein, as applicable, and you are licensed to use SchemaAgent Server solely in connection with your use of Altova Software and solely for the purposes described in the accompanying documentation.

(iii) If you have licensed Software that enables users to generate source code, your license to install and use a copy of the Software as provided herein permits you to generate source code based on (i) Altova Library modules that are included in the Software (such generated code hereinafter referred to as the "Restricted Source Code") and (ii) schemas or mappings that you create or provide (such code as may be generated from your schema or mapping source materials hereinafter referred to as the "Unrestricted Source Code"). In addition to the rights granted herein, Altova grants you a non-exclusive, non-transferable, limited license to compile the complete generated code (comprised of the combination of the Restricted Source Code and the Unrestricted Source Code) into executable object code form, and to use, copy, distribute or license that executable. You may not distribute or redistribute, sublicense, sell, or transfer the Restricted Source Code to a third-party in the un-compiled form unless said third-party already has a license to the Restricted Source Code through their separate agreement with Altova. Notwithstanding anything to the contrary herein, you may not distribute, incorporate or combine with other software, or otherwise use the Altova Library modules or Restricted Source Code, or any Altova intellectual property embodied in or associated with the Altova Library modules or Restricted Source Code, in any manner that would subject the Restricted Source Code to the terms of a copyleft, free software or open source license that would require the Restricted Source Code or Altova Library modules source code to be disclosed in source code form. Notwithstanding anything to the contrary herein, you may not use the Software to develop and distribute other software programs that directly compete with any Altova software or service without prior written permission. Altova reserves all other rights in and to the Software. With respect to the feature(s) of UModel that permit reverse-engineering of your own source code or other source code that you have lawfully obtained, such use by you does not constitute a violation of this Agreement. Except as otherwise expressly permitted in Section 1(j) reverse engineering of the Software is strictly prohibited as further detailed therein.

(iv) In the event Restricted Source Code is incorporated into executable object code form, you will include the following statement in (1) introductory splash screens, or if none, within one or more screens readily accessible by the end-user, and (2) in the electronic and/or hard copy documentation: "Portions of this program were developed using Altova® [name of Altova Software, e.g. MapForce® 2016] and includes libraries owned by Altova GmbH, Copyright © 2007-2016 Altova GmbH (www.altova.com)."

(b) Server Use for Installation and Use of SchemaAgent. You may install one (1) copy of the Software on a computer file server within your internal network solely for the purpose of downloading and installing the Software onto other computers within your internal network up to the Permitted Number of computers in a commercial environment only. If you have licensed SchemaAgent, then you may install SchemaAgent Server on any server computer or workstation and use it in connection with your Software. No other network use is permitted, including without limitation using the Software either directly or through commands, data or instructions from or to a computer not part of your internal network, for Internet or Web-hosting services or by any user not licensed to use this copy of the Software through a valid license from Altova.

(c) Named-Use. If you have licensed the "Named-User" version of the software, you may install the Software on up to five (5) compatible personal computers or workstations of which you are the primary user thereby allowing you to switch from one computer to the other as necessary provided that only one (1) instance of the Software will be used by you as the Named-User at any

given time. If you have purchased multiple Named-User licenses, each individual Named-User will receive a separate license key code.

(d) Concurrent Use in Same Local Area Network (LAN). If you have licensed a "Concurrent-User" version of the Software, you may install the Software on any compatible computers in a commercial environment only, up to ten (10) times the Permitted Number of users, provided that only the Permitted Number of users actually use the Software at the same time and further provided that the computers on which the Software is installed are on the same local area network (LAN). The Permitted Number of concurrent users shall be delineated at such time as you elect to purchase the Software licenses. Each separate local area network (LAN) requires its own set of separate Concurrent User Licenses for those wishing to use the Concurrent User versions of the Software in more than one location or on more than one network, all subject to the above Permitted Number limitations and based on the number of users using the Software. If a computer is not on the same local area network (LAN), then a locally installed user license or a license dedicated to concurrent use in a virtual environment is required.

(e) Concurrent Use in Virtual Environment. If you have purchased Concurrent-User Licenses, you may install a copy of the Software on a single host terminal server (Microsoft Terminal Server or Citrix Metaframe), application virtualization server (Microsoft App-V, Citrix XenApp, or VMWare ThinApp) or virtual machine environment within your internal network for the sole and exclusive purpose of permitting individual users within your organization to access and use the Software through a terminal server, application virtualization session, or virtual machine environment from another computer provided that the total number of users that access or use the Software concurrently at any given point in time on such network, virtual machine or terminal server does not exceed the Permitted Number; and provided that the total number of users authorized to use the Software through the terminal server, application virtualization session, or virtual machine environment does not exceed ten (10) times the Permitted Number of users. Key codes for concurrent users cannot be deployed to more than one host terminal server, application virtualization server or virtual machine environment. You must deploy a reliable and accurate means of preventing users from exceeding the Permitted Number of concurrent users. Altova makes no warranties or representations about the performance of Altova software in a terminal server, application virtualization session, or virtual machine environment and the foregoing are expressly excluded from the limited warranty in Section 5 hereof. Technical support is not available with respect to issues arising from use in such environments.

(f) Backup and Archival Copies. You may make one (1) backup and one (1) archival copy of the Software, provided your backup and archival copies are not installed or used on any computer and further provided that all such copies shall bear the original and unmodified copyright, patent and other intellectual property markings that appear on or in the Software. You may not transfer the rights to a backup or archival copy unless you transfer all rights in the Software as provided under Section 3.

(g) Key Codes, Upgrades and Updates. Prior to your purchase and as part of the registration for the thirty (30) day evaluation period, as applicable, you will receive an evaluation key code. You will receive a purchase key code when you elect to purchase the Software from either Altova GmbH or an authorized reseller. The purchase key code will enable you to activate the Software beyond the initial evaluation period. You may not re-license, reproduce or distribute any key code except with the express written permission of Altova. If the Software that you have licensed is an upgrade or an update, then the latest update or upgrade that you download and install replaces all or part of the Software previously licensed. The update or upgrade and the associated license keys does not constitute the granting of a second license to the Software in that you may not use the upgrade or updated copy in addition to the copy of the Software that it is replacing and whose license has terminated.

(h) Title. Title to the Software is not transferred to you. Ownership of all copies of the Software and of copies made by you is vested in Altova, subject to the rights of use granted to you in this Agreement. As between you and Altova, documents, files, stylesheets, generated program code (including the Unrestricted Source Code) and schemas that are authored or created by you via your utilization of the Software, in accordance with its Documentation and the terms of this Agreement, are your property unless they are created using Evaluation Software, as defined in Section 4 of this Agreement, in which case you have only a limited license to use any output that contains generated program code (including Unrestricted Source Code) such as Java, C++, C#, VB.NET or XSLT and associated project files and build scripts, as well as generated XML, XML Schemas, documentation, UML diagrams, and database structures only for the thirty (30) day evaluation period.

(i) Reverse Engineering. Except and to the limited extent as may be otherwise specifically provided by applicable law in the European Union, you may not reverse engineer, decompile, disassemble or otherwise attempt to discover the source code, underlying ideas, underlying user interface techniques or algorithms of the Software by any means whatsoever, directly or indirectly, or disclose any of the foregoing, except to the extent you may be expressly permitted to decompile under applicable law in the European Union, if it is essential to do so in order to achieve operability of the Software with another software program, and you have first requested Altova to provide the information necessary to achieve such operability and Altova has not made such information available. Altova has the right to impose reasonable conditions and to request a reasonable fee before providing such information. Any information supplied by Altova or obtained by you, as permitted hereunder, may only be used by you for the purpose described herein and may not be disclosed to any third party or used to create any software which is substantially similar to the expression of the Software. Requests for information from users in the European Union with respect to the above should be directed to the Altova Customer Support Department.

(j) Other Restrictions. You may not loan, rent, lease, sublicense, distribute or otherwise transfer all or any portion of the Software to third parties except to the limited extent set forth in Section 3 or as otherwise expressly provided. You may not copy the Software except as expressly set forth above, and any copies that you are permitted to make pursuant to this Agreement must contain the same copyright, patent and other intellectual property markings that appear on or in the Software. You may not modify, adapt or translate the Software. You may not, directly or indirectly, encumber or suffer to exist any lien or security interest on the Software; knowingly take any action that would cause the Software to be placed in the public domain; or use the Software in any computer environment not specified in this Agreement. You may not permit any use of or access to the Software by any third party in connection with a commercial service offering, such as for a cloud-based or web-based SaaS offering.

You will comply with applicable law and Altova's instructions regarding the use of the Software. You agree to notify your employees and agents who may have access to the Software of the restrictions contained in this Agreement and to ensure their compliance with these restrictions.

(k) NO GUARANTEE. THE SOFTWARE IS NEITHER GUARANTEED NOR WARRANTED TO BE ERROR-FREE NOR SHALL ANY LIABILITY BE ASSUMED BY ALTOVA IN THIS RESPECT. NOTWITHSTANDING ANY SUPPORT FOR ANY TECHNICAL STANDARD, THE SOFTWARE IS NOT INTENDED FOR USE IN OR IN CONNECTION WITH, WITHOUT LIMITATION, THE OPERATION OF NUCLEAR FACILITIES, AIRCRAFT NAVIGATION, COMMUNICATION SYSTEMS, AIR TRAFFIC CONTROL EQUIPMENT, MEDICAL DEVICES OR LIFE SUPPORT SYSTEMS, MEDICAL OR HEALTH CARE APPLICATIONS, OR OTHER

APPLICATIONS WHERE THE FAILURE OF THE SOFTWARE OR ERRORS IN DATA PROCESSING COULD LEAD TO DEATH, PERSONAL INJURY OR SEVERE PHYSICAL OR ENVIRONMENTAL DAMAGE. YOU AGREE THAT YOU ARE SOLELY RESPONSIBLE FOR THE ACCURACY AND ADEQUACY OF THE SOFTWARE AND ANY DATA GENERATED OR PROCESSED BY THE SOFTWARE FOR YOUR INTENDED USE AND YOU WILL DEFEND, INDEMNIFY AND HOLD ALTOVA, ITS OFFICERS AND EMPLOYEES HARMLESS FROM ANY THIRD PARTY CLAIMS, DEMANDS, OR SUITS THAT ARE BASED UPON THE ACCURACY AND ADEQUACY OF THE SOFTWARE IN YOUR USE OR ANY DATA GENERATED BY THE SOFTWARE IN YOUR USE.

2. INTELLECTUAL PROPERTY RIGHTS

You acknowledge that the Software and any copies that you are authorized by Altova to make are the intellectual property of and are owned by Altova and its suppliers. The structure, organization and code of the Software are the valuable trade secrets and confidential information of Altova and its suppliers. The Software is protected by copyright, including without limitation by United States Copyright Law, international treaty provisions and applicable laws in the country in which it is being used. You acknowledge that Altova retains the ownership of all patents, copyrights, trade secrets, trademarks and other intellectual property rights pertaining to the Software, and that Altova's ownership rights extend to any images, photographs, animations, videos, audio, music, text and "applets" incorporated into the Software and all accompanying printed materials. You will take no actions which adversely affect Altova's intellectual property rights in the Software. Trademarks shall be used in accordance with accepted trademark practice, including identification of trademark owners' names. Trademarks may only be used to identify printed output produced by the Software, and such use of any trademark does not give you any right of ownership in that trademark. Altova®, XMLSpy®, Authentic®, StyleVision®, MapForce®, UModel®, DatabaseSpy®, DiffDog®, SchemaAgent®, SemanticWorks®, MissionKit®, Markup Your Mind®, Nanonull™, RaptorXML™, RaptorXML Server™, RaptorXML +XBRL Server™, Powered By RaptorXML™, FlowForce Server™, StyleVision Server™, and MapForce Server™ are trademarks of Altova GmbH. (pending or registered in numerous countries). Unicode and the Unicode Logo are trademarks of Unicode, Inc. Windows, Windows XP, Windows Vista, Windows 7, and Windows 8 are trademarks of Microsoft. W3C, CSS, DOM, MathML, RDF, XHTML, XML and XSL are trademarks (registered in numerous countries) of the World Wide Web Consortium (W3C); marks of the W3C are registered and held by its host institutions, MIT, INRIA and Keio. Except as expressly stated above, this Agreement does not grant you any intellectual property rights in the Software. Notifications of claimed copyright infringement should be sent to Altova's copyright agent as further provided on the Altova Web Site.

3. LIMITED TRANSFER RIGHTS

Notwithstanding the foregoing, you may transfer all your rights to use the Software to another person or legal entity provided that: (a) you also transfer this Agreement, the Software and all other software or hardware bundled or pre-installed with the Software, including all copies, updates and prior versions, and all copies of font software converted into other formats, to such person or entity; (b) you retain no copies, including backups and copies stored on a computer; (c) the receiving party secures a personalized key code from Altova; and (d) the receiving party accepts the terms and conditions of this Agreement and any other terms and conditions upon which you legally purchased a license to the Software. Notwithstanding the foregoing, you may not transfer education, pre-release, or not-for-resale copies of the Software.

4. PRE-RELEASE AND EVALUATION PRODUCT ADDITIONAL TERMS

If the product you have received with this license is pre-commercial release or beta Software ("Pre-

release Software”), then this Section applies. In addition, this section applies to all evaluation and/or demonstration copies of Altova software (“Evaluation Software”) and continues in effect until you purchase a license. To the extent that any provision in this section is in conflict with any other term or condition in this Agreement, this section shall supersede such other term(s) and condition(s) with respect to the Pre-release and/or Evaluation Software, but only to the extent necessary to resolve the conflict. You acknowledge that the Pre-release Software is a pre-release version, does not represent final product from Altova, and may contain bugs, errors and other problems that could cause system or other failures and data loss. CONSEQUENTLY, THE PRE-RELEASE AND/OR EVALUATION SOFTWARE IS PROVIDED TO YOU **“AS-IS” WITH NO WARRANTIES FOR USE OR PERFORMANCE**, AND ALTOVA DISCLAIMS ANY WARRANTY OR LIABILITY OBLIGATIONS TO YOU OF ANY KIND, WHETHER EXPRESS OR IMPLIED. WHERE LEGALLY LIABILITY CANNOT BE EXCLUDED FOR PRE-RELEASE AND/OR EVALUATION SOFTWARE, BUT IT MAY BE LIMITED, ALTOVA’S LIABILITY AND THAT OF ITS SUPPLIERS SHALL BE LIMITED TO THE SUM OF FIFTY DOLLARS (USD \$50) IN TOTAL. If the Evaluation Software has a time-out feature, then the software will cease operation after the conclusion of the designated evaluation period. Upon such expiration date, your license will expire unless otherwise extended. Your license to use any output created with the Evaluation Software that contains generated program code (including Unrestricted Source Code) such as Java, C++, C, VB.NET or XSLT and associated project files and build scripts as well as generated XML, XML Schemas, documentation, UML diagrams, and database structures terminates automatically upon the expiration of the designated evaluation period but the license to use such output is revived upon your purchase of a license for the Software that you evaluated and used to create such output. Access to any files created with the Evaluation Software is entirely at your risk. You acknowledge that Altova has not promised or guaranteed to you that Pre-release Software will be announced or made available to anyone in the future, that Altova has no express or implied obligation to you to announce or introduce the Pre-release Software, and that Altova may not introduce a product similar to or compatible with the Pre-release Software. Accordingly, you acknowledge that any research or development that you perform regarding the Pre-release Software or any product associated with the Pre-release Software is done entirely at your own risk. During the term of this Agreement, if requested by Altova, you will provide feedback to Altova regarding testing and use of the Pre-release Software, including error or bug reports. If you have been provided the Pre-release Software pursuant to a separate written agreement, your use of the Software is governed by such agreement. You may not sublicense, lease, loan, rent, distribute or otherwise transfer the Pre-release Software. Upon receipt of a later unreleased version of the Pre-release Software or release by Altova of a publicly released commercial version of the Software, whether as a stand-alone product or as part of a larger product, you agree to return or destroy all earlier Pre-release Software received from Altova and to abide by the terms of the license agreement for any such later versions of the Pre-release Software.

5. LIMITED WARRANTY AND LIMITATION OF LIABILITY

(a) **Limited Warranty and Customer Remedies.** Altova warrants to the person or entity that first purchases a license for use of the Software pursuant to the terms of this Agreement that (i) the Software will perform substantially in accordance with any accompanying Documentation for a period of ninety (90) days from the date of receipt, and (ii) any support services provided by Altova shall be substantially as described in Section 6 of this agreement. Some states and jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. To the extent allowed by applicable law, implied warranties on the Software, if any, are limited to ninety (90) days. Altova’s and its suppliers’ entire liability and your exclusive remedy shall be, at Altova’s option, either (i) return of the price paid, if any, or (ii) repair or replacement of the Software that does not meet Altova’s Limited Warranty and which is returned to Altova with a copy of your receipt. This Limited Warranty is void if failure of the Software has resulted from accident, abuse, misapplication, abnormal use, Trojan horse, virus, or

any other malicious external code. Any replacement Software will be warranted for the remainder of the original warranty period or thirty (30) days, whichever is longer. This limited warranty does not apply to Evaluation and/or Pre-release Software.

(b) No Other Warranties and Disclaimer. THE FOREGOING LIMITED WARRANTY AND REMEDIES STATE THE SOLE AND EXCLUSIVE REMEDIES FOR ALTOVA OR ITS SUPPLIER'S BREACH OF WARRANTY. ALTOVA AND ITS SUPPLIERS DO NOT AND CANNOT WARRANT THE PERFORMANCE OR RESULTS YOU MAY OBTAIN BY USING THE SOFTWARE. EXCEPT FOR THE FOREGOING LIMITED WARRANTY, AND FOR ANY WARRANTY, CONDITION, REPRESENTATION OR TERM TO THE EXTENT WHICH THE SAME CANNOT OR MAY NOT BE EXCLUDED OR LIMITED BY LAW APPLICABLE TO YOU IN YOUR JURISDICTION, ALTOVA AND ITS SUPPLIERS MAKE NO WARRANTIES, CONDITIONS, REPRESENTATIONS OR TERMS, EXPRESS OR IMPLIED, WHETHER BY STATUTE, COMMON LAW, CUSTOM, USAGE OR OTHERWISE AS TO ANY OTHER MATTERS. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, ALTOVA AND ITS SUPPLIERS DISCLAIM ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, INFORMATIONAL CONTENT OR ACCURACY, QUIET ENJOYMENT, TITLE AND NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.

(c) Limitation of Liability. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW EVEN IF A REMEDY FAILS ITS ESSENTIAL PURPOSE, IN NO EVENT SHALL ALTOVA OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF ALTOVA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, ALTOVA'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT. Because some states and jurisdictions do not allow the exclusion or limitation of liability, the above limitation may not apply to you. In such states and jurisdictions, Altova's liability shall be limited to the greatest extent permitted by law and the limitations or exclusions of warranties and liability contained herein do not prejudice applicable statutory consumer rights of person acquiring goods otherwise than in the course of business. The disclaimer and limited liability above are fundamental to this Agreement between Altova and you.

(d) Infringement Claims. Altova will indemnify and hold you harmless and will defend or settle any claim, suit or proceeding brought against you by a third party that is based upon a claim that the content contained in the Software infringes a copyright or violates an intellectual or proprietary right protected by United States or European Union law ("Claim"), but only to the extent the Claim arises directly out of the use of the Software and subject to the limitations set forth in Section 5 of this Agreement except as otherwise expressly provided. You must notify Altova in writing of any Claim within ten (10) business days after you first receive notice of the Claim, and you shall provide to Altova at no cost such assistance and cooperation as Altova may reasonably request from time to time in connection with the defense of the Claim. Altova shall have sole control over any Claim (including, without limitation, the selection of counsel and the right to settle on your behalf on any terms Altova deems desirable in the sole exercise of its discretion). You may, at your sole cost, retain separate counsel and participate in the defense or settlement negotiations. Altova shall pay actual damages, costs, and attorney fees awarded

against you (or payable by you pursuant to a settlement agreement) in connection with a Claim to the extent such direct damages and costs are not reimbursed to you by insurance or a third party, to an aggregate maximum equal to the purchase price of the Software. If the Software or its use becomes the subject of a Claim or its use is enjoined, or if in the opinion of Altova's legal counsel the Software is likely to become the subject of a Claim, Altova shall attempt to resolve the Claim by using commercially reasonable efforts to modify the Software or obtain a license to continue using the Software. If in the opinion of Altova's legal counsel the Claim, the injunction or potential Claim cannot be resolved through reasonable modification or licensing, Altova, at its own election, may terminate this Agreement without penalty, and will refund to you on a pro rata basis any fees paid in advance by you to Altova. THE FOREGOING CONSTITUTES ALTOVA'S SOLE AND EXCLUSIVE LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT. This indemnity does not apply to situations where the alleged infringement, whether patent or otherwise, is the result of a combination of the Altova software and additional elements supplied by you.

6. SUPPORT AND MAINTENANCE

Altova offers multiple optional "Support & Maintenance Package(s)" ("SMP") for the version of Software product edition that you have licensed, which you may elect to purchase in addition to your Software license. The Support Period, hereinafter defined, covered by such SMP shall be delineated at such time as you elect to purchase a SMP. Your rights with respect to support and maintenance as well as your upgrade eligibility depend on your decision to purchase SMP and the level of SMP that you have purchased:

(a) If you have not purchased SMP, you will receive the Software AS IS and will not receive any maintenance releases or updates. However, Altova, at its option and in its sole discretion on a case by case basis, may decide to offer maintenance releases to you as a courtesy, but these maintenance releases will not include any new features in excess of the feature set at the time of your purchase of the Software. In addition, Altova will provide free technical support to you for thirty (30) days after the date of your purchase (the "Support Period" for the purposes of this paragraph 6(a), and Altova, in its sole discretion on a case by case basis, may also provide free courtesy technical support during your thirty (30) day evaluation period. Technical support is provided via a Web-based support form only, and there is no guaranteed response time.

(b) If you have purchased SMP, then solely for the duration of its delineated Support Period, **you are eligible to receive the version of the Software edition** that you have licensed and all maintenance releases and updates for that edition that are released during your Support Period. For the duration of your SMP's Support Period, you will also be eligible to receive upgrades to the comparable edition of the next version of the Software that succeeds the Software edition that you have licensed for applicable upgrades released during your Support Period. The specific upgrade edition that you are eligible to receive based on your Support Period is further detailed in the SMP that you have purchased. Software that is introduced as separate product is not included in SMP. Maintenance releases, updates and upgrades may or may not include additional features. In addition, Altova will provide Priority Technical Support to you for the duration of the Support Period. Priority Technical Support is provided via a Web-based support form only and Altova will make commercially reasonable efforts to respond via e-mail to all requests within forty-eight (48) hours during Altova's business hours (MO-FR, 8am UTC – 10pm UTC, Austrian and US holidays excluded) and to make reasonable efforts to provide work-arounds to errors reported in the Software.

During the Support Period you may also report any Software problem or error to Altova. If Altova determines that a reported reproducible material error in the Software exists and significantly impairs the usability and utility of the Software, Altova agrees to use reasonable commercial efforts to correct or provide a usable work-around solution in an upcoming maintenance release or

update, which is made available at certain times at Altova's sole discretion.

If Altova, in its discretion, requests written verification of an error or malfunction discovered by you or requests supporting example files that exhibit the Software problem, you shall promptly provide such verification or files, by email, telecopy, or overnight mail, setting forth in reasonable detail the respects in which the Software fails to perform. You shall use reasonable efforts to cooperate in diagnosis or study of errors. Altova may include error corrections in maintenance releases, updates, or new major releases of the Software. Altova is not obligated to fix errors that are immaterial. Immaterial errors are those that do not significantly impact use of the Software as determined by Altova in its sole discretion. Whether or not you have purchased the Support & Maintenance Package, technical support only covers issues or questions resulting directly out of the operation of the Software and Altova will not provide you with generic consultation, assistance, or advice under any circumstances.

Updating Software may require the updating of software not covered by this Agreement before installation. Updates of the operating system and application software not specifically covered by this Agreement are your responsibility and will not be provided by Altova under this Agreement. Altova's obligations under this Section 6 are contingent upon your proper use of the Software and your compliance with the terms and conditions of this Agreement at all times. Altova shall be under no obligation to provide the above technical support if, in Altova's opinion, the Software has failed due to the following conditions: (i) damage caused by the relocation of the Software to another location or CPU; (ii) alterations, modifications or attempts to change the Software without Altova's written approval; (iii) causes external to the Software, such as natural disasters, the failure or fluctuation of electrical power, or computer equipment failure; (iv) your failure to maintain the Software at Altova's specified release level; or (v) use of the Software with other software without Altova's prior written approval. It will be your sole responsibility to: (i) comply with all Altova-specified operating and troubleshooting procedures and then notify Altova immediately of Software malfunction and provide Altova with complete information thereof; (ii) provide for the security of your confidential information; (iii) establish and maintain backup systems and procedures necessary to reconstruct lost or altered files, data or programs.

7. SOFTWARE ACTIVATION, UPDATES AND LICENSE METERING

(a) License Metering. The Software includes a built-in license metering module that is designed to assist you with monitoring license compliance in small local area networks (LAN). The metering module attempts to communicate with other machines on your local area network (LAN). You permit Altova to use your internal network for license monitoring for this purpose. This license metering module may be used to assist with your license compliance but should not be the sole method. Should your firewall settings block said communications, you must deploy an accurate means of monitoring usage by the end user and preventing users from using the Software more than the Permitted Number.

(b) License Compliance Monitoring. You are required to utilize a process or tool to ensure that the Permitted Number is not exceeded. Without prejudice or waiver of any potential violations of the Agreement, Altova may provide you with additional compliance tools should you be unable to accurately account for license usage within your organization. If provided with such a tool by Altova, you (a) are required to use it in order to comply with the terms of this Agreement and (b) permit Altova to use your internal network for license monitoring and metering and to generate compliance reports that are communicated to Altova from time to time.

(c) Software Activation. The Software may use your internal network and Internet connection for the purpose of transmitting license-related data at the time of installation, registration, use, or update to an Altova Master License Server and validating the

authenticity of the license-related data in order to protect Altova against unlicensed or illegal use of the Software and to improve customer service. Activation is based on the exchange of license related data between your computer and the Altova Master License Server. You agree that Altova may use these measures and you agree to follow any applicable requirements. You further agree that use of license key codes that are not or were not generated by Altova and lawfully obtained from Altova, or an authorized reseller as part of an effort to activate or use the Software violates Altova's intellectual property rights as well as the terms of this Agreement. You agree that efforts to circumvent or disable Altova's copyright protection mechanisms, the license management mechanism, or the Altova Master License Server violate Altova's intellectual property rights as well as the terms of this Agreement. Altova expressly reserves the rights to seek all available legal and equitable remedies to prevent such actions and to recover lost profits, damages and costs.

(d) **LiveUpdate.** Altova provides a new LiveUpdate notification service to you, which is free of charge. Altova may use your internal network and Internet connection for the purpose of transmitting license-related data to an Altova-operated LiveUpdate server to validate your license at appropriate intervals and determine if there is any update available for you.

(e) **Use of Data.** The terms and conditions of the Privacy Policy are set out in full at <http://www.altova.com/privacy> and are incorporated by reference into this Agreement. By your acceptance of the terms of this Agreement and/or use of the Software, you authorize the collection, use and disclosure of information collected by Altova for the purposes provided for in this Agreement and/or the Privacy Policy. Altova has the right in its sole discretion to amend this provision of the Agreement and/or Privacy Policy at any time. You are encouraged to review the terms of the Privacy Policy as posted on the Altova Web site from time to time.

(f) **Audit Rights.** You agree that Altova may audit your use of the Software for compliance with the terms of this Agreement at any time, upon reasonable notice. In the event that such audit reveals any use of the Software by you other than in full compliance with the terms of this Agreement, you shall reimburse Altova for all reasonable expenses related to such audit in addition to any other liabilities you may incur as a result of such non-compliance.

(g) **Notice to European Users.** Please note that the information as described in paragraph 7(d) above may be transferred outside of the European Economic Area, for purposes of processing, analysis, and review, by Altova, Inc., a company located in Beverly, Massachusetts, U.S.A., or its subsidiaries or Altova's subsidiaries or divisions, or authorized partners, located worldwide. You are advised that the United States uses a sectoral model of privacy protection that relies on a mix of legislation, governmental regulation, and self-regulation. You are further advised that the Council of the European Union has found that this model does not provide "adequate" privacy protections as contemplated by Article 25 of the European Union's Data Directive. (Directive 95/46/EC, 1995 O.J. (L 281) 31). Article 26 of the European Union's Data Directive allows for transfer of personal data from the European Union to a third country if the individual has unambiguously given his consent to the transfer of personal information, regardless of the third country's level of protection. By agreeing to this Agreement, you consent to the transfer of all such information to the United States and the processing of that information as described in this Agreement and the Privacy Policy.

8. TERM AND TERMINATION

This Agreement may be terminated (a) by your giving Altova written notice of termination; (b) by Altova, at its option, giving you written notice of termination if you commit a breach of this Agreement and fail to cure such breach within ten (10) days after notice from Altova; or (c) at the

request of an authorized Altova reseller in the event that you fail to make your license payment or other monies due and payable. In addition the Agreement governing your use of a previous version of the Software that you have upgraded or updated is terminated upon your acceptance of the terms and conditions of the Agreement accompanying such upgrade or update. Upon any termination of the Agreement, you must cease all use of the Software that this Agreement governs, destroy all copies then in your possession or control and take such other actions as Altova may reasonably request to ensure that no copies of the Software remain in your possession or control. The terms and conditions set forth in Sections 1(h), 1(i), 1(j), 1(k), 1(l), 2, 5, 7, 9, 10, 11, and 11 survive termination as applicable.

9. RESTRICTED RIGHTS NOTICE AND EXPORT RESTRICTIONS

The Software was developed entirely at private expense and is commercial computer software provided with **RESTRICTED RIGHTS**. Use, duplication or disclosure by the U.S. Government or a U.S. Government contractor or subcontractor is subject to the restrictions set forth in this Agreement and as provided in FAR 12.211 and 12.212 (48 C.F.R. §12.211 and 12.212) or DFARS 227. 7202 (48 C.F.R. §227-7202) as applicable. Consistent with the above as applicable, Commercial Computer Software and Commercial Computer Documentation licensed to U.S. government end users only as commercial items and only with those rights as are granted to all other end users under the terms and conditions set forth in this Agreement. Manufacturer is Altova GmbH, Rudolfplatz 13a/9, A-1010 Vienna, Austria/EU. You may not use or otherwise export or re-export the Software or Documentation except as authorized by United States law and the laws of the jurisdiction in which the Software was obtained. In particular, but without limitation, the Software or Documentation may not be exported or re-exported (i) into (or to a national or resident of) any U.S. embargoed country or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nationals or the U.S. Department of Commerce's Table of Denial Orders. By using the Software, you represent and warrant that you are not located in, under control of, or a national or resident of any such country or on any such list.

10. U.S. GOVERNMENT ENTITIES

Notwithstanding the foregoing, if you are an agency, instrumentality or department of the federal government of the United States, then this Agreement shall be governed in accordance with the laws of the United States of America, and in the absence of applicable federal law, the laws of the Commonwealth of Massachusetts will apply. Further, and notwithstanding anything to the contrary in this Agreement (including but not limited to Section 5 (Indemnification)), all claims, demands, complaints and disputes will be subject to the Contract Disputes Act (41 U.S.C. §§7101 *et seq.*), the Tucker Act (28 U.S.C. §1346(a) and §1491), or the Federal Tort Claims Act (28 U.S.C. §§1346(b), 2401-2402, 2671-2672, 2674-2680), FAR 1.601(a) and 43.102 (Contract Modifications); FAR 12.302(b), as applicable, or other applicable governing authority. For the avoidance of doubt, if you are an agency, instrumentality, or department of the federal, state or local government of the U.S. or a U.S. public and accredited educational institution, then your indemnification obligations are only applicable to the extent they would not cause you to violate any applicable law (e.g., the Anti-Deficiency Act), and you have any legally required authorization or authorizing statute.

11. THIRD PARTY SOFTWARE

The Software may contain third party software which requires notices and/or additional terms and conditions. Such required third party software notices and/or additional terms and conditions are located at our Website at http://www.altova.com/legal_3rdparty.html and are made a part of and incorporated by reference into this Agreement. By accepting this Agreement, you are also accepting the additional terms and conditions, if any, set forth therein.

12. JURISDICTION, CHOICE OF LAW, AND VENUE

If you are located in the European Union and are using the Software in the European Union and not in the United States, then this Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht, Wien (Commercial Court, Vienna) in connection with any such dispute or claim.

If you are located in the United States or are using the Software in the United States then this Agreement will be governed by and construed in accordance with the laws of the Commonwealth of Massachusetts, USA (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the federal or state courts of the Commonwealth of Massachusetts and you further agree and expressly consent to the exercise of personal jurisdiction in the federal or state courts of the Commonwealth of Massachusetts in connection with any such dispute or claim.

If you are located outside of the European Union or the United States and are not using the Software in the United States, then this Agreement will be governed by and construed in accordance with the laws of the Republic of Austria (excluding its conflict of laws principles and the U.N. Convention on Contracts for the International Sale of Goods) and you expressly agree that exclusive jurisdiction for any claim or dispute with Altova or relating in any way to your use of the Software resides in the Handelsgericht, Wien (Commercial Court, Vienna) and you further agree and expressly consent to the exercise of personal jurisdiction in the Handelsgericht Wien (Commercial Court, Vienna) in connection with any such dispute or claim. This Agreement will not be governed by the conflict of law rules of any jurisdiction or the United Nations Convention on Contracts for the International Sale of Goods, the application of which is expressly excluded.

13. TRANSLATIONS

Where Altova has provided you with a foreign translation of the English language version, you agree that the translation is provided for your convenience only and that the English language version will control. If there is any contradiction between the English language version and a translation, then the English language version shall take precedence.

14. GENERAL PROVISIONS

This Agreement contains the entire agreement and understanding of the parties with respect to the subject matter hereof, and supersedes all prior written and oral understandings of the parties with respect to the subject matter hereof. Any notice or other communication given under this Agreement shall be in writing and shall have been properly given by either of us to the other if sent by certified or registered mail, return receipt requested, or by overnight courier to the address shown on Altova's Web site for Altova and the address shown in Altova's records for you, or such other address as the parties may designate by notice given in the manner set forth above. This Agreement will bind and inure to the benefit of the parties and our respective heirs, personal and legal representatives, affiliates, successors and permitted assigns. The failure of either of us at any time to require performance of any provision hereof shall in no manner affect such party's right at a later time to enforce the same or any other term of this Agreement. This Agreement may be amended only by a document in writing signed by both of us. In the event of a breach or

threatened breach of this Agreement by either party, the other shall have all applicable equitable as well as legal remedies. Each party is duly authorized and empowered to enter into and perform this Agreement. If, for any reason, any provision of this Agreement is held invalid or otherwise unenforceable, such invalidity or unenforceability shall not affect the remainder of this Agreement, and this Agreement shall continue in full force and effect to the fullest extent allowed by law. The parties knowingly and expressly consent to the foregoing terms and conditions.

Last updated: 2015/09/03

Index

▪

.NET Framework,

Include file, 149

2

2-way project,

merge, 152

3

3-way project,manual merge, 155
merge, 153

A

Absolute,

and relative links, 198

Abstract,

class, 22

Activation box,

Execution Specification, 261

Activity,Add diagram to transition, 226
Add operation, 226
Add to state, 226
BPMN, 348
create branch / merge, 213
diagram elements, 215
diagram SysML, 383
icons, 557**Activity diagram, 210**

inserting elements, 211

Actor, 14

customize, 14

user-defined, 14

Add, 14, 535diagram to package, 14
insert - delete from Model Tree, 65
Model link, 102
move - delete - diagram, 91
new project, 116
package to project, 14
project to source control, 535
to Favorites, 72
to source control, 535**ADO,**as data connection interface, 392
setting up a connection, 398**Align,**elements when dragging, 14
snap lines when dragging, 613**Aligning,**

elements, 95

All,

expand / collapse, 294

Altova Scripting Projects, 651**Annotation,**documentation, 82
text - BPMN, 358
XML schema, 335**API,**documentation, 710
overview, 711**Application,**

external - arguments, 607

Application Events, 668**Application object, 711****Arguments,**

external tools, 607

Artifact, 44add to node, 44
BPMN, 358
manifest, 44**ASPRJ files, 651****Assign,**shortcut to a command, 610
stereotype, 322**Association, 91**aggregate/composite, 22
automatic display of, 188
between classes, 22
BPMN, 356

Association, 91

- class memberEnd, 188
- create by drag and drop, 193
- defining the type, 188
- line thickness, 91
- object links, 34
- qualifier, 188
- role, 188
- Show property as, 91
- Show relationships, 91, 191
- show typed property, 165
- use case, 14

Attribut,

- select in Model Tree, 74

Attribute, 322

- autocompletion window, 613
- coloring, 300
- show / hide, 294
- show / hide tagged values, 322
- stereotype, 322

Authentic View Events, 668**Autocomplete,**

- function, 22

Autocompletion, 95

- single / multi mode, 95
- window - types, 95
- window on class editing, 613

Autogenerate,

- reply message, 266

Auto-Macro setting, 669**Automatically add operation, 226****Automatic,**

- display of associations, 188
- hyperlink, 102

Automatically,

- update transformation, 185

- class, 29

Base class,

- expand, collapse compartments, 294
- inserting derived, 100
- multiple instances on diagram, 294
- overriding, 294

Batch, 114

- full batch mode, 114
- new / load / save, 114
- processing, 110, 114

Behavioral,

- diagrams, 209

Behaviour,

- diagrams SysML, 383

Binary files,

- importing C# and Java, 129

Binding,

- template, 164

Bitmap,

- save elements as, 584

Block definition diagram,

- SysML, 372

Block properties,

- as nodes, 372

Borland,

- bsdj project file, 587

BPMN, 347

- artifacts, 358
- association, 356
- convert 1.0 to 2.0, 347
- flow objects, 348

Branch,

- create in Activity, 213

bsdj,

- Borland project, 587

Business Process Modeling Notation, 347

- icons, 573

B

Background,

- transparent color, 330

Ball and socket,

- interface notation, 294

Bank,

- sample files, 107

Base,

C

C#,

- code, 622
- code to model correspondence, 145
- error handling, 726
- import binary file, 129
- import settings, 124

- C++**,
 - code, 622
 - error handling, 726
- Call**,
 - message, 266
- Call message**,
 - go to operation, 266
- CallBehavior**,
 - insert, 211
- CallOperation**,
 - insert, 211
- Cascading**,
 - styles, 76
- Catalog**,
 - file - XMLSpy Catalog file, 613
- Center**,
 - elements, 95
- Change provider**,
 - source control, 547
- Change scripts**,
 - updating databases, 454
- Check**,
 - project syntax, 587
- Check In, 533**
- Check Out, 531**
- Class, 29, 119**
 - abstract and concrete, 22
 - add new, 22
 - add operations, 22
 - add properties, 22
 - associations, 22
 - base, 29
 - create in namespace, 119
 - derived, 29
 - diagrams, 22
 - enable autocompletion window, 613
 - generate in namespace, 119
 - icons, 558
 - in component diagram, 39
 - inserting derived classes, 100
 - name changes - synchronization, 140
 - synchronization, 135
 - syntax coloring, 300
- Class diagram, 294**
- Class name changing**,
 - effect on code file name, 140
- Classifier**,
 - constraining, 161
 - new, 138
 - renaming, 138
- Close**,
 - all but active diagram, 91
- Code, 140**
 - adding code to sequence diagram, 281
 - default, 613
 - generate from sequence diagram, 278
 - generate multiple sequence diagrams from, 275
 - generate sequence diagram from, 271, 716
 - generate sequence diagram manually, 717
 - generation - min. conditions, 142
 - Java code and class file names, 140
 - prerequisites, 48
 - refactoring, 140
 - round trip engineering, 48
 - SPL, 623
 - synchronization, 135
 - target directory, 48
- Code - C#**,
 - to UModel elements, 145
- Code - Java**,
 - to UModel elements, 144
- Code - VB.NET**,
 - to UModel elements, 147
- Code - XML Schema**,
 - to UModel elements, 146
- Code engineering, 48**
 - generate ComponentRealizations, 138
 - import directory, 54
 - minimalist project, 119
 - move project file to new location, 116
- Code generation**,
 - use namespace as directory, 48
- Code Generator, 622**
- Collaboration**,
 - Composite Structre diagram, 309
- Collapse**,
 - class compartments, 294
- Collapsed**,
 - sub process, 354
- Color**,
 - syntax coloring - enable/disable, 300
 - transparent background, 330
- Column**,
 - adding to model, 454
- COM-API**,
 - documentation, 710

Combined fragment, 262**Command, 114, 610**

- add to toolbar/menu, 606
- context menu, 610
- delete from menu, 610
- line - new / load / save, 114
- line processing, 110, 114
- reset menu, 610

Comments,

- documentation, 82
- show source code comments in diagram, 91

Communication,

- icons, 559

Communication diagram, 251

- generate from Sequence diagram, 251

Compact,

- mode - toggle, 335

Compare source files, 542**Compartment,**

- expand single / multiple, 294

Compatibility,

- updating projects, 135

Component, 39

- diagram, 39
- icons, 561
- insert class, 39
- realization, 39

Component diagram, 311**ComponentRealizations,**

- autogeneration, 138

Composite state, 233

- add region, 233

Composite Structure,

- icons, 560
- insert elements, 309

Composite Structure diagram, 309**Composition,**

- association - create, 22

Concrete,

- class, 22

Conditional flow, 356**Configure,**

- XMLSPY UI, 699

conjugated port,

- create atomic, 375

Connecting objects, 356**Connection wizard,**

- importing databases, 446

Constrain,

- element, 65

Constraining,

- classifiers, 161

Constraint, 65

- add in diagram, 65
- assign to multiple element, 65
- defining parameters, 378
- syntax check, 587

Content model,

- of XML Schema, 341

Context menu,

- commands, 610

Continuous flow,

- modeling / streaming SysML Activity, 383

Convert,

- BPMN 1.0 to 2.0, 347
- C# to Java - model to model, 177
- Java to C# transformation, 171

Copy,

- paste in Diagram, Model Tree, 97

Copyright information, 1236**CR/LF,**

- for ump file on save, 116

Create,

- getter / setter methods, 294
- XML schema, 345

csproj - csdproj,

- MS Visual Studio .Net, 587

CTRL+Space,

- on-demand autocompletion, 95

Custom,

- stereotype icon, 330

Customize, 606, 610

- actor, 14
- context menu, 610
- menu, 610
- toolbar/menu commands, 606

D

Data object,

- BPMN, 358

Database, 450, 454

- add new table, 457
- adding column to model, 454

Database, 450, 454

- adding Index to model, 450
- change scripts on update, 454
- creating ForeignKey classes, 450
- importing into UModel, 390
- syntax checks, 450
- types supported, 446
- writing model to code / database, 454

Database connection,

- setting up, 392
- setup examples, 411
- starting the wizard, 394

Database drivers,

- overview, 395

Datatype,

- defining in Schema, 341

Datatypes,

- adding - autocompletion, 95

Debugging macros, 674**Default,**

- layer, 84
- menu, 610
- path - examples folder, 8
- project code, 613
- SPL templates, 135
- tagged value, 322

Default flow, 356**Defined,**

- symbols - importing code, 124

Delete,

- class relationships, 188
- command from context menu, 610
- command from toolbar, 606
- from Favorites, 72
- icon from toolbar, 606
- layer, 84
- shortcut, 610
- toolbar, 606

Dependency,

- include, 14
- Show relationships, 91, 191
- usage, 39

Deployment,

- diagram, 44
- icons, 562

Deployment diagram, 312**Derived,**

- class, 29

- classes inserting, 100

Diagram, 91, 314, 613

- Activity, 210
- BPMN, 347
- Communication, 251
- Component, 311
- Composite structure, 309
- Deployment, 312
- Interaction Overview, 254
- Object, 313
- Package, 314
- Profile, 320
- Sequence, 259
- State machine, 225
- Timing, 285
- Use Case, 250
- XML schema, 335
- Add activity to transition, 226
- adding code to sequence diagram, 281
- Additional - XML schema, 334
- behavior SysML, 383
- Class, 294
- close all but active, 91
- constrain elements, 65
- database - importing, 390
- generate code from sequence diagram, 278
- generate multiple sequence diagrams from, 275
- generate Package dependency diagram, 314
- generate sequence diagram from, 271
- generate sequence diagram from getter/setter, 275
- hyperlink, 102
- icons, 556
- ignore elem. from included files, 613
- insert SysML, 371
- line weight, 91
- multiple instances of class, 294
- open, 70
- Paste in Diagram only, 97
- Profile and stereotypes, 322
- properties, 91
- save as png, 579
- save elements as bitmap, 584
- save open diagrams with project, 613
- Sequence diagrams SysML, 385
- share package and diagram, 158
- show source code comments, 91
- sizing, 91
- State Machine SysML, 386

Diagram, 91, 314, 613

- styles, 76
- SysML internal block, 375
- SysML structure, 371
- Use case SysML, 387
- using layers, 84
- XML schema - import, 335

Diagram - sequence,

- generate from code, 716
- generate manually from code, 717
- generate sequence diagram from code, 717

Diagram frame,

- show UML diagram heading, 95

Diagram heading,

- show UML diagram heading, 95

Diagram pane, 91**Diagram Tree, 70****Diagrams, 208**

- behavioral, 209
- structural, 293

Directive,

- defined symbols, 124

Directory,

- change project location, 116
- examples folder, 8
- for code generation, 48
- ignoring on merge, 613
- import, 54
- importing code from, 124
- use namespace in path, 48

Disable source control, 527**Distribution,**

- of Altova's software products, 1236, 1237, 1239

Document,

- hyperlink to, 102

Document Events, 668**Documentation, 198**

- Annotation, 82
- generate UML project, 198
- relative links, 198

Documentation tab, 82**Dot,**

- Ownership, 193

Download source control project, 524**Drag and drop,**

- create association with handles, 193
- right mouse button, 100

Drid,

- snap lines while dragging, 14

DurationConstraint,

- Timing diagram, 289

E

Edit, 584**Element, 65**

- add to Favorites, 72
- align, 95
- assign constraint to, 65
- constrain, 65
- cut, copy paste, 97
- generate documentation, 198
- hyperlink to, 102
- inserting, 100
- move to layer, 84
- properties, 74
- relationships, 188
- save selected as bitmap, 584
- show hierarchy, 79
- styles, 76

Elements,

- ignore from include files, 613
- insert State Machine, 226
- select all on layer, 84

Enable source control, 527**End User License Agreement, 1236, 1240****Enhance,**

- performance, 167

Entry point,

- add to submachine, 233

Enumeration, 326

- and stereotypes, 326
- default value, 326

Error,

- messages, 90
- syntax check, 48

Error handling,

- general description, 726

Evaluation period,

- of Altova's software products, 1236, 1237, 1239

Event,

- BPMN, 348

Event handlers,

- in Scripting Project, 668

Event handlers,

- overview, 657

Event/Stimulus,

- Timing diagram, 289

Events, 668

- and event handlers, 659

Examples,

- tutorial folder, 8

Exception,

- Adding raised exception, 294
- Java operation, 124

Execute,

- change script on database, 454

Execution specification,

- lifeline, 261

Exit point,

- add to submachine, 233

Expand,

- all class compartments, 294
- collapsing packages, 65

Expanded,

- sub process, 353

Export,

- as XMI, 464

Extension,

- XMI, 464

F

Favorites, 72**Fetch file,**

- source control, 528

Field,

- adding field/column to model, 454

File, 579

- merging project files, 151
- new / load / save - batch, 114
- open from URL, 579
- tutorial example, 8
- ump, 116

File DSN,

- setting up, 404

Files,

- sample files, 107

Filter types,

- autocompletion window, 95

Find, 65

- modeling elements, 65, 584
- searching tabs, 64
- unused elements, 65

Firebird,

- Connecting through ODBC, 411

Flow, 356

- conditional, 356
- default, 356
- message, 356
- sequence, 356

Flow objects, 348**Flow port, 375**

- create atomic conjugated, 375
- defining the direction of, 375
- inserting a, 375

Flows,

- item flow creating, 375

Folder,

- examples folder, 8

Folders,

- get in source control, 530

ForeignKey,

- creating in database diagram, 450

Form Object Palette, 652**Form Object properties, 663****Format,**

- autocompletion window, 95
- user-defined stereotype, 328

Forms,

- and built-in commands, 677
- and event handling, 665
- and Form Objects, 663
- creating new, 662
- in Scripting Projects, 662
- invocation of, 659
- naming, 662
- overview, 657
- properties of, 662
- setting tab sequence of objects, 663

Forward,

- engineering, 142

Frame,

- show UML diagram heading, 95

Full batch,

- mode, 114

G

- Gate,**
 - sequence diagram, 265
- Gateway,**
 - BPMN, 348
- General Value lifeline,**
 - Timing diagram, 286
- Generalize,**
 - specialize, 29
- Generate,**
 - code from schema, 622
 - ComponentRealizations automatically, 138
 - multiple sequence diagrams from code, 275
 - reply message automatically, 266
 - Sequence dia from Communication, 251
 - sequence diagram from code, 271, 716
 - UML project documentation, 198
 - XML Schema, 345
- Generate manually,**
 - sequence diagram from code, 717
- Get,**
 - getter / setter methods, 294
- Get file,**
 - source control, 529
- Get folders,**
 - source control, 530
- Get latest version, 528**
- Getter / Setter,**
 - generate sequence diagram from, 275
- Global declarations, 661**
 - overview, 657
- Global scripting project,**
 - of UModel, 651
- Goto,**
 - lifeline, 261
- Graph view,**
 - single set of relations, 79
- Grid,**
 - show- snap to, 91
 - snap lines, 613
- Grid View Events, 668**
- Group,**
 - BPMN, 358

H

- Handle,**
 - create relationship, 193
 - Heading,**
 - show UML diagram heading, 95
 - Help, 620**
 - Hide,**
 - show - slot, 294
 - tagged values - attributes, 322
 - text labels, 91, 191
 - view - layer, 84
 - Hierarchy,**
 - show all relations, 79
 - Hierarchy diagram,**
 - levels shown in documentation, 198
 - History,**
 - show, 540
 - Hotkey, 610**
 - HRESULT,**
 - and error handling, 726
 - Hyperlink, 102**
 - automatic, 102
-
- IBM DB2,**
 - connecting through ODBC, 415
 - IBM DB2 for i,**
 - connecting through ODBC, 421
 - IBM Informix,**
 - connecting through JDBC, 424
 - Icon,**
 - Activity, 557
 - add to toolbar/menu, 606
 - Business Process Modeling Notation, 573
 - class, 558
 - Communication, 559
 - component, 561
 - Composite Structure, 560
 - custom stereotype icon, 330
 - deployment, 562
 - Interaction Overview, 563

Icon,

- object, 564
- Package, 565
- Sequence, 568
- show large, 611
- State machine, 569
- Timing, 570
- use case, 571
- XML Schema, 572

Icons,

- visibility, 294

ID,

- IDs and UUIDs, 464

Ignore, 613

- directories, 613
- elements in list, 613
- operation names, 271

Import, 124

- binary files, 129
- C# project, 124
- directory, 54
- project, 124
- relative to UMP file, 124
- source code, 124
- source project, 54
- SQL database, 390
- XMI file, 464
- XML Schema, 335

Importing,

- UModel generated XMI, 464

Include, 158

- .NET Framework, 149
- dependency, 14
- share package and diagram, 158
- status - changing, 158
- UModel project, 149

Including,

- subprojects back into main, 469

Index,

- adding to table, 450

Initial values,

- add to property - SysML, 375

Insert, 100, 211

- action (CallBehavior), 211
- action (CallOperation), 211
- Composite Structure elements, 309
- elements, 100
- Interaction Overview elements, 255

Package diagram elements, 315

simple state, 226

Timing diagram elements, 286

with..., 100

Installation,

- examples folder, 8

Installer,

- multi-user, 8

Instance, 34

- diagram, 34
- multiple class, and display of, 294
- object, 34

Integrating,

- subprojects into main, 469

Intelligent,

- autocomplete, 22

Interaction operand, 262

- multi-line, 262

Interaction operator,

- defining, 262

Interaction Overview,

- icons, 563
- inserting elements, 255

Interaction Overview diagram, 254**Interaction use, 265****Internal block diagram,**

- SysML, 375

Introduction, 6**Item flow,**

- creating, 375

J

Java,

- code, 622
- code and class file names, 140
- code to model correspondence, 144
- exception, 124
- import binary file, 129
- namespace root, 142

JavaDocs, 82**JavaScript,**

- error handling, 726

JDBC,

- as data connection interface, 392
- setting up a connection (Windows), 407

K

Keyboard shortcut, 610

L

Label,

- IDs and UUIDs, 464
- show / hide text label, 91, 191

Language,

- scripting language - changing, 659

Layer, 84

- add to diagram, 84
- delete, 84
- lock element, 84
- reset, 84

Layout, 594

Legal information, 1236

License, 1240

- information about, 1236

License metering,

- in Altova products, 1238

Lifeline, 261

- attributes, 261
- General Value, 286
- typed property as, 261

Lifeline,

- goto, 261

Limit,

- constrain elements, 65

Line,

- orthogonal, 39
- weight/thickness in diagrams, 91

Line break,

- in actor text, 14

Line up,

- elements, 95

Lines,

- formatting, 34
- snap lines, 613

Link, 102

- create hyperlink, 102
- Model Tree element, 102

Links,

- relative to documentation, 198

List,

- unused elements, 65

Local files,

- absolute or relative links, 198

Local project, 524

Location,

- move project, 116

Lock,

- elements on layer, 84
- toggle on layer, 88

M

Macros,

- creating with Scripting Editor, 669
- debugging, 674
- editing with Scripting Editor, 669
- execution of, 659
- functions for, in Global Declarations, 661
- how to use in Scripting Project, 669
- overview, 657
- running, 673
- setting as Auto-Macro in Scripting Editor, 669

Mail,

- send project, 579

Make same,

- height - width - size, 95

Manifest,

- artifact, 44

Mapping,

- C# to/from model elements, 145
- Java to/from model elements, 144
- VB.NET to/from model elements, 147
- XML Schema to/from model elements, 146

Mark,

- elements on layer, 84

Member end,

- stereotype, 322

MemberEnd,

- association, 188

Menu, 610

- Add menu to, 607
- add/delete command, 606
- customize, 610

Menu, 610

- Default/XMLSPY, 610
- delete commands from, 610
- edit, 584
- file, 579
- help, 620
- layout, 594
- project, 587
- tools, 596
- view, 595
- window, 619

Merge,

- 2-way project merge, 152
- 3-way manual project merge, 155
- 3-way project merge, 153
- code from model, 48
- code into model, 587
- create in Activity, 213
- ignore directory, 613
- model into code, 587
- projects, 151

Merge to,

- layer, 88

Message, 266

- arrows, 266
- call, 266
- create object, 266
- go to operation, 266
- inserting, 266
- moving, 266
- numbering, 266
- Timing diagram, 291

Message flow, 356**Messages pane, 90****Metadata,**

- XMI output, 464

Method,

- Add raised exception, 294
- generate multiple sequence diagram from, 275
- generate sequence diagram from, 271
- generate sequence diagram from getter/setter, 275

Methods,

- getter / setter, 294

Microsoft Access,

- connecting through ADO, 398, 425

Microsoft SQL Server,

- connecting through ADO, 428
- connecting through ODBC, 431

Minimalist,

- project - code engineering, 119

Minimum,

- code generation conditions, 142

Missing elements,

- listing, 65

Model,

- automatically update transformation, 185
- changing class name - effect in Java, 140
- link, add, 102
- transforming - model to model, 177
- transforming models, 171

Model Tree, 65

- create hyperlink to element in..., 102
- opening packages, 65
- pane, 65
- select attribute in, 74

Modeling,

- databases, 446
- enhance performance, 167

Mouse,

- copy, paste, 97

Move,

- element to layer, 84
- project, 116

Moving message arrows, 266**MS Visual Studio .Net,**

- csproj - csdproj project file, 587

Multi mode,

- autocompletion, 95

Multiline, 14**Multi-line,**

- actor text, 14
- interactionOperand, 262
- use case, 14

Multiple elements,

- styles display, 76

Multi-user,

- examples folder, 8

MyDocuments,

- example files, 8

MySQL,

- connecting through ODBC, 433

N

Name,

- region names - hide / show, 233

Namespace,

- creating a class in, 119
- Java namespace root, 142
- use for code generation, 48

Navigate,

- hyperlink, 102

New,

- classifier, 138

New line,

- in Lifeline, 251
- interactionOperand, 262

Node, 44

- add, 44
- add artifact, 44
- block properties as, 372
- styles, 76

Note,

- hyperlink from, 102

Numbering,

- messages, 266

O

Object, 34

- create message, 266
- diagram, 34
- icons, 564
- links - associations, 34

Object diagram, 313**Object model,**

- overview, 711

ODBC,

- as data connection interface, 392
- setting up a connection, 404

ODBC Drivers,

- checking availability of, 404

OLE DB,

- as data connection interface, 392

On-demand autocompletion,

- CTRL+Space, 95

Open,

- diagram, 70
- packages in tree view, 65
- URL, 579

Open Project,

- source control, 524

Operand,

- interaction, 262

Operation,

- autocompletion window, 613
- Automatically add on Activity, 226
- coloring, 300
- exception, 124
- goto from call message, 266
- ignore when generating sequence diagram, 271
- overriding, 294
- reusing, 29
- show / hide, 294
- template, 165

Operations,

- adding, 22

Operator,

- interaction, 262

Options, 613

- project, 166
- source control, 613
- tools, 613

Oracle database,

- connecting through ODBC, 436

Orthogonal,

- line, 39
- state, 233

Output,

- XMI file, 464

Override,

- class operations, 294
- default SPL templates, 135

Overview,

- of UModel API, 711

Overview pane, 82**Overwrite,**

- code from model, 587
- model from code, 587

OwnedEnd,

- association, 188

Ownership,

- dot, 193

P

Package,

- create as namespace package, 119
- expand/collapse, 65
- icons, 565
- profile, 322
- sharing, 158

Package diagram, 314

- generating dependency diagram, 314
- insert elements, 315
- SysML, 380

PackageImport, 315**PackageMerge, 315****Page,**

- prevent split over pages, 579

Parameter,

- batch, 110
- template, 165

Parametric, 378

- constraints - defining, 378
- diagram SysML, 378

Partial,

- documentation - generate, 198

Paste,

- element in diagram, 97
- in Diagram only, 97

Path,

- change project location, 116
- examples folder, 8
- import relative to UMP file, 124
- SPL template path, 625
- use namespace in code, 48

Performance,

- enhancement, 167

Plugin,

- register - no need to, 696

Plug-in,

- registration, 697
- User interface configuration, 699
- XMLSPY, 689

PNG,

- save diagram, 579

Pool,

- swimlane, 358

Port, 375

- create atomic conjugated flow port, 375
- create flow port, 375
- inserting a flow port, 375
- joining two ports, 375

PostgreSQL,

- connecting through ODBC, 441

Prerequisites,

- forward engineering, 142

Pretty print,

- project on save, 116
- XMI output, 464

Print,

- preview, 579

Process,

- collapsed sub process, 354
- expanded sub process, 353

Profile, 320, 322

- diagram, 320
- Application, 320
- diagram, 320, 322
- stereotypes, 322

Programming points,

- in Scripting Project, 676

Project, 116, 587, 613

- 2-way merge, 152
- 3-way manual merge, 155
- 3-way merge, 153
- add to source control, 535
- create, 116
- create namespace package, 119
- creating subprojects, 469
- default code, 613
- file - updating, 135
- generating documentation, 198
- import, 124
- include UModel project, 149
- insert package, 116
- Merge, 151
- move, 116
- open last on start, 613
- options, 166
- remove from source control, 538
- save - pretty print, 116
- save open diagrams, 613
- send by mail, 579
- styles, 76
- syntax checking, 587

Project, 116, 587, 613

workflow, 116

Project files,

Borland - MS Visual Studio .Net, 587

Project open,

source control, 524

Properties, 74

adding, 22

source control, 544

Properties and Events pane, 652**Property,**

coloring, 300

create initial values, 375

reusing, 29

show as association, 91, 191

typed - show, 165

typed as lifeline, 261

Provider,

select, 524

Q

Qualifier,

association, 188

R

Raised exception, 124

Adding, 294

Realization,

component, 39

generate ComponentRealizations, 138

Refactoring code,

class names - synchronization, 140

Reference, 578

show referenced class, 91

Refresh status,

source control, 545

Region,

add to composite state, 233

Region name,

show / hide, 233

Register,

plug-in, 697

Reject source edits, 534**Relation,**

show all - hierarchy tab, 79

Relationship,

Show model relationships, 91, 191

Relationships,

create using handles, 193

creating between tables, 457

element, 188

Relative,

documentation links, 198

importing/saving of files, 124

Remove,

from Favorites, 72

from source control, 538

Rename,

classifier, 138

layer, 88

Reply,

message - autogenerate, 266

Requirement diagram,

SysML, 382

Reset,

Layer, 84

menu commands, 610

shortcut, 610

toolbar & menu commands, 606

Restore,

toolbars and windows, 596

Reuse,

of subprojects, 469

Right dragging, 100**Role,**

association, 188

Root,

catalog - XMLSpy, 613

Java namespace, 142

package/class synchronization, 135

Round trip,

code - model -code, 54

engineering, 48

model - code - model, 48

Run native interface, 546

S

Sample,

- example files, 107

Save,

- diagram as image, 579
- elements as bitmaps, 584
- subproject files, 469

SC,

- syntax coloring, 300

Schema,

- code generator, 622
- create XML Schema, 345
- Datatype - defining, 341
- XML Schema, 335
- XML Schema - import, 335

Scripting Editor,

- GUI description, 652
- Main Window, 652

Scripting Environment, 648

- usage overview, 650

Scripting language, 659**Scripting Project,**

- and Events, 668
- application event handlers, 668
- Event Handlers, 657
- Forms, 657
- Forms in, 662
- Global Declarations, 657
- Global Declarations in, 661
- Macros, 657
- Macros in, 669
- programming points, 676
- steps for creating, 659

Scripting Project Tree pane, 652**Scripting Projects,**

- for UModel, 651
- for UModel Project, 651

Search,

- Find, 584

Searching tabs, 64**Select all,**

- elements on layer, 84

Select attribute,

- in Model Tree, 74

Send by mail,

- project, 579

Sequence,

- diagram SysML, 385
- icons, 568

Sequence diagram, 259

- adding code to, 281
- combined fragment, 262
- gate, 265
- generate code from, 278
- generate from code, 716
- generate from Communication diag., 251
- generate manually from code, 717
- inserting elements, 260
- interaction use, 265
- lifeline, 261
- messages, 266
- operation names to ignore, 271
- state invariant, 266

Sequence flow, 356**Set,**

- getter / setter methods, 294

Setter / Getter,

- generate sequence diagram from, 275

Setting,

- synchronization, 135

Settings,

- source control, 613

Share,

- from source control, 539
- package and diagram, 158

Shortcut, 610

- assigning/deleting, 610
- show in tooltip, 611

Show, 79, 91, 191

- all relations - hierarchy tab, 79
- graph view, 79
- hide - slot, 294
- hide - tagged values / attributes, 322
- hide- region name, 233
- layer, 84
- model relationships, 91, 191
- or snap to grid, 91
- property as association, 91, 165
- tagged values, 335, 339
- text labels, 91, 191

Show differences, 542**Show history, 540**

- Show/hide,**
 - attributes, operations, 294
- Signature,**
 - template, 161, 163
- Single mode,**
 - autocompletion, 95
- Size,**
 - diagram pane, 91
- Slot,**
 - show / hide, 294
- Snap,**
 - line - when dragging, 613
 - to grid - show grid, 91
- Snap lines, 14**
- Socket,**
 - Ball and socket, 294
- Software product license, 1240**
- Sort,**
 - diagram, 70
 - elements in Model Tree, 65
- Source,**
 - code comments - show on diagram, 91
- Source code,**
 - importing, 124
- Source control,**
 - add to source control, 535
 - change provider, 547
 - Check In, 533
 - Check Out, 531
 - commands, 523
 - enable / disable, 527
 - get file, 529
 - get latest version, 528
 - installing a source-control plug-in, 518
 - open project, 524
 - options / settings, 613
 - properties, 544
 - refresh status, 545
 - remove from, 538
 - run native interface, 546
 - show differences, 542
 - show history, 540
 - subprojects - teamwork, 468
 - Undo Check out, 534
- Space,**
 - across, 95
- Specialize,**
 - generalize, 29
- Speed,**
 - enhancement, 167
- SPL, 623**
 - code blocks, 624
 - conditions, 634
 - foreach, 635
 - subroutines, 637
 - templates user-defined, 135
- SPL templates,**
 - template path, 625
- Split,**
 - prevent split over pages, 579
- SQL,**
 - importing into UModel, 390
- SQL Server,**
 - connecting through ADO, 398
- SQLite,**
 - setting up a connection (Windows), 410
- Start,**
 - UModel, 10
 - with previous project, 613
- State, 233**
 - add activity, 226
 - composite, 233
 - define transition between, 226
 - insert simple, 226
 - orthogonal, 233
 - submachine state, 233
- State changes,**
 - defining on a timeline, 286
- State invariant, 266**
- State machine,**
 - composite states, regions, 233
 - diagram elements, 243
 - diagram SysML, 386
 - icons, 569
 - insert elements, 226
 - states, activities, transitions, 226
- State Machine Diagram, 225**
- Stereotype, 322, 326**
 - and enumeration, 326
 - assigning, 322
 - attributes - defining, 322
 - custom icon, 330
 - default tagged value, 326
 - member end, 322
 - profiles, 322
 - user-defined styles, 328

Stereotypes,

- add to Profile diagram, 322
- definition of, 320

Structural,

- diagrams, 293

Structure,

- diagram SysML, 371

Style,

- user-defined stereotype, 328

Styles, 76

- cascading precedence, 76
- multiple selections, 76

Styles tab, 76**Sub class,**

- inserting into diagram, 100

Sub Process,

- collapsed, 354
- expanded, 353

Submachine state,

- add entry/exit point, 233

Subproject, 468, 469

- and source control, 468
- creating, 469
- opening / editing, 469
- saving subproject files, 469
- teamwork support, 468

Swimlane,

- pool, 358

Sybase,

- connecting through JDBC, 443

Symbols,

- defined - importing code, 124
- visibillity icons, 294

Synchronization, 140

- class and code file name, 140
- class name changes, 140
- settings, 135

Synchronize,

- database model with database, 454
- merge code from model, 48
- merge model from code, 54
- root/package/class, 135
- to new location, 116

Syntax,

- batch file, 110
- check - messages, 90
- check of database elements, 450
- check project syntax, 587

- checking, 48

- errors - warnings, 48

Syntax coloring, 300**SysML,**

- activity diagram, 383
- behavior diagrams, 383
- Block Definition Diagram, 372
- inserting a SysML diagram, 371
- Internal block diagram, 375
- introduction, 371
- package diagram, 380
- parametric diagram, 378
- Requirement diagram, 382
- Sequence diagram, 385
- State Machine diagram, 386
- structure diagram, 371
- Use Case diagram, 387

System DSN,

- setting up, 404

System hierarchy,

- Package diagram SysML, 380

T

Table, 457

- add new to model, 457
- relationships - creating, 457

Table association,

- creating relationships, 457

Tagged, 322

- definitions - values, 320
- value - default, 326
- value - hide/show attributes, 322
- values, 322

Tagged values,

- displaying, 335
- show, 339

Teamwork,

- support - subprojects, 468

Template,

- binding, 164
- operation/parameter, 165
- signature, 161, 163

Templates,

- SPL templates, 625
- user-defined SPL, 135

Text annotation,

BPMN, 358

Text labels,

show - hide, 91, 191

Text View Events, 668**Tick mark,**

Timing diagram, 288

TimeConstraint,

Timing diagram, 290

Timeline,

defining state changes, 286

Timing,

icons, 570

Timing diagram, 285, 286

DurationConstraint, 289

Event/Stimulus, 289

General Value lifeline, 286

inserting elements, 286

Lifeline, 286

Message, 291

switch between types, 286

Tick mark, 288

TimeConstraint, 290

Timeline, 286

Toggle,

compact mode, 335

lock on layer, 88

Toggle visibility,

elements on layer, 88

Toolbar,

activate/deactivate, 606

add command to, 606

create new, 606

reset toolbar & menu commands, 606

restore to default, 596

show large icons, 611

Tools, 596

Add to Tools menu, 607

options, 613

Tooltip, 611

show, 611

show shortcuts in, 611

Transformation,

automatically update, 185

C# to Java, 177

Java to C#, 171

settings, 184

Transforming models, 170**Transition,**

Add Activity diagram to, 226

define between states, 226

define trigger, 226

transparent,

background color, 330

Traverse,

hyperlinks, 102

Trigger,

define transition trigger, 226

Tutorial, 8

aims, 8

example files, 8

examples folder, 8

Type,

property - show, 165

Typed,

property - as lifeline, 261

Types,

and autocompletion, 95

U

UML,

diagram - sharing, 158

diagram heading - show, 95

Diagrams, 208

templates, 161

variables, 625

visibility icons, 294

UModel,

API, 710

importing generated XMI, 464

starting, 10

to C# code, 145

to Java code, 144

to VB.NET code, 147

to XML Schema code, 146

UModel API, 710

overview, 711

UModel diagram icons, 556**UModel Introduction, 6****UModel Plug-in for Visual Studio,**

installing, 478

Ump, 116

change project location, 116

Ump, 116

file extension, 116

Undo Check out, 534**Unlock,**

elements on layer, 88

Unused elements,

listing, 65

Update,

model to database, 454

project file, 135

transformation automatically, 185

URL,

open file from, 579

Usage,

dependency, 39

Use case, 14

adding, 14

association, 14

compartments, 14

diagram SysML, 387

icons, 571

multi-line, 14

Use Case diagram, 250**User,**

multi-user examples folder, 8

User defined,

actor, 14

User DSN,

setting up, 404

User interface, 64

configure using plug-in, 699

User-defined,

SPL templates, 135

stereotype styles, 328

UUID,

Universal Unique identifiers, 464

V

value,

tagged, 322

tagged, show, 339

Values,

tagged, 320

Tagged, displaying, 335

Variables,

external tool arguments, 607

UML, 625

VB.NET,

code to model correspondence, 147

Version control,

commands, 523

View, 595

hide - layer, 84

to multiple instances of element, 294

Viewpoints,

Package diagram SysML, 380

Visibility,

icons - selecting, 294

Visual Basic,

error handling, 726

Visual Studio,

add in UModel support to solutions, 480

automatic synchronization of code and model, 485

Integrating UModel as a plug-in, 476

loading/unloading UModel projects, 484

synchronizing code with model, 485

W

Warning,

messages, 90

syntax check, 48

Web,

hyperlink, 102

Weight,

Line thickness, 91

Window, 619

restore to default, 596

Workflow,

project, 116

Working directory,

source control, 524

X

XMI, 464

extensions, 464

pretty print output, 464

XML Schema, 341

XML Schema, 341

- annotation, 335
- code to model correspondence, 146
- create/generate, 345
- diagram, 335
- icons, 572

XML schema - insert element,

- Content model, 341

XMLSPY,

- plug-in registration, 697

XMLSPY plug-in, 689

Z

Zoom,

- sizing, 91