



User's Guide

Proprietary Notice

The software described in this document is a proprietary product of Indigo Rose Software Design Corporation and is furnished to the user under a license for use as specified in the license agreement.

The software may be used or copied only in accordance with the terms of the agreement.

Information in this document is subject to change without notice and does not represent a commitment on the part of Indigo Rose Software Design Corporation. No part of this document may be reproduced, transmitted, transcribed, stored in any retrieval system, or translated into any language without the express written permission of Indigo Rose Software Design Corporation.

Trademarks

TrueUpdate and the Indigo Rose logo are trademarks of Indigo Rose Software Design Corporation. All other trademarks and registered trademarks mentioned in this document are the property of their respective owners.

Copyright

Copyright © 1992 - 2007 Indigo Rose Software Design Corporation.

All Rights Reserved.

LUA is copyright © 2003 Tecgraf, PUC-Rio.

UPX executable compression (<http://upx.sf.net>) copyright (C) 1996-2007 Markus Franz Xavier Johannes Oberhumer, copyright (C) 1996-2007 Laszlo Molnar, copyright (C) 2000-2007 John F. Reiser. All Rights Reserved.

Note: This user's guide is also available as a professionally printed, perfect-bound manual. To order your copy, please visit www2.ondemandmanuals.com/indigorose.

INTRODUCTION	10
Key Features of TrueUpdate	12
What's New in TrueUpdate?	15
Frequently Asked Questions	20
About this Guide	23
Document Conventions	24
 CHAPTER 1: THE TRUEUPDATE MODEL	 26
What Does TrueUpdate Do?	28
The TrueUpdate Client	29
TrueUpdate Servers	30
Server Configuration Files	30
The Update Process	31
Scripts	32
Script Tabs: Client vs. Server	33
The Client Script	33
The Server Script	34
Screens: Client vs. Server	34
The Client Screens	34
The Server Screens	35
Updating An Existing Client	35
 CHAPTER 2: THE PROJECT WIZARD	 36
Starting a New Project	38
 CHAPTER 3: THE DEVELOPMENT ENVIRONMENT	 60
Updating TrueUpdate	62
Learning the Interface	62

Getting Help.....	69
Setting Preferences.....	72
CHAPTER 4: INTRODUCTION TO SCRIPTING.....	76
What Are Scripts?	78
What Are Actions?	78
The Script Editor	79
Programming Features	81
Client Script	85
Server Scripts.....	85
Screen Events	86
Using the Action Wizard.....	87
Adding Actions.....	88
Editing Actions.....	93
Getting Help on Actions	95
Includes	96
Plugins.....	97
Where to Go from Here.....	99
CHAPTER 5: CREATING THE USER INTERFACE.....	100
The User Interface	102
Screens	102
The Screen Panes.....	104
Client Screens	104
Server Screens.....	105
Screen Lists.....	105
Adding Screens	106
Removing Screens	107
Editing Screens	107

Showing Screens.....	107
Screen Properties.....	108
The Language Selector.....	109
Session Variables.....	110
Screen Navigation	110
Screen Controls.....	113
Screen Layout	115
Themes.....	120
Choosing a Theme	121
Creating a Custom Theme	122
Overriding Themes	123
Other Options	124
Taskbar Settings.....	124
Actions	124
Alternative Interfaces	126
Silent Updates	126
Dialog-based Updates	126

CHAPTER 6: TRUEUPDATE SERVERS 128

What Are TrueUpdate Servers?	130
Types of TrueUpdate Servers.....	130
HTTP Server.....	131
HTTPS Server	131
FTP Server.....	131
LAN/Local Server.....	131
Adding, Removing and Editing Servers	132
TrueUpdate Server Redundancy.....	132
TrueUpdate Server Scalability.....	133

CHAPTER 7: SESSION VARIABLES	136
What Are Session Variables?	138
Built-in Session Variables	138
Custom Session Variables	142
Setting Session Variables	142
Using the Session Variables Tab	142
Using Actions.....	144
Removing Session Variables.....	145
Using the Session Variables Tab	145
Using Actions.....	145
Using Session Variables on Screens.....	146
When Are Session Variables Expanded?	146
Expanding Session Variables in Scripts	147
 CHAPTER 8: LANGUAGES.....	 152
Internationalizing Your Project.....	154
Run-time Language Detection.....	154
The Language Manager	155
Default Language	156
Language Files	156
Adding Languages.....	158
Removing Languages	159
The Language Selector	159
Localizing Screens.....	160
Importing and Exporting Screen Translations	162
Customizing Error Messages and Prompts.....	162
Advanced Techniques.....	163
Determining the Current Language	163
Changing the Current Language	165

Localizing Actions.....	166
Working with Existing Translated Messages.....	167
CHAPTER 9: SECURITY	170
Security in TrueUpdate.....	172
Client Side Security.....	172
Client Executable File	172
Client Data File	172
Client File Updating	173
Server Files Security	173
Types of Server Files	173
Client-Server Communication.....	175
Secure Protocols	176
Custom Client-Server Communication.....	176
Important Considerations	177
CHAPTER 10: BUILDING AND DISTRIBUTING.....	178
The Build Process.....	180
The Publish Wizard.....	181
Build Settings.....	184
Output	185
Upload.....	186
Constants	188
Pre/Post Build Steps.....	190
Build Preferences.....	191
Integrating the Client into your Software	192
Step 1: Adding the Client Files	192
Step 2: Triggering TrueUpdate.....	193
Source Code.....	194

Testing Your Update	196
Log Files.....	196
CHAPTER 11: SCRIPTING GUIDE	198
A Quick Example of Scripting in TrueUpdate	200
Important Scripting Concepts	201
Script is Global	201
Script is Case-Sensitive	202
Comments.....	202
Delimiting Statements	203
Variables.....	204
What Are Variables?	204
Variable Scope	204
Variable Naming	206
Reserved Keywords.....	207
Types and Values	207
Expressions and Operators	215
Arithmetic Operators.....	215
Relational Operators.....	216
Logical Operators	217
Concatenation	217
Operator Precedence.....	218
Control Structures	219
If.....	219
While	220
Repeat.....	221
For.....	222
Tables (Arrays)	223
Creating Tables	223

Accessing Table Elements.....	224
Numeric Arrays.....	224
Associative Arrays	225
Using For to Enumerate Tables.....	226
Copying Tables:.....	228
Table Functions	230
Functions	231
Function Arguments.....	232
Returning Values	233
Returning Multiple Values	234
Redefining Functions	234
Putting Functions in Tables.....	235
String Manipulation	236
Concatenating Strings.....	236
Comparing Strings	236
Counting Characters	238
Finding Strings:.....	238
Replacing Strings:.....	239
Extracting Strings.....	240
Converting Numeric Strings into Numbers.....	241
Other Built-in Functions.....	243
Script Functions.....	243
Actions	245
Debugging Your Scripts	245
Error Handling	245
Syntax Errors.....	245
Functional Errors	247
Debug Actions	248
Final Thoughts	255



Welcome!

Introduction

TrueUpdate is the finest toolkit available for adding a sophisticated software updating and patch management solution to both software products and network infrastructure. Whether you're a software developer needing to add a "Check for Update" feature to your program, or you're a network administrator wanting to automate the detection and application of system patches to hundreds or thousands of computer systems in your organization, TrueUpdate is an excellent solution.

In either case, the benefits include lower technical support costs, faster time-to-market, more frequent bug fixes, hassle-free security updates and quicker feature additions. Additionally, TrueUpdate was designed from the ground up to be flexible, easy to use, and easy to integrate. It's a solution that can be implemented with a minimum of effort so you can realize positive returns as quickly as possible.

We've improved the development environment to streamline your workflow and have introduced unprecedented flexibility with a brand-new scripting engine and action library. We've also introduced a customizable screen manager, support for secure data transfer, project themes (skins) and many other powerful and timesaving features.



What is TrueUpdate?

TrueUpdate is a comprehensive solution for software developers, network administrators and IT departments wanting to integrate automated update capabilities into their software and business processes. TrueUpdate provides a sophisticated client/server framework for determining required updates and then retrieving and applying the necessary patch or installation files using Internet or LAN protocols.

TrueUpdate was designed to meet the demand for a complete, web-enabled software updating solution that can be integrated quickly and easily into new and existing software products and networks, regardless of the installation and deployment methods used. TrueUpdate makes adding software update and patch management capability to a company's development infrastructure as simple and cost effective as possible.

TrueUpdate can be used to securely update any electronic content, from software applications and operating system patches to time sensitive data such as product catalogs, spreadsheets, financial information, sales figures and budgets. Additionally, TrueUpdate is extremely easy to integrate into your applications and offers nearly unlimited flexibility and customization options to ensure that your particular update requirements and goals are met.

The TrueUpdate brand is a recognizable mark of quality that you can proudly display to assure your clients and customers that your software updates are in the hands of the experts.

Key Features of TrueUpdate

Vista Compatible

TrueUpdate's design workspace and generated updates are compatible with Windows Vista, including a configurable "requested execution level" setting for the update's manifest.

Custom Resource Stamping

TrueUpdate allows you to use your own product icon and provides control of the resource information that you want written into the update's resources.

Integrated Code Signing

Protect the integrity of your company and products by code signing your updates with your own certificate during the build process.

MSI Actions

Over 35 MSI actions that leverage the Windows Installer service technology on the user's system. These are perfect for adding Windows Installer functionality to your update.

Dynamic Control of Client Systems

Once the TrueUpdate Client application is installed on a computer system, you have everything you need to ensure that the system is always up-to-date with the latest software and patches. Operating in conjunction with a TrueUpdate Server connection, the client software can be continually modified and reconfigured to carry out whatever system modification you require. This completely dynamic system puts you in full control and affords you flexibility that other products simply cannot match.

You Control the Server

With TrueUpdate, there is no need to relinquish control over the reliability of your update process. Other services lock you into using their servers; with TrueUpdate, you decide where your update files are hosted. You decide on the level of redundancy. You are in control of your update files, patches and servers. There is no need to rely on the uncertain future of an "update service," pay exorbitant annual fees or wait helplessly during downtimes you are powerless to resolve.

Industry Standard Protocols and Servers

TrueUpdate uses readily available client/server technologies rather than the proprietary servers required by competitive products. By making use of affordable and trusted protocols such as HTTP, HTTPS and FTP, organizations of any size can deploy TrueUpdate enabled software without the need for specialized and costly hardware and software platforms. TrueUpdate is built on the trusted, dependable standards you already rely on.

Easy to Integrate

TrueUpdate was designed to minimize the time it takes to add automated update capabilities to software applications. As a compact and standalone executable, the TrueUpdate Client application is extremely easy to integrate into your software. A typical software developer can have it done in only a few hours, and it doesn't matter what language you are working in – TrueUpdate is compatible with everything from Visual Basic to Delphi, C++, COBOL or whatever you are working with. In fact, it was designed from the ground up to be flexible, easy to use, and easy to integrate.

Runs Stand-alone or Embedded

The TrueUpdate Client application can be invoked in a variety of ways, depending on your particular needs. Software developers can easily embed the client software directly into their application, making use of the extensive “theme” support to match their own unique look and feel. Network administrators, meanwhile, can simply install the client application on each system and configure an appropriate execution schedule using standard system tools.

Automates Complex Tasks

TrueUpdate is built on a powerful scripting engine that is capable of quickly processing any of the more than 250 included high-level actions. Featuring everything from registry editing to file copying to web file downloads, this complete scripting environment contains everything you need to automate complex tasks and handle even the most sophisticated software updating requirements. No other tool gives you the same level of ready-to-use commands. TrueUpdate helps you to get your job done both faster and better!

Scalable and Fault-Tolerant

From the ground up, TrueUpdate was created to be fully scalable and fault-tolerant. It's easy to configure the client application to access redundant servers. If a server is unavailable for any reason, the client will move on to the next one until it can

establish a connection. Additionally, since you control the underlying server technology such as HTTP, HTTPS, FTP or LAN, you have ultimate control over load-balancing and distributed processing of client/server requests.

Reduces Costs

Automating the update process saves considerable time and expense. For software vendors, it reduces support costs by making it easier for your users to keep their software up to date—giving your tech support department fewer legacy support issues to deal with. And for network administrators who maintain hundreds or thousands of systems, the benefits of TrueUpdate far outweigh the initial investment.

Lightweight and Stand-alone

Written completely in optimized C and C++ code, the TrueUpdate client is small, weighing in around 500K in size. It's also completely self-contained—the TrueUpdate client has no external dependencies, so you don't have to distribute any extras to make it work. Unlike competitive products, it doesn't require the Java runtime, Visual Basic runtime, .NET framework or any other multi-megabyte runtime engine.

Works with Any Patch/Install Builder

TrueUpdate works with your choice of installation and patching tools. For a complete and fully integrated end-to-end solution, we'd recommend choosing Indigo Rose's Setup Factory and Visual Patch; however you're certainly not locked into doing so. If your company has standardized on other install/patch builders, such as those offered by Installshield, Wise or ZeroG, TrueUpdate can accommodate them. In fact, TrueUpdate is even able to work with zip archives and individual data files, should you desire.

Trusted by Professionals

Thousands of software developers trust Indigo Rose software tools. In fact, our products such as TrueUpdate, Setup Factory and Visual Patch are used to distribute and manage software on millions of customer and client systems around the world. Additionally, all of our products are backed up by world-class technical support services.

What's New in TrueUpdate?

Vista Compatible

TrueUpdate's design workspace and generated updates are compatible with Windows Vista, including a configurable "requested execution level" setting for the update's manifest.

Custom Resource Stamping

TrueUpdate allows you to use your own product icon and provides control of the resource information that you want written into the update's resources.

Integrated Code Signing

Protect the integrity of your company and products by code signing your updates with your own certificate during the build process.

MSI Actions

Over 35 MSI actions that leverage the Windows Installer service technology on the user's system. These are perfect for adding Windows Installer functionality to your update.

Extensive Project Wizard

Getting your project started has never been easier, thanks to the new Project Wizard. You'll be walked through each step of the process, including setting up your TrueUpdate Server, choosing download methods, customizing the TrueUpdate Client and configuring your automatic upload settings. The wizard includes dozens of options and project templates to choose from.

More Server Types

TrueUpdate supports all of the most popular protocols for client/server communication. This includes HTTP, FTP and LAN (both UNC paths and mapped drives), as well as secure connections and file transfers using HTTPS. Full support for HTTP basic authentication, timeouts, ports, FTP passive mode, usernames and passwords is also built-in.

Automatic Firewall and Proxy Server Detection

TrueUpdate handles proxy servers and firewalls in an entirely seamless and industry-standard manner. Internet communication is now done entirely through the WinINet

API, making it fully compatible with any corporate network hardware and software that supports standard Windows/Internet Explorer functionality.

Supports More Patching Methods

It doesn't matter what method you plan to use to patch/update the software on the client system; TrueUpdate can work with them all. Whether you're deploying a self-contained patch executable (such as those created by Visual Patch), a single-file installer (like those created with Setup Factory), zip files, individual data files, multiple binary patch files or whatever else you require, TrueUpdate can handle it. We've even included a variety of project templates and samples to get you started.

Sophisticated Version Analysis

TrueUpdate provides many flexible methods for analyzing existing software versions on the client system. It can access the registry, read values from INI files, compare file CRC values, query file version resource information or even use time/date stamps if necessary. Once a version has been identified, TrueUpdate can use a series of actions to bring that version up to date as required.

Powerful Scripting Engine

TrueUpdate includes the same scripting engine as Setup Factory and AutoPlay Media Studio. Based on the popular "Lua" language, this all-new and incredibly powerful free-form scripting engine gives you unprecedented control over your software updating system. This easy to use language features everything from "for, repeat and while" loops, to "if/else" conditions, functions, variables and associative arrays. Paired with the built-in action library, full mathematical evaluation and Boolean expressions, there is simply nothing you can't achieve. We've also built in an "Action Wizard" and "Quick Scripts" feature so you can get right up to speed creating powerful projects to handle even the most demanding update tasks.

Extensive Action Library

TrueUpdate includes a built-in library of more than 250 powerful yet easy to use actions. Here, you'll find high-level actions to handle everything from text file editing to system registry changes. You can execute programs, call DLL functions, query drive information, manipulate strings, copy files, enumerate processes, start and stop services, interact with web scripts, display dialog boxes and much more. There is also a full suite of file download actions including FTP, HTTP and secure HTTPS transfers, including new automatic support for firewalls and proxy servers.

Easy to Use Action Wizard

You don't have to be a wizard to create powerful update systems with TrueUpdate. We've built the wizard into the software! Simply choose the action you want from a categorized list (complete with on-screen interactive help), fill in the requested information fields and the wizard does the rest. Making changes is just as easy. Click on the line you want to change and press the "edit" button to go back to the original form. It's really that easy.

Color Syntax Highlighting Script Editor

The TrueUpdate script editor features all of the professional features you'd expect. There's color syntax highlighting, code completion, function highlighting, as-you-type action prototypes, Ctrl+Space function listings and even context-sensitive help. If you're used to programming in Microsoft® Visual Basic, Microsoft® Visual C++ or any other modern development language, you'll be right at home.

Improved Client Interface

TrueUpdate gives you nearly unlimited flexibility in designing the user interface. You can choose from a fully interactive wizard, a minimal dialog box style, a completely silent approach or a "silent until update available" system. Additionally, thanks to the new screen gallery and manager, you can choose from over 25 ready to use wizard-style dialogs, or customize them to fit your needs.

Screen Gallery & Manager

With a library of more than 25 different screen templates to choose from, TrueUpdate is miles ahead of both previous versions *and* the competition. There are pre-built layouts to handle just about any task you can dream up, and it's easy to adjust them to fit your needs exactly. You'll find check boxes, radio buttons and edit fields to popular screens like license agreements, folder selection and other advanced options. The Screen Manager allows you to add and remove screens at will and adjust the sequence with a simple drag-and-drop motion. Each screen features a real-time preview so you can see the result of your changes as you work.

Themes and Skins

Choose from over twenty included themes (skins) for your project or even make your own. It's as easy as viewing a live dialog preview and picking your favorite style. You can configure everything from fonts (face, color, size, style) and banner images to body/background graphics, control colors (buttons, check boxes, radio buttons) and more.

Publishing Wizard

Once you've got your project ready to go, the Publishing Wizard will help you to package it up and upload it to your server. The wizard will create the client application, server configuration files and everything else you need to put your update system into operation. You'll also get a full project manifest that tells you what files have been generated and what you need to do with them.

Automatic Server Uploads with Secure FTP Support

TrueUpdate makes it easy to keep both your client and server files up-to-date. The new automatic upload feature turns the build process into a complete publishing solution. It supports file copying to UNC or mapped drives, standard FTP, or secure SFTP. Of course, should you wish to handle it all yourself, there is a manual upload option as well.

Compact Client Application

Smaller and faster means a better experience, and TrueUpdate delivers. Compare our tiny ~500 KB client application to the competition and see for yourself. And since it is written completely in optimized C++ code, there are no external dependencies or runtimes required.

Encrypted Configuration Files

Both client configuration files and server data files are automatically compressed and encrypted using your own private key. Featuring the secure Blowfish algorithm, your scripts and configuration info are safe from any casual tampering or viewing.

Runs Silent or Interactive

Your TrueUpdate projects can be configured to operate without displaying user interface dialogs, prompts, messages or errors. Silent operation lets you maintain control over hundreds or thousands of workstations while enforcing corporate standards. The client application can easily be called from programs, system schedulers or automatic processes.

Expandable with Action Plugins

TrueUpdate can be easily expanded with Action Plugins. These plug-in modules can extend the product in infinitely powerful ways, such as adding support for SQLite databases, XML, MD5 hashing, data encryption and much more. Tight integration with the design environment—including IntelliSense style code completion and syntax highlighting—makes them just as easy to use as built-in actions. Plugins are

available through Indigo Rose as well as third-party developers thanks to Indigo Rose's freely available plug-in development kit.

International Language Support

TrueUpdate offers unsurpassed support for multilingual projects right out of the box. Update systems created with TrueUpdate can automatically determine the language of the client operating system and adjust the display of screens and messages appropriately. Whether you need to support English, French, German, Spanish, Italian or any other language recognized by Windows, you simply provide the text and TrueUpdate takes care of the rest!

Built-in Spelling Checker

Now it's easier than ever to make sure that typos don't creep into your projects. Basically anywhere you can type, you can perform a spell check to ensure error-free text. Dictionaries are available for over a dozen languages including English, French, German, Italian, Spanish, Dutch, Swedish, Danish, Croatian, Czech, Polish and Slovenian.

Client Log Files

The client application can easily log each action giving you an accurate record of everything that is happening behind the scenes. It's perfect for debugging or even archiving. You can control the level of detail being logged, including options for recording script actions.

Unattended Builds

TrueUpdate fits seamlessly into your daily build process. Creating your update project every time you build your source code makes it easy to test early and often. Used in conjunction with design-time constants (e.g. DEFINE's) and build response files, your TrueUpdate project can be kept up to date simply and automatically.

Works with Windows 95 and Up

Update systems created with TrueUpdate work just fine on every Windows operating system from Windows 95 to Vista and beyond. Compare that to competitive tools and you're sure to be surprised at their requirements. If you need to support legacy systems, your choice is clear!

Frequently Asked Questions

Who needs TrueUpdate?

Software developers, network administrators and end-users alike share in the benefits of TrueUpdate enabled software. Developers appreciate it for the control it gives them over software that has already been deployed. Network administrators benefit from improved security and better control of network nodes. End-users see TrueUpdate enabled software as an assurance of quality—a symbol that the developer is there to stand behind their product.

Any company who develops software applications or distributes data needs TrueUpdate. Any organization that needs to ensure its network is secure and updated with the latest patches needs TrueUpdate. Anyone who needs timely and secure synchronization of documents and files needs TrueUpdate.

Ensuring that everyone is using the most recent point release of a particular package makes good sense. It eliminates legacy technical support calls, and corresponding user frustration. It ensures that data is current. It also serves to maintain customer satisfaction. And that's something everyone can appreciate.

What can you do with TrueUpdate?

Software products and network clients that have been TrueUpdate enabled can quickly and efficiently determine if they are out of date. Embedding TrueUpdate into a software product makes it extremely easy to manage, control and update “in the field”. Likewise, deploying the TrueUpdate client to computer systems throughout your network gives you a fast and manageable way to ensure that each node is current with the latest patches, documents and data that your company requires.

What's wrong with traditional update methods?

One of the most serious problems with traditional update methods is that they require the users to do most of the work. The problem with relying on an update process that demands too much user involvement is obvious; the users might decide it isn't worth the hassle to keep their systems up to date. This results in a higher incidence of legacy support issues, more security holes and other related problems. Additionally, when users don't update, they don't benefit from bug fixes and product improvements. TrueUpdate helps solve all of these problems.

What is automated updating?

Automated updating is the ability of software to handle some or all of the update process so the user doesn't have to. The update process consists of all the steps required to determine whether a newer version exists for a given piece of software, as well as all the steps required to bring an older version up to date. Automating the update process allows software to keep itself current after it has been deployed to users.

How important is automated updating?

Today's rapid product cycles, security vulnerabilities and short turnaround times make it more important than ever to get new versions into the hands of users quickly and efficiently. The Internet has created a highly competitive market where users expect immediate results; the prize often goes to the company that reacts the most quickly to changes in user needs and perceptions. In order to streamline software deployment, it is becoming increasingly desirable for software authors to incorporate automated updating abilities into their software.

How does TrueUpdate benefit the software developer?

The easier it is for your users to update your software, the more likely it is that your users will be using the latest version. Your technical support team will have fewer legacy issues to deal with. The easier it is for you to release updates, the more often you can release them. You won't have to hold back releases until you have made enough changes to justify the effort required to prepare updates using traditional update methods.

Why add TrueUpdate to your software application?

In a traditional release cycle, once your application or data files are released to customers, clients or other end-users, they are static, expensive and time-consuming to alter. However, once you add TrueUpdate to your application, you'll be able to easily update your product or data as often as you require.

From the developer's point of view, when a new software release is available—perhaps a bug has been fixed—TrueUpdate makes it trivial to publish the changes and bring all of your users up to date. The next time the user runs the application, TrueUpdate will detect that a new release is available and take steps to handle it.

How easy is it to add TrueUpdate to an application?

As a self-contained executable, the TrueUpdate client can be integrated into your application in less than a day. While the actual time required depends on the amount

of integration you desire and your level of programming knowledge, most developers should be able to complete the job in just a few hours. We even include sample source code for adding a “Check for Update” menu command and tips to get you going.

Alternatively, the TrueUpdate client executable can be distributed directly, rather than being embedded within an application. This allows the TrueUpdate client to be called manually from the start menu, from a shortcut on the user’s desktop or automatically with system schedulers. This method of adding TrueUpdate to your application is as easy as installing a shortcut onto the user’s system.

How does TrueUpdate impact technical support?

TrueUpdate allows your users to benefit quickly from any new features and bug fixes you develop, which in turn reduces the incidence of support calls. Keeping users up to date makes it easier to support them when incidents occur.

How will TrueUpdate impact our customers and clients?

Today’s users are savvy; they demand responsiveness from software companies and they want tools that meet their needs and make them more productive. In order to maintain customer loyalty and maximize the user’s experience with your software, you need to make updating the software as easy as possible. Making it easy for users to update your software shows that you’re committed to supporting it.

How does TrueUpdate benefit the network administrator?

Keeping a corporate, educational or government network up-to-date with the latest security patches, applications updates and operating system fixes is a time consuming ordeal. Without tools like TrueUpdate, the task is virtually impossible. By installing the TrueUpdate client onto your networked computers, you’ll be able to quickly and effectively roll out whatever software you want throughout your organization. The client software can analyze the computer system, decide what is currently installed and then take whatever actions you determine are needed to bring that system up-to-date. It’s fast, easy and automatic.

I’m not a developer...do I still need TrueUpdate?

Absolutely! You don’t need to be a software developer to benefit from TrueUpdate. As a stand-alone executable, the TrueUpdate client can be used to update all kinds of files. You could use TrueUpdate to distribute product catalogs to your sales teams, or to remotely configure system files across your corporate network. Price lists, help files, quarterly reports, internal support videos—TrueUpdate can help you keep anything up to date.

Does TrueUpdate actually install files?

With a full suite of file operations, including specialized actions to download, copy, delete, rename and even zip and unzip files, TrueUpdate may be all you need to install files onto a user's system. For situations requiring a more manageable and compartmentalized solution to file installation and patching, we'd recommend using Indigo Rose's Setup Factory and Visual Patch products. Used in conjunction with TrueUpdate, these products make up a complete and robust software deployment and management solution.

About this Guide

This user's guide is intended to teach you the basic concepts you need to know in order to build a working update system. You'll learn the ins and outs of the program interface and how to perform many common tasks.

The guide is organized into 11 chapters:

- Chapter 1:** The TrueUpdate Model
- Chapter 2:** The Project Wizard
- Chapter 3:** The Development Environment
- Chapter 4:** Introduction to Scripting
- Chapter 5:** Creating the User Interface
- Chapter 6:** TrueUpdate Servers
- Chapter 7:** Session Variables
- Chapter 8:** Languages
- Chapter 9:** Security
- Chapter 10:** Building and Distributing
- Chapter 11:** Scripting Guide

Each chapter begins with a brief overview and a list of the things you will learn in that chapter.

Document Conventions

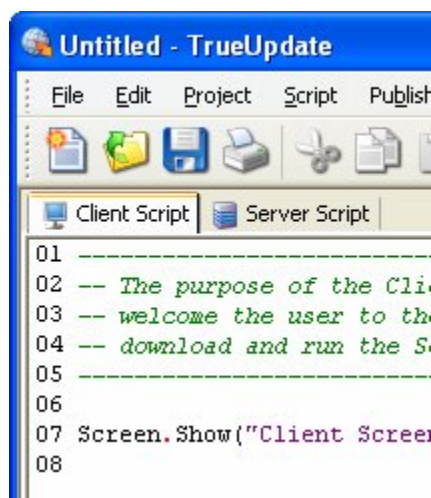
This user's guide follows some simple rules for presenting information such as keyboard shortcuts and menu commands.

Keyboard Shortcuts

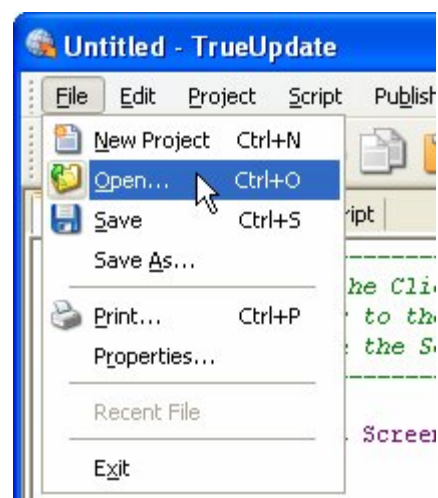
Keyboard shortcuts are described like this: press Ctrl+V. The “+” means to hold the Ctrl key down while you press the V key.

Menu Commands

Menu commands are described like this: choose File > Open. This means to click on the File menu at the top of the TrueUpdate program window, and then click on the Open command in the list that appears.



Click on the File menu...



...and click on the Open command

Typed-In Text

When you're meant to type something into a text field, it will be presented in italics, like this: type *"TrueUpdate makes updates easy"* into the Message setting. This means to type in "TrueUpdate makes updates easy", including the quotes.



Chapter 1:

The TrueUpdate Model

Like any system that involves client-server interaction, updating software is a sophisticated process. It involves advanced technologies and protocols and rules that, without a clear and defined plan, would seem incredibly daunting to employ.

TrueUpdate is specifically designed to make the entire update process as simple to understand and as easy to accomplish as possible. Much of this is due to the unique update model that TrueUpdate employs. TrueUpdate provides the framework that makes it possible—in fact, easy—for you to build an update for your software.

Before you start working with TrueUpdate, however, it is important to understand the way TrueUpdate works.

In This Chapter

In this chapter, you'll learn about:

- What TrueUpdate does (and what it's capable of)
- The TrueUpdate Client
- TrueUpdate Servers
- Server configuration files
- A typical update process
- The value of scripts in TrueUpdate
- Client and server scripts
- Client and server screens
- How the client is able to update itself

What Does TrueUpdate Do?

At its simplest, TrueUpdate is an updater of software. It determines whether an update is available, downloads the required files, and then uses them to perform the update.

Anything that falls within that broad definition is possible with TrueUpdate. All of the details are in your control, from the specific methods used to detect the installed version, to the steps that are taken to actually update the software. An update can be as simple as making a subtle change to a registry key, or it can involve downloading and applying multiple patch files from a geographically selected download location.

Most commonly, however, TrueUpdate will be used to perform these basic functions:

- Determining the version that is installed
- Deciding whether an update is available
- Deciding whether an update is required
- Acquiring the appropriate patch file
- Applying the patch to update the installed version

Other Uses for TrueUpdate

The above steps describe the most common use of an updater like TrueUpdate. However, TrueUpdate is an incredibly flexible tool. In addition to handling the countless variations on the update process that are possible, TrueUpdate can be used to perform many other tasks—including some that are completely unrelated to updating software. For example, you could use TrueUpdate as a communications tool, automatically detecting and displaying a new “message of the day” from a central location. Or you could use it as a general development environment, taking advantage of its sophisticated system actions and wizard-like screens to build standalone configuration utilities. The possibilities are indeed limitless.

The TrueUpdate Client

The TrueUpdate Client is the client application that will actually perform the update process. It is generated when you build your TrueUpdate project. This is the “updater” that you will distribute to your users in order to provide the update ability.

In other words, the TrueUpdate Client is the program that does the work of your update. Everything in your TrueUpdate project—from the project settings that you configure, to the server-side scripts that you create—either directly affect how the TrueUpdate Client is built, or serve as instructions that define how it performs the update.

The TrueUpdate Client is a compact, self-contained application, without any external dependencies such as .Net, Java, or any other needlessly complicated technologies. It takes advantage of standard Internet protocols and encryption methods to perform the update process in a robust and secure way.

It’s also designed to be mindful of privacy needs. The TrueUpdate Client doesn’t require any information to be sent to the server at all—everything from the installed version detection to the decision about whether an update is required is handled entirely at the client end. This *client-pull* method completely avoids the privacy issues that revolve around other methods that require detailed information about the user’s system and installed software to be transmitted to a remote location.

Of course, TrueUpdate can still communicate with a remote location if you need it to; in fact, it has built-in actions to submit information to standard web scripts using both unsecured and secured transfer protocols. However, this functionality is entirely in your control; a TrueUpdate Client will never send anything to a server unless you specifically design it to do so.

The TrueUpdate Client is also designed to keep itself up to date. It does this by automatically downloading any updated versions of its client data or executable files that it finds at the TrueUpdate Server. (The latest versions of these files are included in the server configuration files that are uploaded to each TrueUpdate Server location.)

Tip: For more information on the client’s self-updating ability, see *Updating An Existing Client* on page 35.

TrueUpdate Servers

A TrueUpdate Server is any location where the TrueUpdate Client can download the server configuration files that tell it how to perform the update. This can be a standard HTTP server, a standard FTP server, or even a folder on your local area network.

Note that this is completely independent of any specific hardware platform, operating system, or server software. TrueUpdate doesn't lock you into any proprietary server technology. In fact, it doesn't require you to run anything on a server at all.

All you need to create a TrueUpdate Server is a place to put your server configuration files where the TrueUpdate Client will be able to get them. "Creating" the server involves nothing more than uploading a handful of files and adding the new location in your TrueUpdate project settings.

For example, if you upload the server configuration files to your company's web server, and add the location in your project settings, you've just created a TrueUpdate Server. Want another TrueUpdate Server for redundancy? Upload the files to another web server, or to an FTP server, or to a folder on your network. You could even copy the files to a USB key and pass it around the office—anywhere that your TrueUpdate Client can access the files is a valid TrueUpdate Server location.

Because TrueUpdate Servers use standard Internet protocols, you are free to take advantage of the full range of server technologies available to you. Whether your server is an old PC in your basement, or a fully load-balanced and distributed server farm in a high-security building, the choice of where and how to host your files is entirely up to you.

Note: For more information on TrueUpdate Servers, please see Chapter 6.

Server Configuration Files

Server configuration files provide the server-side component of the update process in TrueUpdate. This handful of files contain all the information the TrueUpdate Client needs in order to keep itself up to date and to perform the actual update of your software.

Specifically, the server configuration files allow each client to:

- Update its client-side data and executable files if newer versions become available
- Determine what version of your software is currently installed
- Determine whether an update is required
- Take the steps required to update the installed software if it is out of date

The server configuration files are generated whenever you build a project. They are completely project-specific—the server configuration files that you build in one project will only work with a TrueUpdate Client generated from that same project. In other words, TrueUpdate Clients and server configuration files from different projects are *incompatible*.

Note: TrueUpdate Clients are only compatible with server configuration files from the same project.

The Update Process

Here is a description of a typical update process in detail.

1. The TrueUpdate Client welcomes the user to the update by displaying an appropriate screen or popup dialog box.
2. The client then attempts to download the server configuration files from the first server in its list of TrueUpdate Servers. (By default, it will attempt to connect to each server location in the list until it succeeds.)
3. Once it connects to a server location, it checks to see if the location contains a newer version of the client data or client executable file. If it does, the client downloads the appropriate server configuration files (e.g. the .ts2 or .ts3 file, or both), and then restarts in order to use the new version of itself.
4. Assured that it is up to date, the client next downloads the server script and screens, which are contained in the .ts1 server configuration file.
5. After loading the server screens into memory, the TrueUpdate Client runs the server script.

6. The first function of the server script is to determine what version of the target software is currently installed on the user's system. This version is generally referred to as the *installed version*.
7. Once the installed version has been determined, the server script compares it to a target version. The target version represents the desired version of the software—i.e. the version that the script was designed to update the software to.
8. If the above test determines that the installed version is already current, the server script informs the user that they already have the latest version, and then exits.
9. If on the other hand the test determines that an update is required, the server script proceeds with the actual update process. While the actual update method will vary from project to project, the most common method involves downloading a single executable patch file and then running that file with appropriate command line options.
10. If the patch file returns a value that can be used to determine whether it succeeded, the server script will usually check this value and then inform the user of the success or failure of the update.
11. If the patch file doesn't return any such information, the server script will either wait for the patch to finish and then inform the user that the update process is complete, or simply exit silently and let the patch continue on its own.

Scripts

Scripts are a key component of TrueUpdate. In fact, the entire client-server interaction is scripted, and the built-in screens implement their functionality using scripts triggered by various screen events.

This is a very important feature, because it exposes the implementation to you for modification. In other words, the scripted nature of TrueUpdate makes it possible for you to completely control the behavior of your update.

A script is essentially a list of instructions for the TrueUpdate Client to follow. Each script contains a sequence of commands, called *actions*, which work together to perform a specific task. A script can contain a single action, or any number of actions. These actions combine to form a series of steps that are performed sequentially, with

one action following another. The sequence of actions can either occur in a direct and straightforward manner, or—by taking advantage of control structures like “if” and “while”—can incorporate sophisticated techniques such as decision-making, branching and looping.

Note: Scripts are ultimately just text documents that follow a specific syntax. The actions and control structures in a script are represented by specific keywords, which the TrueUpdate Client interprets as instructions that tell it what to do. You can edit these instructions as easily as you would edit any other text.

Script Tabs: Client vs. Server

Each TrueUpdate project contains two main scripts: a client script, and a server script. In the TrueUpdate development environment, these two scripts are represented by the Client Script tab, and the Server Script tab.

The scripts are divided according to where they are stored. The client script is built into the TrueUpdate Client application, and the server script is stored in one of the server configuration files.

Together, these two scripts implement the client-server interaction and control the entire update process.

The Client Script

The client script’s job is to introduce the user to the update, and download and run the server script.

Downloading the server script is done using the TrueUpdate.GetServerFile action. Since this action involves downloading files, it is usually called from a client screen (e.g. the Download Server Script screen) so that progress can be shown.

Once the server script is downloaded, it can be executed using the TrueUpdate.RunScript action.

The Server Script

The server script's job is to determine whether an update is required, and then to actually perform the update—either by running a series of actions or by launching a separate installer or patch file.

Because it is downloaded each time your TrueUpdate Client connects to a TrueUpdate Server, you are free to modify the server script at any time—even after the TrueUpdate Client has been distributed.

This separation between the client and the server scripts is an essential feature of TrueUpdate. It allows you to adjust the update process for your software at any time, without any modifications to the client at all. In other words, it allows you to modify your update process without having to redistribute new clients to your users.

Screens: Client vs. Server

Screens are the individual windows that make up any wizard-styled update. When you navigate through an update by clicking the Next and Back buttons, you are navigating from screen to screen.

Note: Each screen has a number of events, with associated scripts that get executed when the corresponding events are triggered. By modifying these scripts you have full control over the functionality of the screens in your project.

Like the client and server scripts, the screens in your project are divided into two main areas. In the TrueUpdate development environment, these areas are represented by two screen panes, which by default are tabbed together in the upper right corner of the program window. There is one screen pane for the client-side screens, and one for the server-side screens.

The Client Screens

When you build your project, the screens on the Client Screens pane are packaged with the TrueUpdate Client and are accessible to the end user without having to download anything from the Internet. These screens cannot be updated without updating the client itself.

The Server Screens

The screens on the Server Screens pane are stored in a server configuration file. These screens are accessible to the end user only after the client has downloaded the server configuration files. Because the screens on the Server Screens pane are downloaded whenever the TrueUpdate Client connects to a TrueUpdate Server, you are free to change them after your client has been distributed.

Updating An Existing Client

The client script and client screens are stored within the client's data file, which is normally distributed initially with your software.

Generally, it is preferable to write the client script in such a way that it does not need updating after the client has been distributed. If changes are made, the client will need to be restarted in order to update itself.

Of course, there may be an occasion when you need to add actions to your client script or to add another screen after your client has been distributed.

Not to worry; TrueUpdate has been designed to handle such cases. Every time your project is built, three server configuration files are generated. One of these files (the .ts1 file) contains the server scripts and server screens. The other two server configuration files contain the latest versions of your client data file (.ts2) and client executable (.ts3). This allows each client to automatically update itself if you have made changes to the client script and screens.

Here's how it works: Whenever a TrueUpdate Client connects to a TrueUpdate Server in order to begin the update process, it first checks to see if the .ts2 and .ts3 files on the server are newer than the client's current data file and executable. If either of the files is newer, the client downloads the new file, replacing the existing file with the new version. Remember that in order to replace either of these files, however, the client needs to restart itself; therefore, if at all possible, it is better to avoid updating the client script and client screens after the client has been distributed to your users.



Chapter 2:

The Project Wizard

Every journey begins with a first step, and TrueUpdate makes this step as painless as possible. With the built-in project wizard, creating professional product updates is as easy as filling in the blanks and clicking Next. This chapter will introduce you to the easy-to-use TrueUpdate project wizard.

In This Chapter

In this chapter, you'll learn about:

- Starting a new project
- The project wizard

Starting a New Project

Everything has to start somewhere. In TrueUpdate, the design process starts with the creation of a new project.

A project is simply the collection of scripts, screens, settings and everything else that goes into building an update. For example, each project will contain the scripts and screens that will be built into the TrueUpdate Client application as well as those that will be contained in the server files. The settings for all of the scripts and screens in a project are stored in a single file, known as the project file.

When you start a new project, TrueUpdate's project wizard walks you through the first few steps of project creation. This helps you get your project started quickly without missing any of the basics.

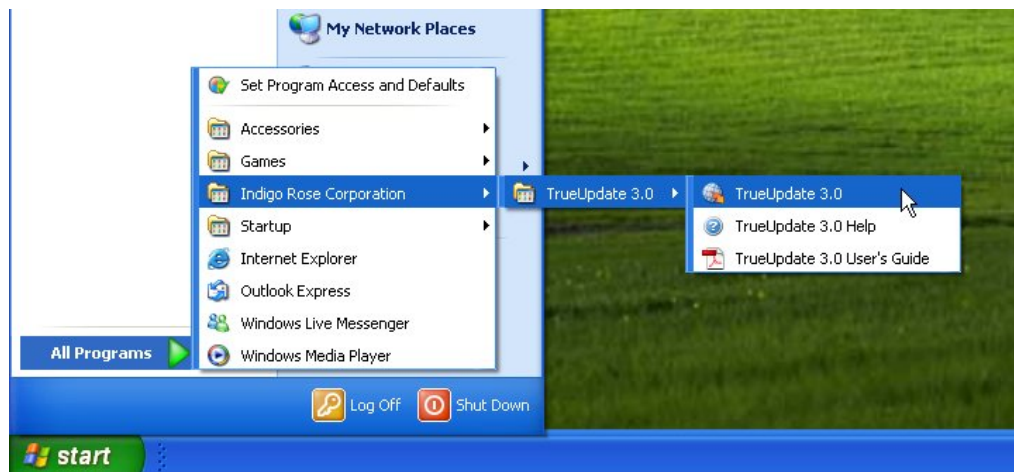
Let's open the TrueUpdate program and start a new project.

1) Open TrueUpdate.

Use the Start menu to launch the TrueUpdate program.

You'll find TrueUpdate under:

Start > Programs > Indigo Rose Corporation > TrueUpdate 3.0



2) When the Welcome dialog appears, click on “Create a new project.”

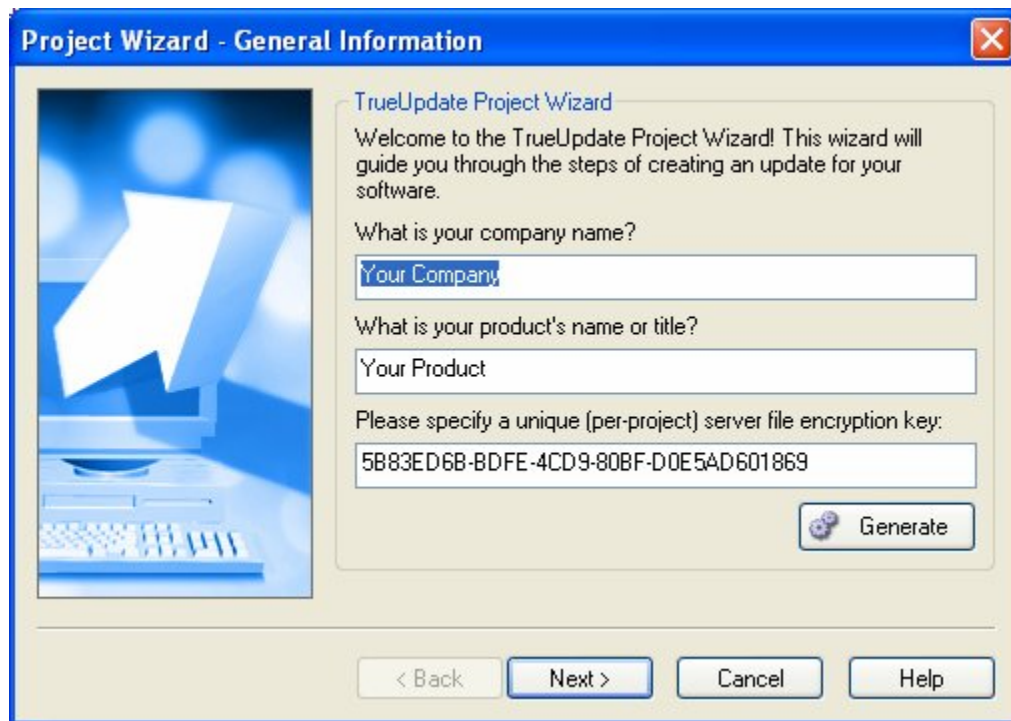
The Welcome dialog appears whenever you run TrueUpdate. It not only welcomes you to the program, it also lets you easily create a new project, open an existing one, or restore the last project you worked on. (Restoring the last project automatically opens the project you were working on the last time you ran TrueUpdate.)



When you click on “Create a new project,” the Welcome dialog closes and the project wizard appears.

3) Enter your project information and click Next.

First, the project wizard asks you for three pieces of information related to your project. Simply enter your company name and product name in the appropriate fields. Additionally, you may specify a unique server file encryption key; however, the randomly generated default key is unique to each project and is typically sufficient.



The screenshot shows a Windows-style dialog box titled "Project Wizard - General Information". On the left is a blue-tinted image of a computer monitor displaying a document. The main area contains the following text and fields:

- Header: TrueUpdate Project Wizard
- Welcome message: "Welcome to the TrueUpdate Project Wizard! This wizard will guide you through the steps of creating an update for your software."
- Question: "What is your company name?"
- Field: A text box containing "Your Company".
- Question: "What is your product's name or title?"
- Field: A text box containing "Your Product".
- Text: "Please specify a unique (per-project) server file encryption key:"
- Field: A text box containing the hexadecimal string "5B83ED6B-BDFE-4CD9-80BF-D0E5AD601869".
- Button: A "Generate" button with a key icon.
- Footer: Four buttons: "< Back", "Next >", "Cancel", and "Help".

When you've entered all your information, click Next to move to the next step in the project wizard.

Tip: At any step in the project wizard, you can click Cancel to go straight to the program window with all of the default project settings untouched (i.e. to start with a "blank" project).

4) Select the Wizard style interface and click Next.

The next step in the project wizard is to specify which type of user interface your TrueUpdate Client will use.



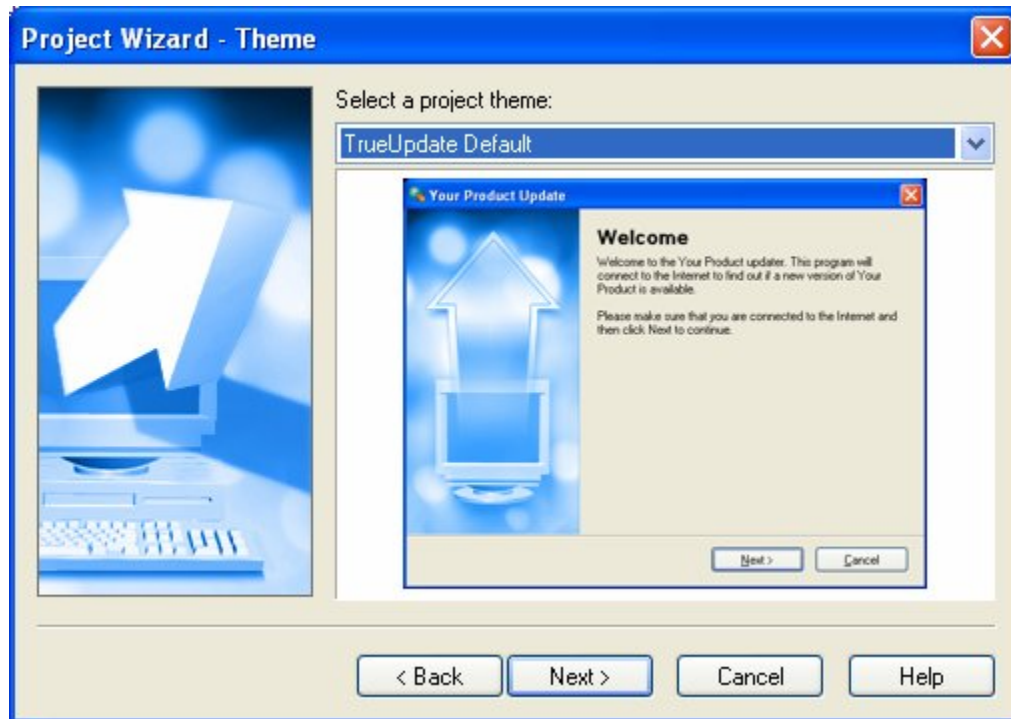
The most common update user interface is the default Wizard style. A Wizard style interface presents the user with a series of screens which they can navigate through by clicking Next and Back buttons. Wizard interfaces are considered very user friendly because they present and request information in discrete, guided steps, which makes the overall process easier for the user to understand.

The other two interface styles are Dialog and Silent. A dialog user interface uses popup dialogs or “message boxes” as opposed to screens to guide the user through the update. A silent update runs entirely in the background, and has no user interaction whatsoever.

The last option in this step allows your TrueUpdate Client to run silently until an update is available. Once an update is available, the client will continue in whatever style you have chosen (Wizard or Dialog).

5) Select a theme for your update, and click Next.

This step allows you to configure how your update will look.



Once you've selected your project theme, click Next to proceed to the next step.

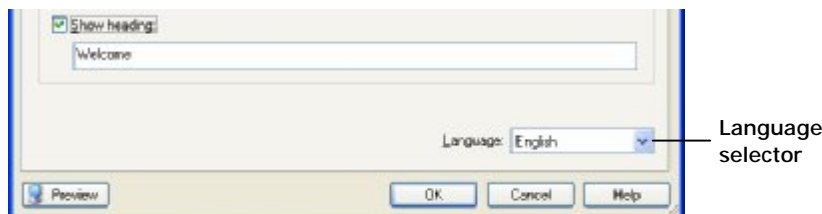
6) Select the languages you plan to support in your update and click next.

TrueUpdate supports multilingual updates. In this step, you can select which languages you want your update to support by checking them off. As well, you can select the default language to be used in the event that the user's system employs a language you haven't specifically accounted for.



Two things happen when a language is supported. First, TrueUpdate will use the text from that language's message file (if one exists) for the update's built-in messages when that language is detected at run time. (If a message file doesn't exist for a particular message, the default language's message file will be used.)

Second (and more important), you will be able to localize the text in your project for each supported language. For example, if you choose English, French and German as your three supported languages, you will be able to enter different English, French and German text on your project's screens. This is done by simply choosing a different language from the *language selector*, which appears wherever there is text that you can translate in your project.

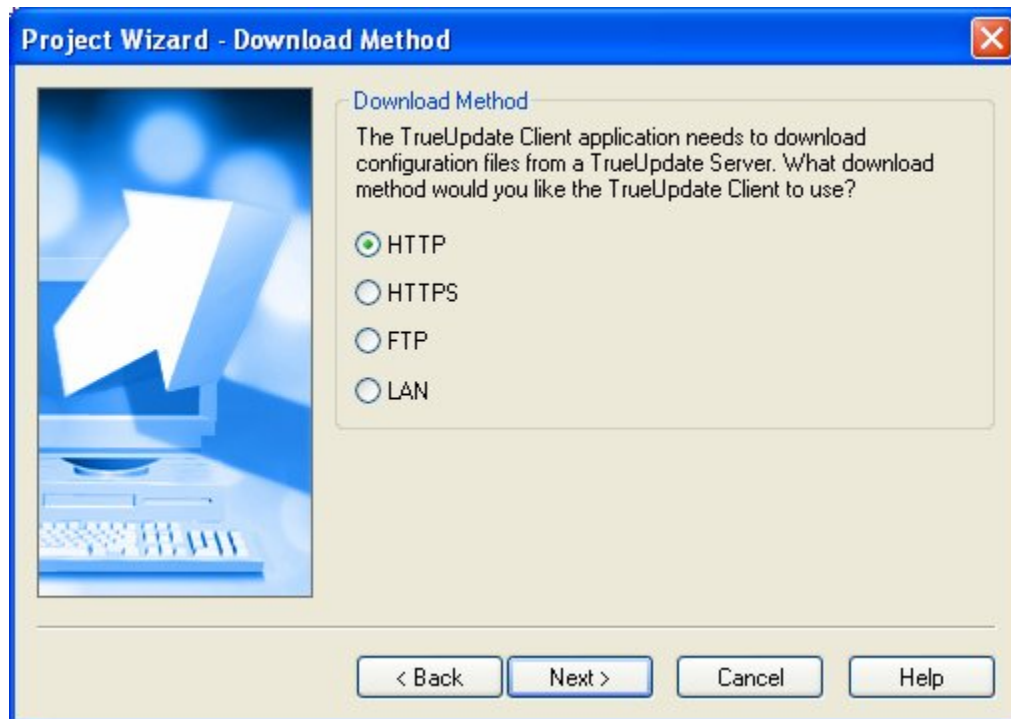


Note: Only supported languages will appear in the language selector; you will only be able to select a language on a screen if that language was “checked” in the list of supported languages.

Once you’ve selected the languages you want to support, click Next to proceed to the next step.

7) Select which download method your update will use, and click Next.

TrueUpdate supports a variety of file transfer protocols. This step allows you to specify which protocol your update will employ.



The first option is to use the standard hypertext transfer protocol (HTTP). This protocol was designed for accessing files on a web server from your local system. This is the same protocol that Internet Explorer uses, and is generally the easiest method to use. Since it uses port 80 by default, it's also the most likely method to be allowed across a user's firewall; chances are that if a user can browse the Internet from their computer, they'll be able to run your update.

The second option is to use the secure hypertext transfer protocol (HTTPS). This is essentially a "secured" version of HTTP that uses the Secure Sockets Layer (SSL) to perform the data transfer in a more safeguarded fashion. It operates on port 443 by default.

Note: In order to use the HTTPS protocol, you must have a secure web server to connect to. You can't use HTTPS to connect to a regular web server; you can only use the HTTPS protocol if your web server supports it.

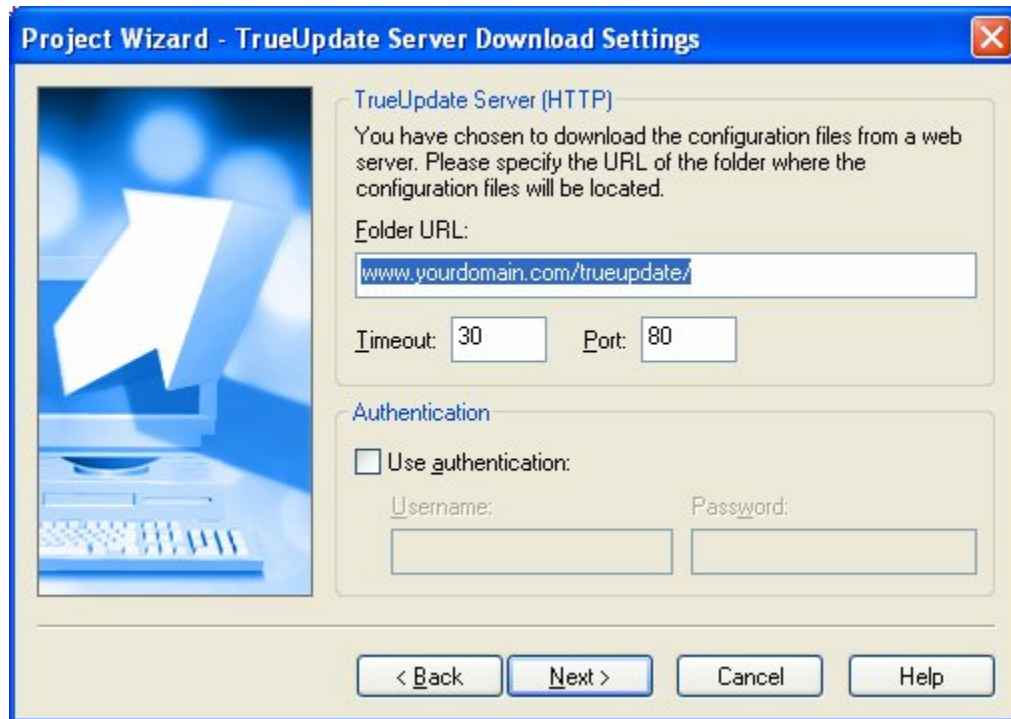
The third option is to use the file transfer protocol (FTP). This protocol was designed for exchanging files over the Internet, and supports file transfers in both directions, i.e. downloading *and* uploading. (It is commonly used to upload web page files to the web server that will host them.) In order to transfer files to or from an FTP server, you must logon to it by providing a user name and password. Many FTP servers provide public access to their files by accepting a "guest" logon; this is usually referred to as "anonymous ftp." The FTP method transfers files over port 21 by default.

The fourth option is to copy the files over a local area network (LAN). This method is ideal if your update will be distributed within your organization, as transfer speeds are quite high. However, it cannot be used to download the configuration files from a remote site (over the Internet). To use the LAN transfer method, the client must have direct access to your network.

For this walkthrough, select the standard HTTP transfer method, and click Next.

8) Input your server's download settings, and click Next.

TrueUpdate needs to know the specifics of the server from which your client will download the configuration files.



Fill in the URL to the folder that will contain your TrueUpdate server files.

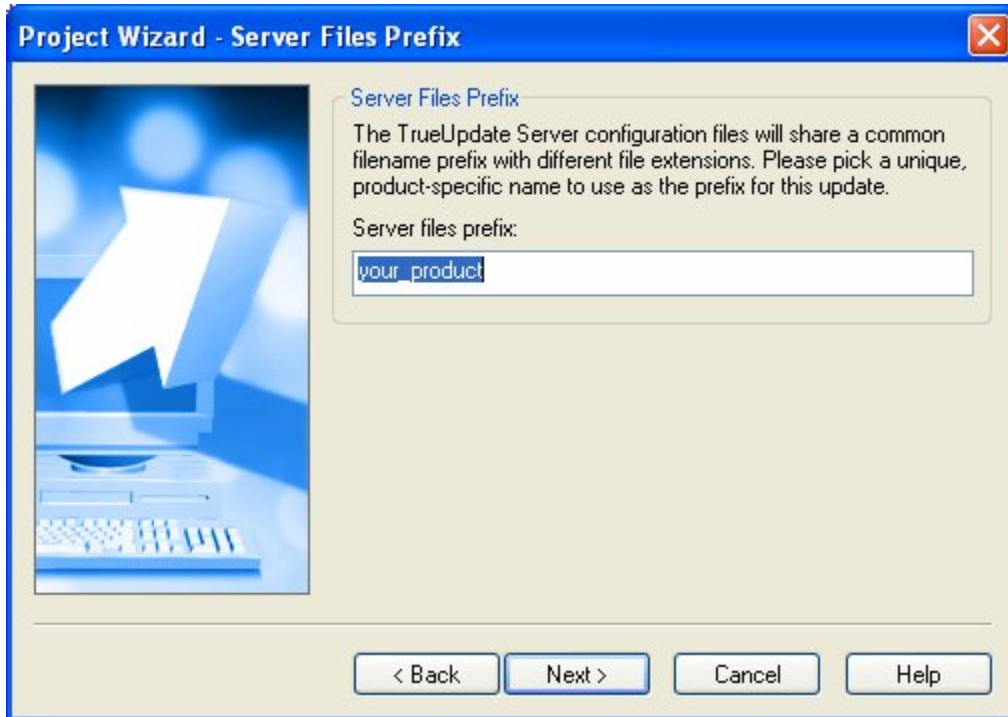
The Timeout setting refers to the amount of time in seconds that the TrueUpdate Client will wait with no reply before giving up on the server and returning an error.

The Port field allows you to specify a different communication port if your web server doesn't use port 80 (the standard HTTP port). Use this field to specify which port your server is accessed by.

You can also use basic HTTP authentication to connect to your server, if required. This allows you an additional layer of security by requiring a valid username and password in order to download the server configuration files. If your web server supports basic HTTP authentication, select the "Use authentication" option and insert the authentication username and password.

9) Specify a server files prefix, and click Next.

This step allows you to specify the prefix that your server files will use. This is a useful step if you have multiple products and would like to host your products in the same folder on your web server.



Project Wizard - Server Files Prefix

Server Files Prefix

The TrueUpdate Server configuration files will share a common filename prefix with different file extensions. Please pick a unique, product-specific name to use as the prefix for this update.

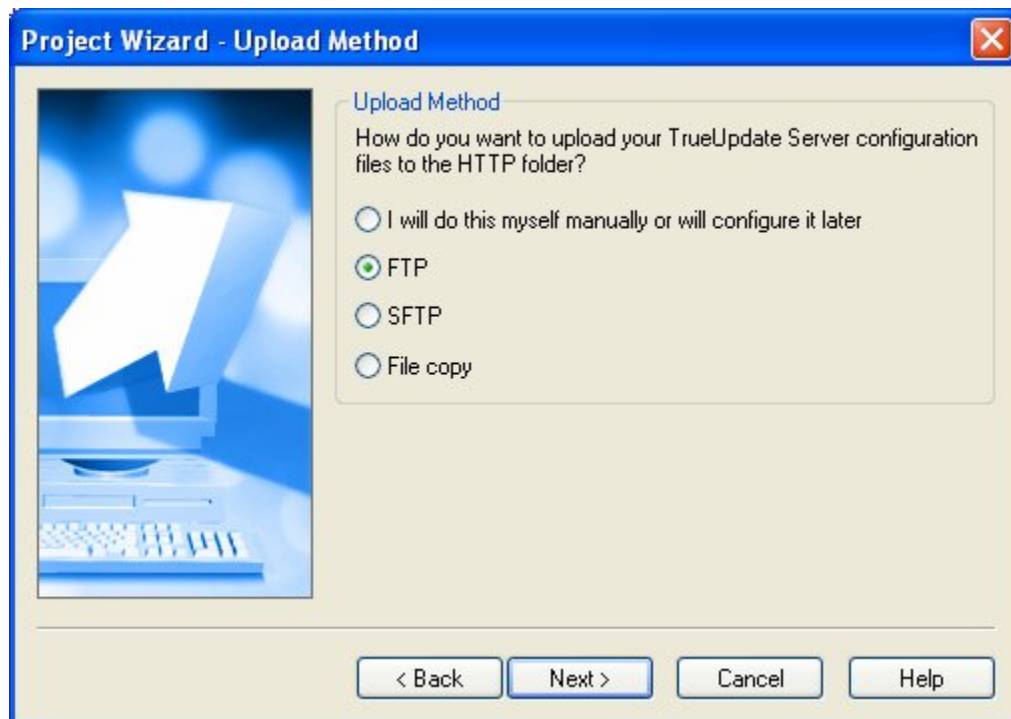
Server files prefix:

your_product

< Back Next > Cancel Help

10) Specify how TrueUpdate should transfer the files to your web server, and click Next.

For your convenience, TrueUpdate can automatically upload the server configuration files for you whenever you build your project. This feature can be a helpful time saver but is, of course, completely optional. You can always upload the configuration files to your TrueUpdate server locations on your own —by using your favorite FTP program, for example.



The first option allows you to take care of uploading the server configuration files on your own, separate from TrueUpdate, or to configure the upload automation later. (You can add automatic upload locations at any time on the Upload tab, which you can access by choosing Publish > Settings from the program menu.)

The second option will automatically transfer your files by FTP, and the third option will transfer your files by SFTP (secure FTP). These options are useful when you need to upload the files to a web server, as we will be doing in this walkthrough.

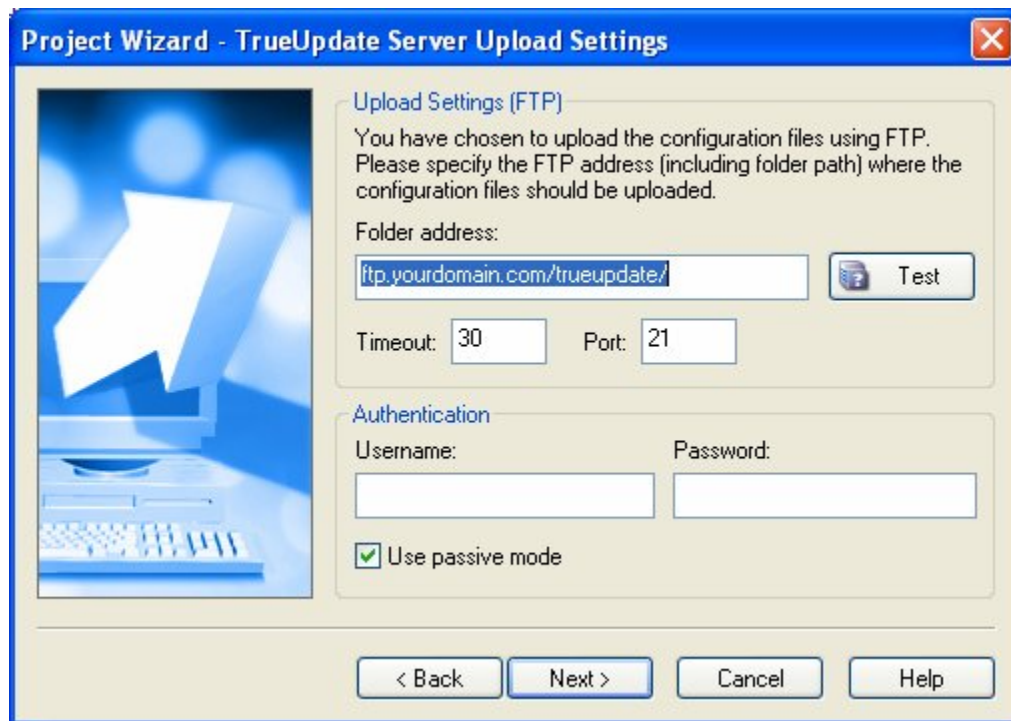
The last option performs a simple file copy and is useful if you selected LAN as your download method.

Since we chose to download the server files from an HTTP server earlier in this walkthrough, it would be convenient to have TrueUpdate upload the configuration files to that web server for us. Uploading files to a web server is normally done via FTP, so select the FTP option and click Next.

11) Configure your FTP server settings, and click Next.

This step allows you to specify where to upload the TrueUpdate server configuration files.

Note: Though we are uploading the server files via FTP, the client will access them via HTTP. Since HTTP is strictly a download protocol, we need to use a different protocol to upload the files to the web server.



Project Wizard - TrueUpdate Server Upload Settings

Upload Settings (FTP)
You have chosen to upload the configuration files using FTP. Please specify the FTP address (including folder path) where the configuration files should be uploaded.

Folder address:

Timeout: Port:

Authentication
Username: Password:

☒ Use passive mode

< Back Next > Cancel Help

In this step, you need to specify the full address of the folder where the server files need to be uploaded to. This will consist of the FTP server address and the full path to

the folder on your FTP server. You can also configure the timeout of your FTP server and the port to be used when connecting.

Lastly, you must enter the username and password that is used to access the FTP server. Be sure to use a username with appropriate access rights, as the TrueUpdate design environment will be uploading files to the server.

The passive mode option is useful when connecting to an FTP server from behind a firewall. When you connect to an FTP server normally, the server attempts to open a second connection back to your computer. If you are behind a firewall, this connection will often fail. (Most firewalls object when external systems attempt inbound connections.) Using passive mode, the FTP server will wait for your computer to open both connections, thus avoiding the objectionable behavior. The FTP server must support passive mode connections in order for this method to work, but it is a widely supported option.

Once you have entered all of your information, click Next.

12) Select a version identification method, and click Next.



Since you are creating an update for an existing product, TrueUpdate needs to know how to determine which version of your software is installed on the user's system so it can decide if a newer version is available.

There are basically three ways to determine which product version is installed. The first is to inspect the value of a registry key. This is an excellent option to choose if your installation program stored some kind of version-identifying value in the registry. For example, many installation programs create a registry key containing the version that is being installed so that other programs can easily determine the current version. This is a popular method supported by most installation tools. For example, Indigo Rose's Setup Factory allows you to easily create and modify registry keys during an installation.

The second option is to use specific file information such as the date a file was last modified. In fact, many files such as exe's and dll's contain version information that can be read using this option. This method is useful if the installer stored no additional version information (for example, if the product didn't use an installer at all). In that case, you may wish to use the version information stored in a primary file such as the software's main executable (if it has one).

The third option is to read values directly from an INI file. This is similar to reading a value from the registry, but the value is retrieved from an INI file instead. The INI file would have either been included with the product or created when the product was installed. (Setup Factory is also capable of creating and modifying INI file entries.)

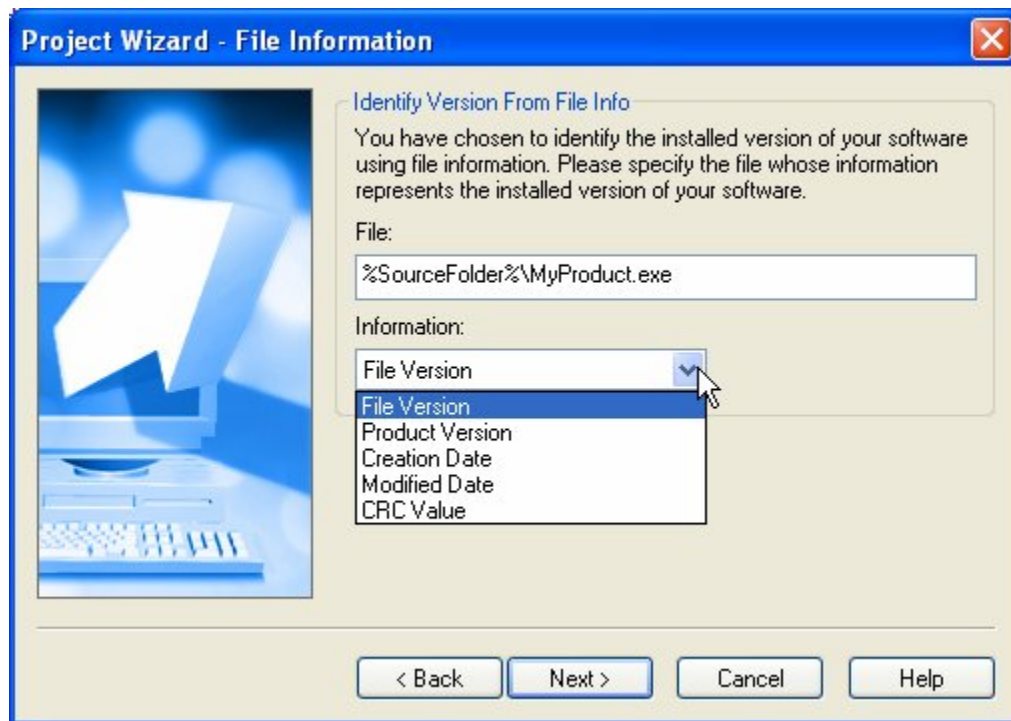
Note: Whether to store version information in the registry or in an INI file is largely a matter of preference. However, your TrueUpdate project needs to use the same method that was chosen for your installer. For example, if your installer stored the installed version in a registry key, your update won't work if you tell it to get the version from an INI file.

There is a fourth option that allows you to skip this step for now. This is useful if you would rather configure this option later, or if, for example, you want to perform the update every time the update is run, regardless of the version currently on the user's system.

For this walkthrough, select the File Information option and click Next.

13) Specify which file information will be checked, and click Next.

This step allows you to specify both which file and which attribute should be checked in order to determine which version of your software is installed.



There are five types of file information that can be checked:

File Version

The “file version” stored in the file. This is normally the version number of the file itself.

Product Version

The “product version” stored in the file. This is normally the version number of the product that the file belongs to.

Creation Date

The date and time when the file was created.

Modified Date

The date and time when the file was last modified.

CRC Value

The CRC value of the file. The CRC value serves as a digital “fingerprint” which can be compared to the CRC value of a known version of that same file in order to determine whether the two files match.

Select the Product Version option, and click Next.

14) Specify the target version, and click Next.

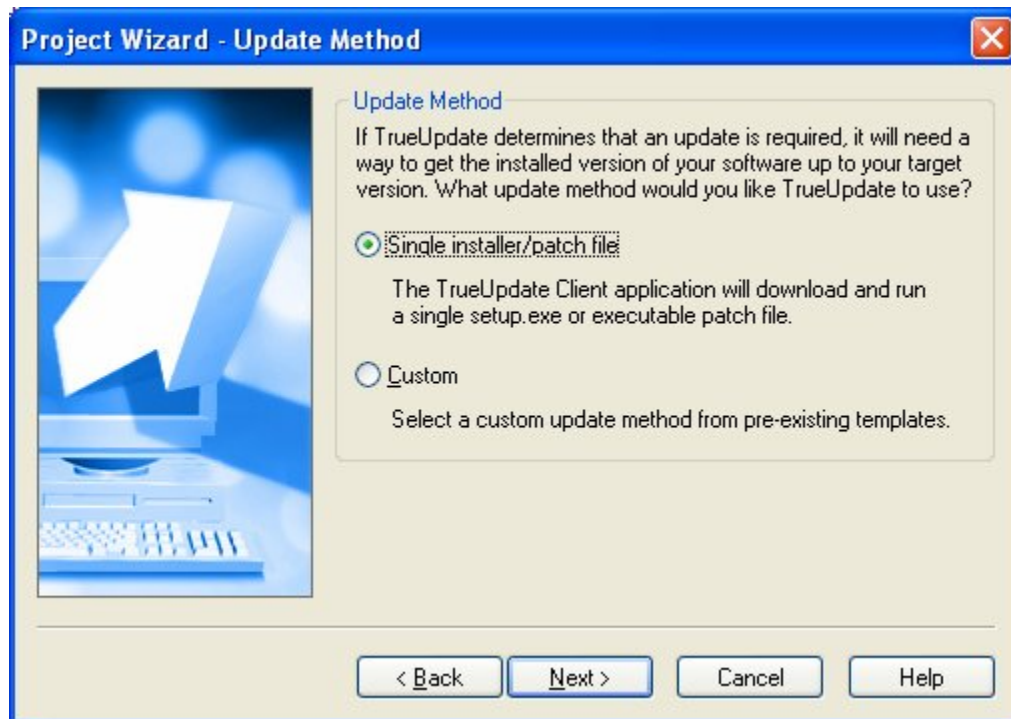
In the previous step, we decided to check a specific file for its product version. In this step, we will specify a value to compare that product version to. This essentially lets you specify the target version for your update—the version that you want to update the software to. This is the value that your TrueUpdate Client will compare the installed version to in order to determine whether a newer version is available.



You can click the Get File Version button to retrieve the version information from a specific file on your system. This allows you to quickly get the most up-to-date version information from your newest software.

15) Select an update method, and click Next.

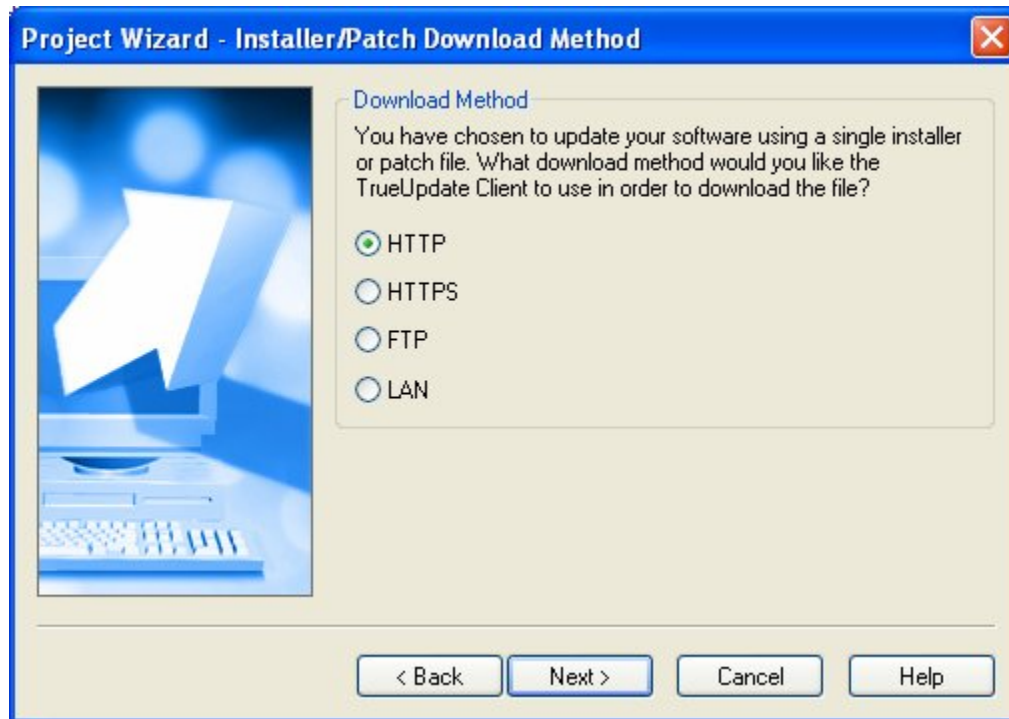
This step controls how your software will be updated when the TrueUpdate Client determines that an update is available. There are two options. The most common method is to have your update download an executable patch file (or installer) and run it. Or, if you prefer, you can select a custom update method from a list of pre-defined templates.



Select the Single installer/patch file option, and click Next.

16) Select a download method for the patch, and click Next.

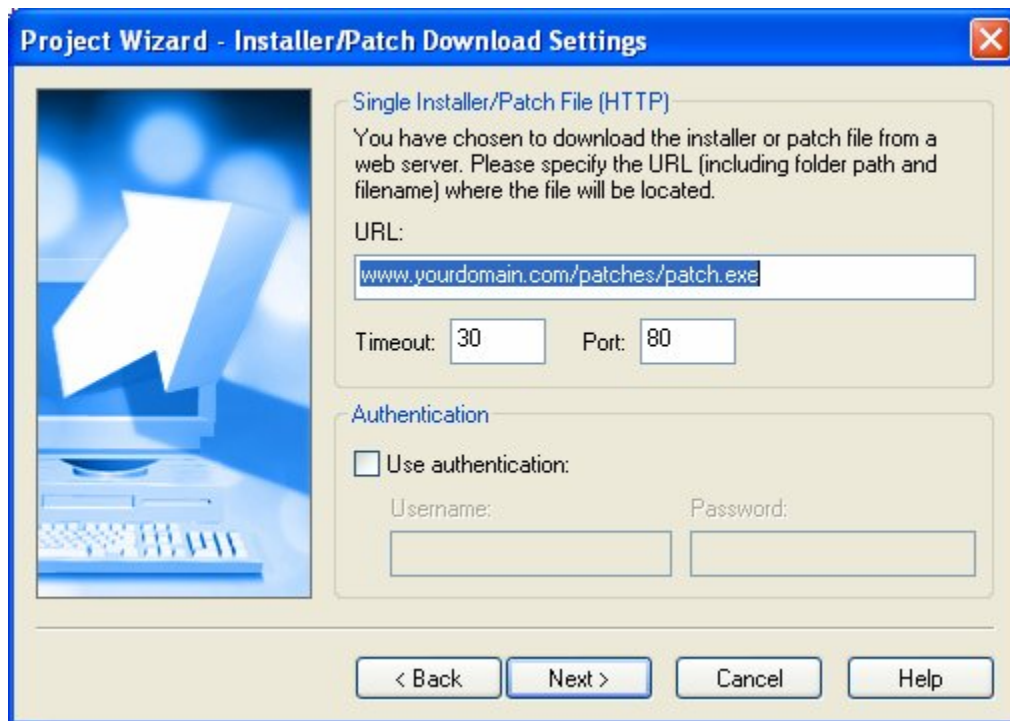
Before the client can run the installer or patch file, it needs to download it from somewhere. This step allows you to specify what method the TrueUpdate Client should use to download the file.



The same options exist as in step 7, where you specified what method to use to download the server configuration files. Select the HTTP method and click Next.

17) Specify your HTTP server settings, and click Next.

As in step 8, this step allows you to specify how the TrueUpdate Client will interact with your web server. Only this time, instead of specifying where the client can find its server configuration files, the settings determine where the client should get the executable installer or patch file that will actually update the installed software.



Project Wizard - Installer/Patch Download Settings

Single Installer/Patch File (HTTP)

You have chosen to download the installer or patch file from a web server. Please specify the URL (including folder path and filename) where the file will be located.

URL:

Timeout: Port:

Authentication

☐ Use authentication:

Username: Password:

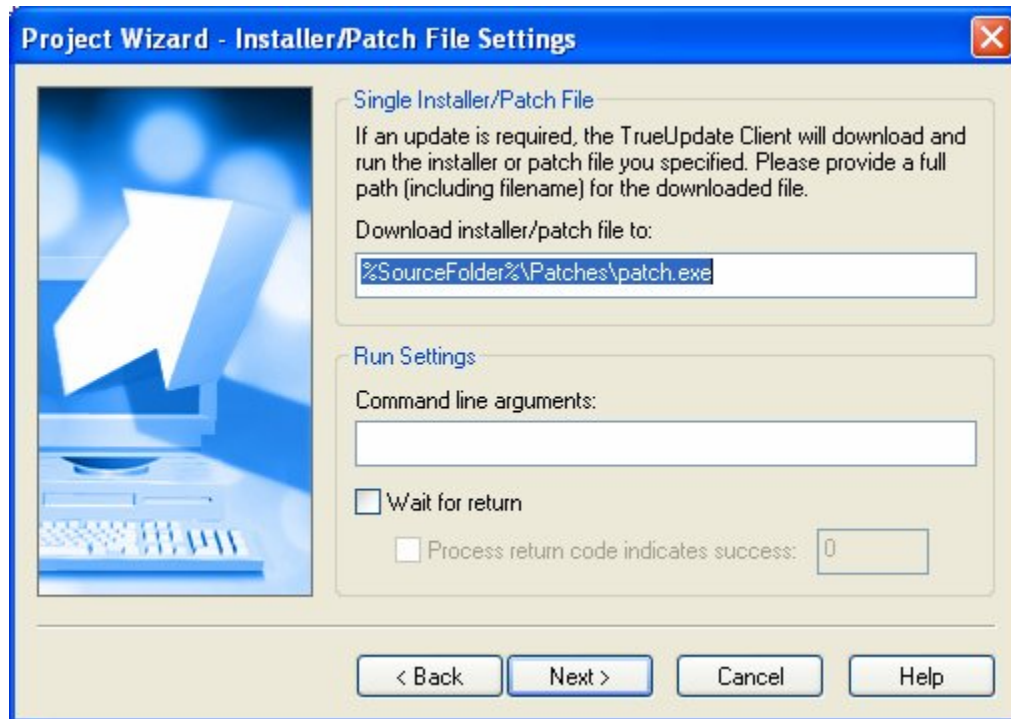
< Back Next > Cancel Help

You must specify the exact path to your patch file, as well as the timeout and port to be used. Additionally, you may use a username and password to connect to the server if it supports basic HTTP authentication.

Once you have specified how TrueUpdate will connect to your web server, click Next.

18) Specify the download location and execution options, and click Next.

This step allows you to tell the TrueUpdate client what to do with the patch once it has been downloaded.



Project Wizard - Installer/Patch File Settings

Single Installer/Patch File

If an update is required, the TrueUpdate Client will download and run the installer or patch file you specified. Please provide a full path (including filename) for the downloaded file.

Download installer/patch file to:

Run Settings

Command line arguments:

☐ Wait for return

☐ Process return code indicates success:

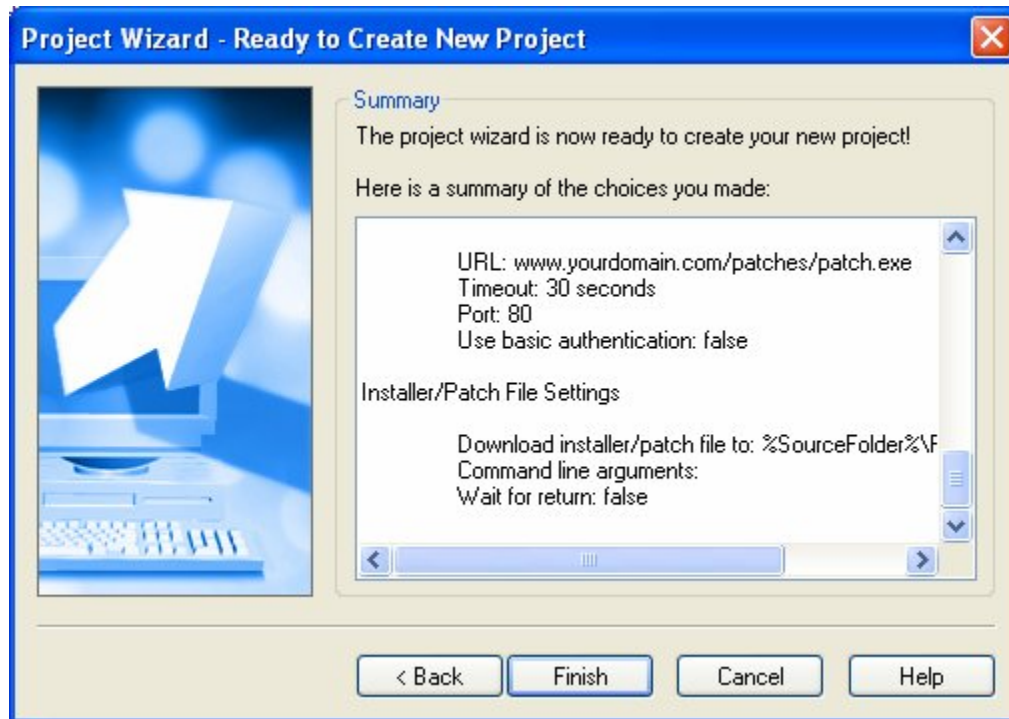
< Back Next > Cancel Help

You must specify a location to download the patch to. Be sure to specify a location that the users will have write access to, or the TrueUpdate Client will not be able to save the file. If the users do not have sufficient access rights to create a file in that folder, the download will fail.

You also have the option of specifying command line arguments to use, and whether the TrueUpdate Client should wait for the patch executable to return before continuing. Also, if your patch executable returns a result code, you can specify a return code that indicates success.

19) Review your settings, and click Finish.

This step allows you to review everything you have configured before you create the project. Review the information presented, and either move back through the wizard to make any changes, or click Finish.



After you click Finish, the project wizard will close and the design environment will appear, complete with scripts and screens configured with the settings you chose in the project wizard.



Chapter 3:

The Development Environment

This chapter will take you on a tour of TrueUpdate's sophisticated program interface. You'll learn how to use the features of the interface that allow you to create a comfortable and productive work environment, customized for the way *you* want to use the program. You'll also learn how to take advantage of TrueUpdate's self-help resources, which are designed to answer any questions you might have while working with TrueUpdate as quickly and efficiently as possible.

In This Chapter

In this chapter, you'll learn about:

- Updating TrueUpdate
- Learning the interface
- Getting help
- Setting preferences

Updating TrueUpdate

TrueUpdate has the built-in ability to check the Internet to see if there is an update available. In fact, this ability is the result of a TrueUpdate Client application, just like the ones you will build with TrueUpdate.

Before we start exploring the program interface, let's use this feature to make sure you have the latest version of the program.

1) Choose Help > Check for Update.

The TrueUpdate Client application will open.

2) Click Next.

When you click Next, the TrueUpdate Client will connect to the Indigo Rose website and determine whether there is a newer version of TrueUpdate available for you to download. If there is, it will give you the option to download and apply a patch that will update your copy of TrueUpdate to the latest version.

Note: If you are running any Internet firewall software such as ZoneAlarm, it may ask you whether to permit the TrueUpdate Client to connect. You will need to allow the client to connect in order for the update to work.

3) If an update is available, click Next and follow the instructions; otherwise, click Finish to exit.

If an update is available, click Next and follow the instructions to update your software to the latest version.

If you already have the latest version, click Finish to exit the TrueUpdate Client application.

Learning the Interface

Now that you know you are using the latest version of TrueUpdate, it's time to get comfortable with the actual program interface.

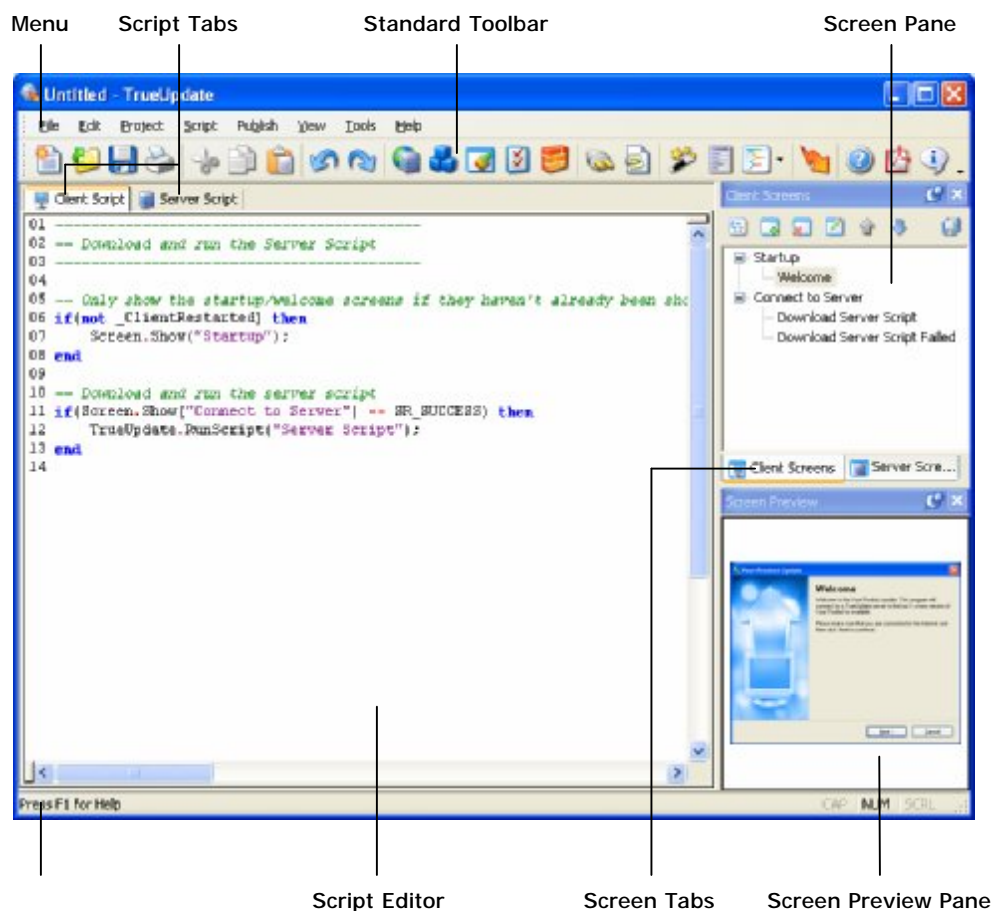
1) Explore the TrueUpdate program window.

The TrueUpdate program window is divided into a number of different parts.

At the top of the window, just under the title bar, is the program menu. You can click on this program menu to access various commands, settings and tools.

Below the program menu are a number of toolbars. The buttons on these toolbars give you easy access to many of the commands that are available in the program menu.

Tip: You can create your own custom toolbars or edit the existing ones by choosing Tools > Customize.



2) Check out the Script Editor.

The script editor takes up most of the program window. It is where you will edit the main scripts that determine what your project does. There are actually two script editors in this window—one for the client script, and one for the server script. You can switch between the client and server scripts by clicking on the respective tabs.

The script editor is a full featured code editor, including autocomplete, dropdown lists, and syntax checking.

3) Keep an eye on the Quick Help bar.

At the very bottom of the window is the quick help bar, which displays any quick help information available. For example, when working with an action in the script editor, the quick help bar will display the prototype for that action.

4) Take a peek at the panes.

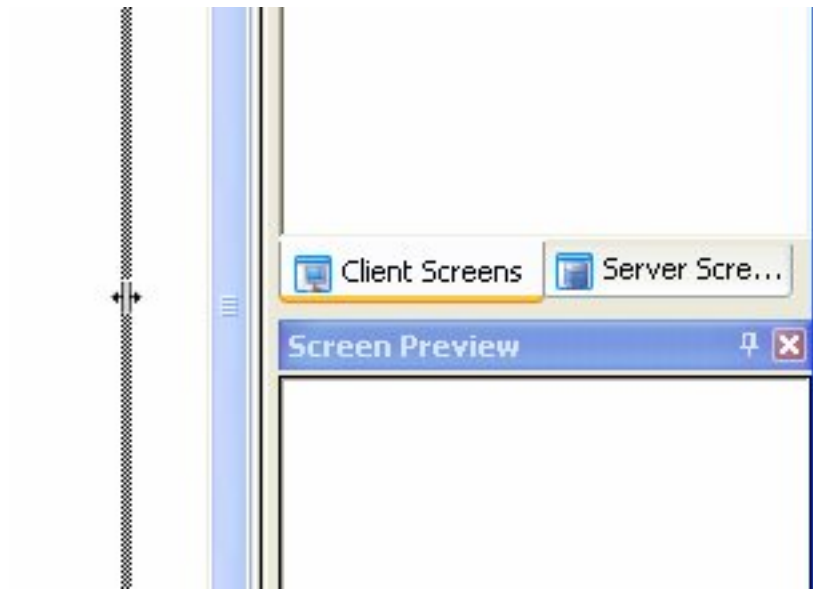
The rest of the program window is made up of individual sub-windows known as *panes*. Each pane can be docked, tabbed, pinned, resized, dragged, and even made to float on top of the design environment.

There are two panes to the right of the script editor. The top-most pane is actually a pair of panes that have been tabbed together: the Client Screens pane, and the Server Screens pane. Because these panes are tabbed by default, we often refer to them as the Client Screens tab and the Server Screens tab.

The client screens are distributed with the TrueUpdate Client application, and the server screens are stored in the server configuration files (which get hosted at your TrueUpdate Server locations). You can switch between the client screens and server screens by clicking on the appropriate tab.

Below the pair of screen panes is the Screen Preview pane. When a screen is selected on one of the screen panes, a preview of how that screen will look is shown in the Screen Preview pane.

5) Make the screen panes larger by dragging their window edges to the left.



You can resize panes by dragging their edges. In this case, you want to drag the part “between” the rightmost panes and the script editor...specifically, the little bit of window surface to the right of the script editor and to the left of the two screen panes and the Screen Preview pane. As you begin to drag the edge of a pane, a line will appear to show where the edge will move to when you release the mouse button.

You can also resize the screen and preview panes vertically by dragging the window edge between the two of them up or down. As one becomes larger, the other will become smaller, and vice-versa.

6) Double-click on the Screen Preview pane's title bar to un-dock it.

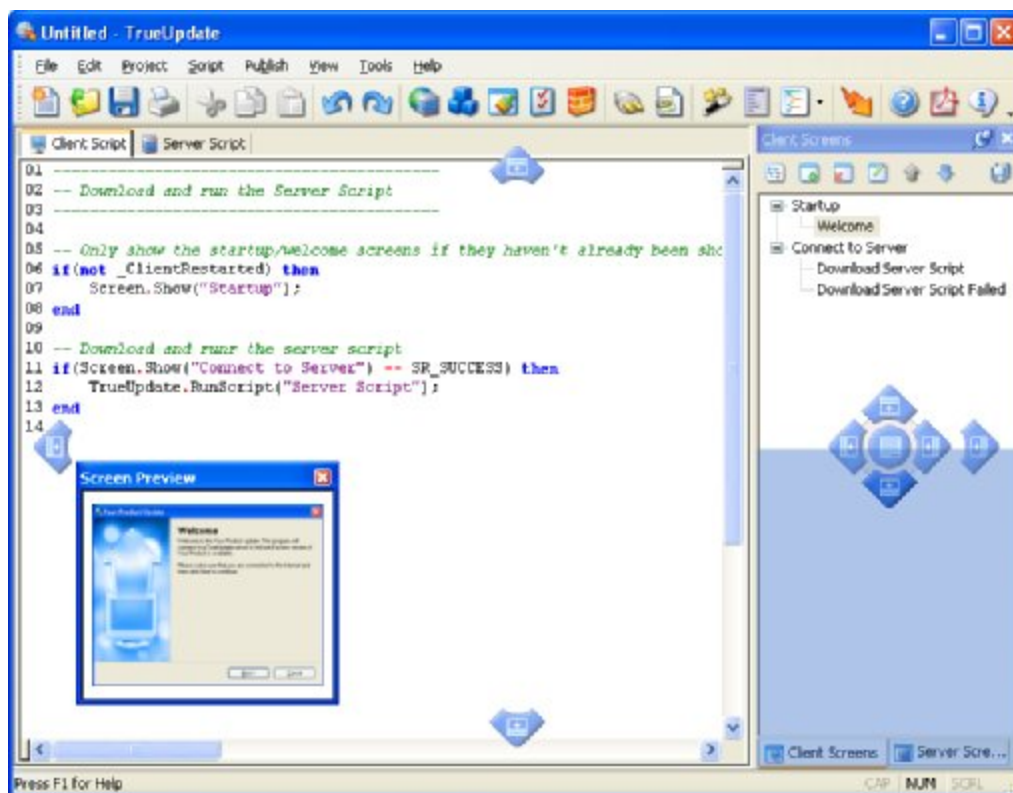
The Screen Preview pane is *docked* by default. A docked pane is seamlessly integrated into the program window. You can “un-dock” it from the program window by double-clicking on its title bar.

When you un-dock the Screen Preview pane, it floats above the program window, and the screen panes expand to fill in the space that the Screen Preview pane left behind.



7) Drag the Screen Preview pane to the bottom of the screen panes to dock it again.

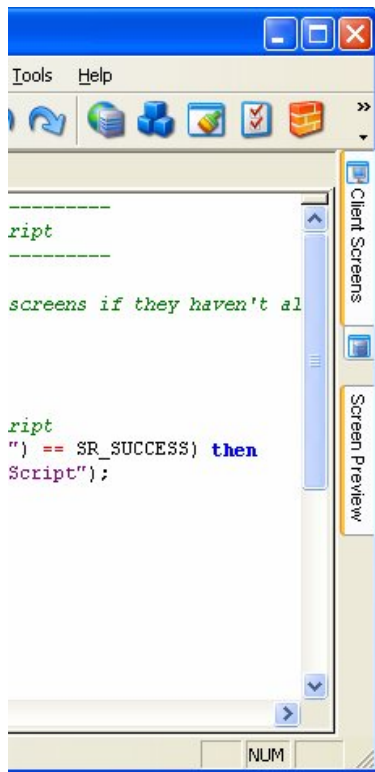
You can move panes around by dragging them by their title bars. As you move a pane, an outline shows you the general area where the panel will end up, and special 'drop points' appear to control where the pane will be docked. We want to put the Screen Preview pane back where it was originally. Drag the Screen Preview pane by its title bar around the design environment until the drop points appear. Then, drop it on the point that will re-dock it below the screen panes.



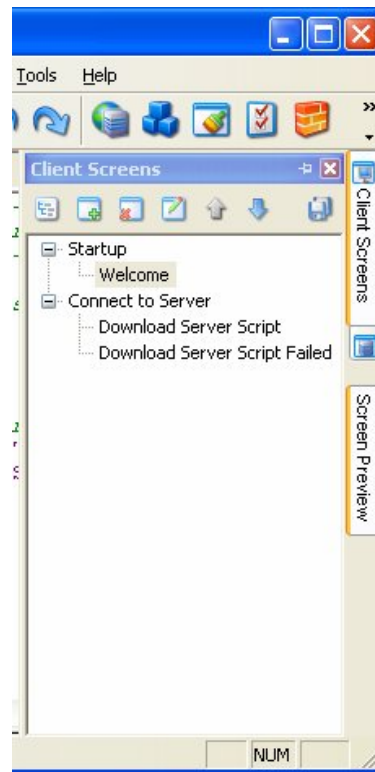
Tip: When you are dragging panes, the position of the mouse cursor is what determines where the pane will land. When moving panes to the drop points, the mouse pointer must be over the point for the screen to dock as you intend.

8) Unpin the Screen Preview pane, and then pin it again.

Docked panes can also be “pinned” or “unpinned.” *Pinned* panes remain open when you’re not using them. (All of the panes in the default layout are pinned.) *Unpinned* panes stay out of the way until you click on them or hover the mouse over them. Whenever you need them, they “slide” open on top of everything else and then slide closed when you’re done.



The Client Screens pane unpinned...



...and after "sliding" open

You can pin or unpin a pane by clicking the little pin icon on the pane’s title bar.

Panes remember their positions even after you unpin them. If you unpin a pane, and then pin it again, it will return to the position it had before it was unpinned.

9) Turn on the Script Help toolbar.

Right clicking on the standard toolbar will bring up the list of available toolbars. Notice that only the standard toolbar is enabled by default. Click on ‘Script Help’ to enable the Script Help toolbar. This toolbar displays useful tips and information, and any other helpful tools relating to scripting.

You can turn on as many toolbars as you like, or disable them all. You can drag and drop toolbars around the interface similar to how you dragged panes around. Dragging a toolbar to the edge of the TrueUpdate window will dock it to that side. Dragging a toolbar to the middle of the TrueUpdate window will make the toolbar float above the interface.

Feel free to customize the design environment as you see fit. TrueUpdate is your tool, so go ahead and arrange it to fit the way you work.

Getting Help

If you still have questions after reading this user’s guide, there are many self-help resources at your disposal.

Here are some tips on how to quickly access these self-help resources.

1) Press the F1 key.

Help is only a key press away! TrueUpdate comes with an extensive online program reference with information on every action and feature in the program.

In fact, pressing F1 will, whenever possible, bring you directly to the appropriate topic in the online help. This context-sensitive help is an excellent way to answer any questions you may have about a specific dialog or action.

Note: You can also access the help system by choosing Help > TrueUpdate Help.

There are three ways to navigate the online help system and find the appropriate topic: use the table of contents, use the keyword index, or search through the entire help system for a specific word or phrase.

2) Close the help window and return to the TrueUpdate design environment.

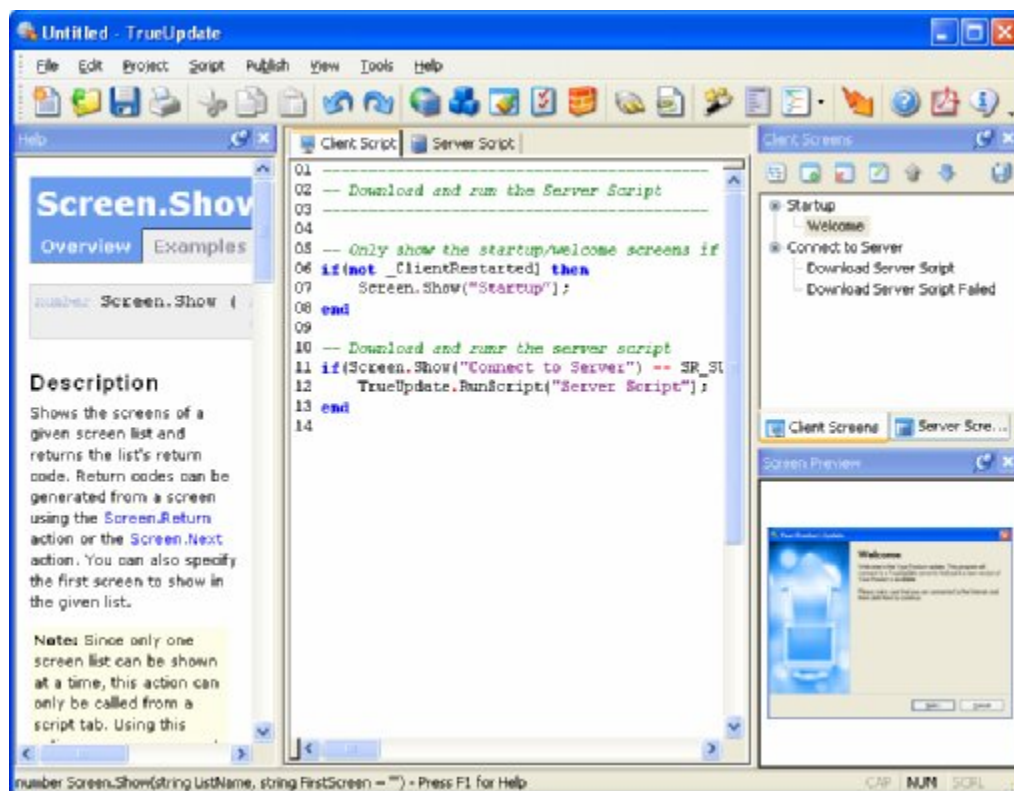
To exit from the online help, just click the Close button on the help window’s title bar.

3) Choose View > Help to open the Help pane.

You can also open the help file inside a pane, right in the TrueUpdate design environment. This is especially useful when you're working on scripts and want quick access to the help topics for the actions you're working on.

4) Select the Client Script tab, and click on the Screen.Show action.

When you click on an action in the script editor, notice that the Help pane automatically displays the help topic for that action.



As you are working on your scripts, the Help pane will automatically display the help topic for any action that you click on. It will also automatically display the help topic for an action as you're typing it into the script editor. (In other words, if you typed "Application.Exit" into the script editor, the Help pane would automatically show you the help topic for the Application.Exit action.)

5) Choose View > Help again to close the Help pane.

Choosing View > Help will close the Help pane if it is currently open.

6) Choose Help > User Forums.

TrueUpdate is used by thousands of people worldwide. Many users enjoy sharing ideas and tips with other users. The online forums can be an excellent resource when you need help with a project or run into a problem that other users may have encountered.

Choosing Help > User Forums opens your default web browser directly to the online user forums at the Indigo Rose website.

7) Close your web browser and return to the TrueUpdate design environment.

Exit from your web browser and switch back to the TrueUpdate program.

Alternatively, you can press Alt+Tab to switch back to TrueUpdate while leaving the web browser open in the background.

8) Choose Help > Technical Support > Support Options.

This takes you to the TrueUpdate web site, where a variety of online technical support resources are available to you, including a large knowledge base with answers to common questions.

This is also where you can find information about submitting a support request.

9) Close your web browser and return to the TrueUpdate design environment.

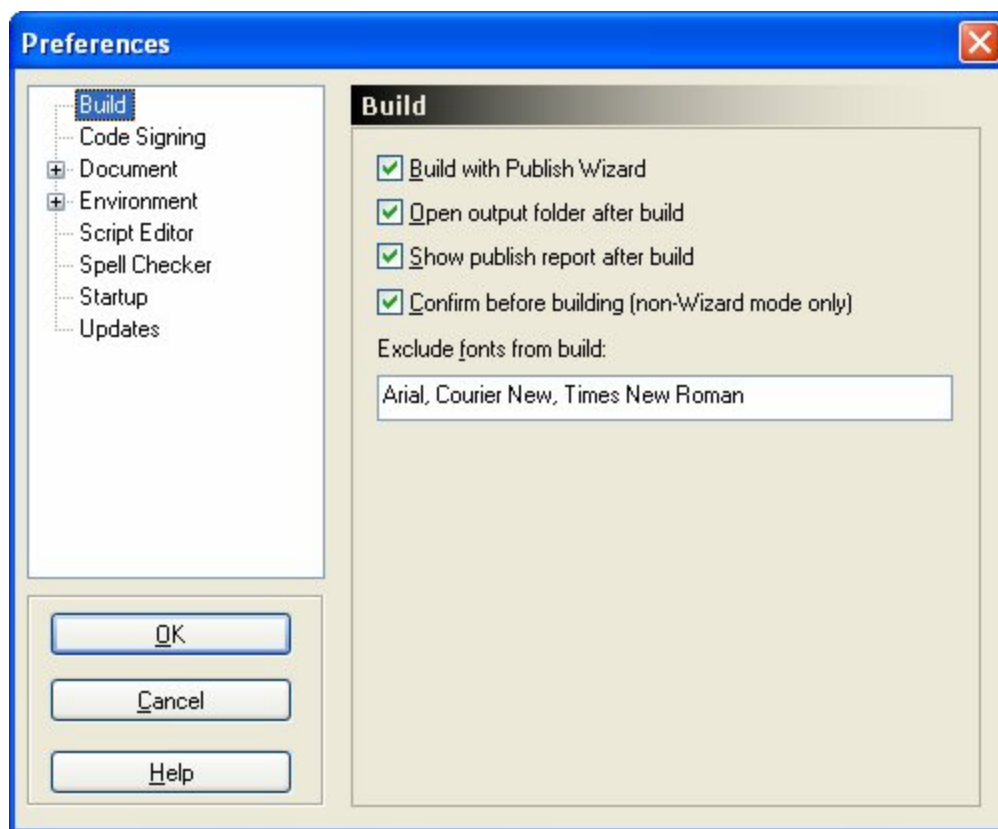
When you're done browsing the technical support information, return to the TrueUpdate design environment to continue with this chapter.

Setting Preferences

There are a number of preferences that you can configure to adjust the TrueUpdate design environment to suit your work style. Let's have a look at some of them.

1) Choose Edit > Preferences.

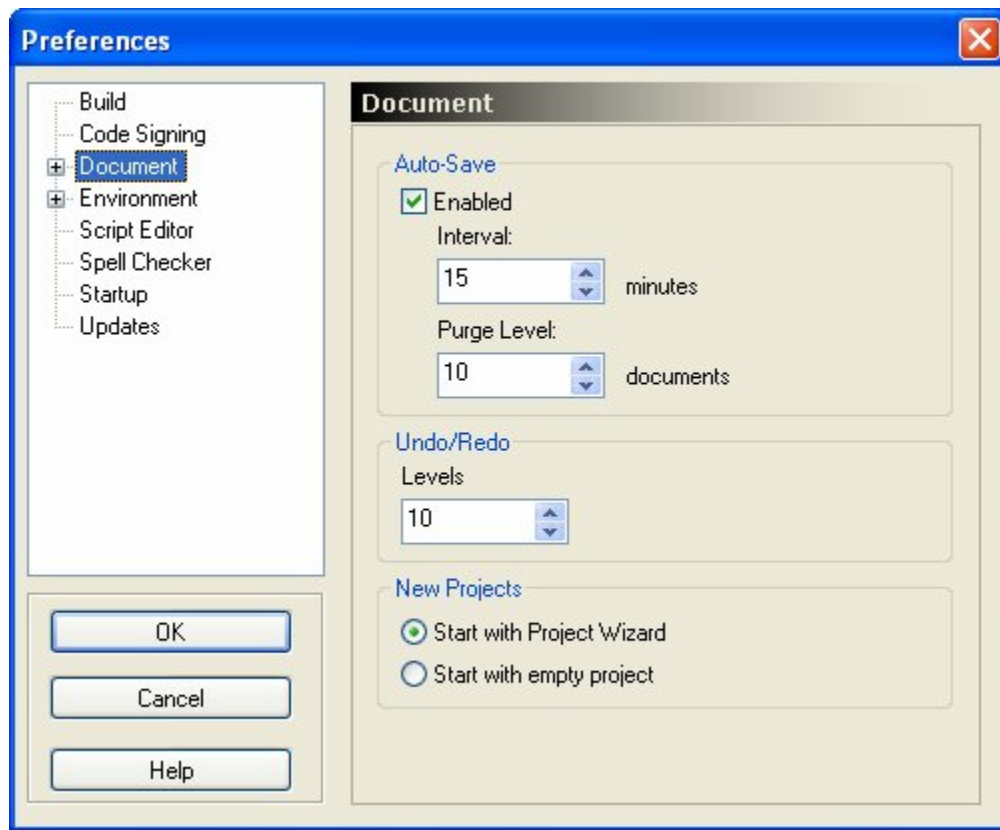
This will open the Preferences dialog, where all of TrueUpdate's preferences can be found.



The preferences are arranged into categories. The categories are listed on the left side of the dialog. When you click on a category, the corresponding preferences appear on the right side of the dialog.

2) Click on the Document category.

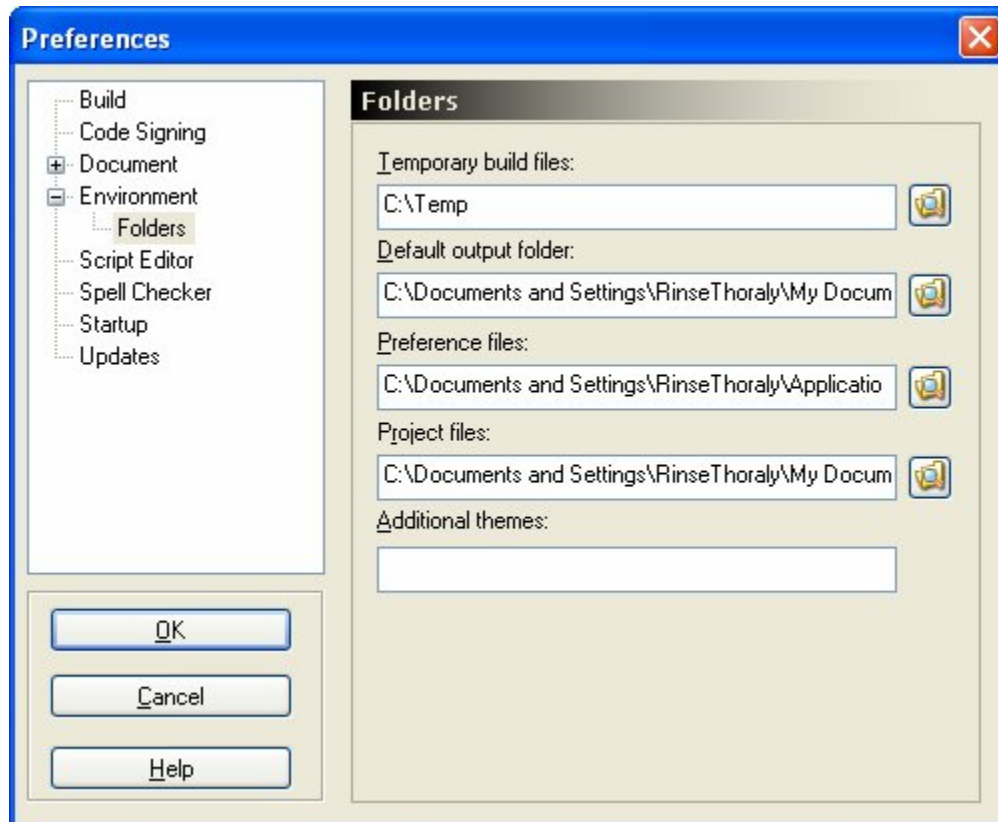
The Document preferences allow you to change settings that affect the project file. For example, you can configure the auto-save feature that can automatically save your project file as you're working on it to avoid any accidental loss of data. You can also configure the number of undo/redo levels, and choose either to use the project wizard to create new projects or to simply start with a blank project.



Tip: It can be helpful to set the number of undo levels to a larger value, like 25 or 50. That way you can undo even more “steps” and go back further in your project if you change your mind while you’re working.

3) Expand the Environment category by double-clicking on it, and then click on the Folders category.

You can also expand the Environment category by clicking on the little plus symbol to the left of it.



The Folders category allows you to specify the locations of various folders that are used by the project. For example, you can specify the location where project files are stored, and specify the default output folder where your update files will be published to.

4) Feel free to explore some of the other categories. When you're done, click OK to close the Preferences dialog.

There are many other preferences that you can set, such as what to do when the design environment is started (in the Startup category). Take some time to look through the categories and familiarize yourself with the different options that are available.

Remember that you can click Help or press F1 to get more information about any of the settings in a specific category.

Chapter 4:

Introduction to Scripting

Scripting is the art of creating and modifying a sequence of actions that work together to perform a specific task.

Actions are among the most important features in TrueUpdate. They are what you use to control the functionality of your update, with virtually limitless possibilities. Each action is a specific command that tells your TrueUpdate Client to do something, such as downloading a file, showing a screen, or modifying the Registry.

Actions also allow your update to react to different situations in different ways. Does the user have your software installed? Is an Internet connection available? You can use actions to answer these types of questions and have your TrueUpdate Client respond accordingly.

In this chapter, we'll introduce you to the basics of scripting in TrueUpdate.

In This Chapter

In this chapter, you'll learn about:

- What scripts are
- What actions are
- The script editor (including features such as syntax highlighting, intellisense, quick help, and context-sensitive help)
- The Client Script tab
- Server Script tabs
- Screen events
- Using the action wizard
- Including external script files
- Extending TrueUpdate with action plugins

What Are Scripts?

A script is a sequence of actions that work together to perform a specific task. It can include a single action, or any number of actions. The simplest scripts comprise a basic series of steps, with one action following another in a direct and straightforward manner. More advanced scripts take advantage of control structures like “if” and “while” to incorporate sophisticated techniques such as decision-making, branching and looping.

Scripts are ultimately just text documents that follow a specific syntax. The actions and control structures in a script are represented by specific keywords, which the TrueUpdate Client ultimately interprets as instructions that tell it what to do.

Note: Scripts are a key component of TrueUpdate. For example, the entire client-server interaction is defined by the script that is entered on the Client Script and Server Script tabs.

What Are Actions?

Actions are what you use in TrueUpdate when you want to get something done. They are specialized commands that your TrueUpdate Client can perform at run time. Each action is a short text instruction that tells the client to do something—whether it’s to download the server configuration files, show a screen, open a document, or modify a registry key.

Actions are grouped into categories like “File” and “Registry.” The category and the name of the command are joined by a period or “dot,” like so: File.Run, Registry.GetValue. The text “File.Run” essentially tells TrueUpdate that you want to perform a “Run” command from the “File” category...a.k.a. the “File.Run” action.

Tip: The period in an action name is either pronounced “dot,” as in “File-dot-Open,” or it isn’t pronounced at all, as in “File Open.”

It is worth noting that the many screen types available to you in TrueUpdate actually use actions to accomplish their tasks. In fact, if you want to modify a screen’s built-in functionality, you can do so simply by editing the screen’s actions. Because of TrueUpdate’s scripting engine, the power is in your hands!

The Script Editor

The script editor is where you create and edit your scripts in the TrueUpdate design environment. It essentially functions like a text editor, allowing you to type the actions and other scripting elements that you want to use.

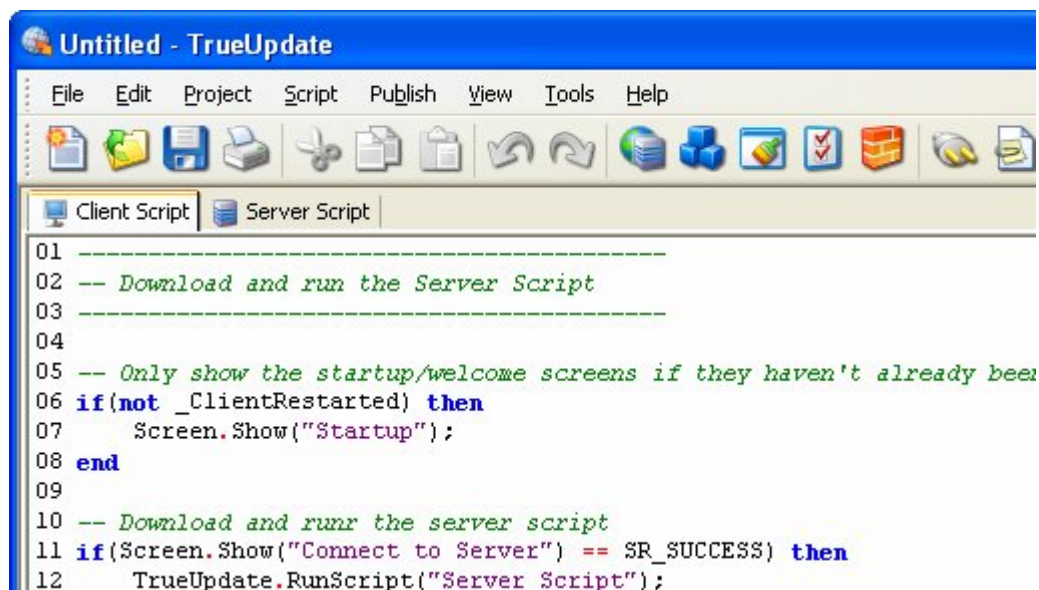
However, while it may function like a text editor, the script editor has powerful features that make it a full-fledged programming environment. Some of these features include syntax highlighting, intellisense, quick help and context-sensitive help.

It also has a built-in action wizard, which provides an easy dialog-based way to select, create, and edit actions without having to type a line of script.

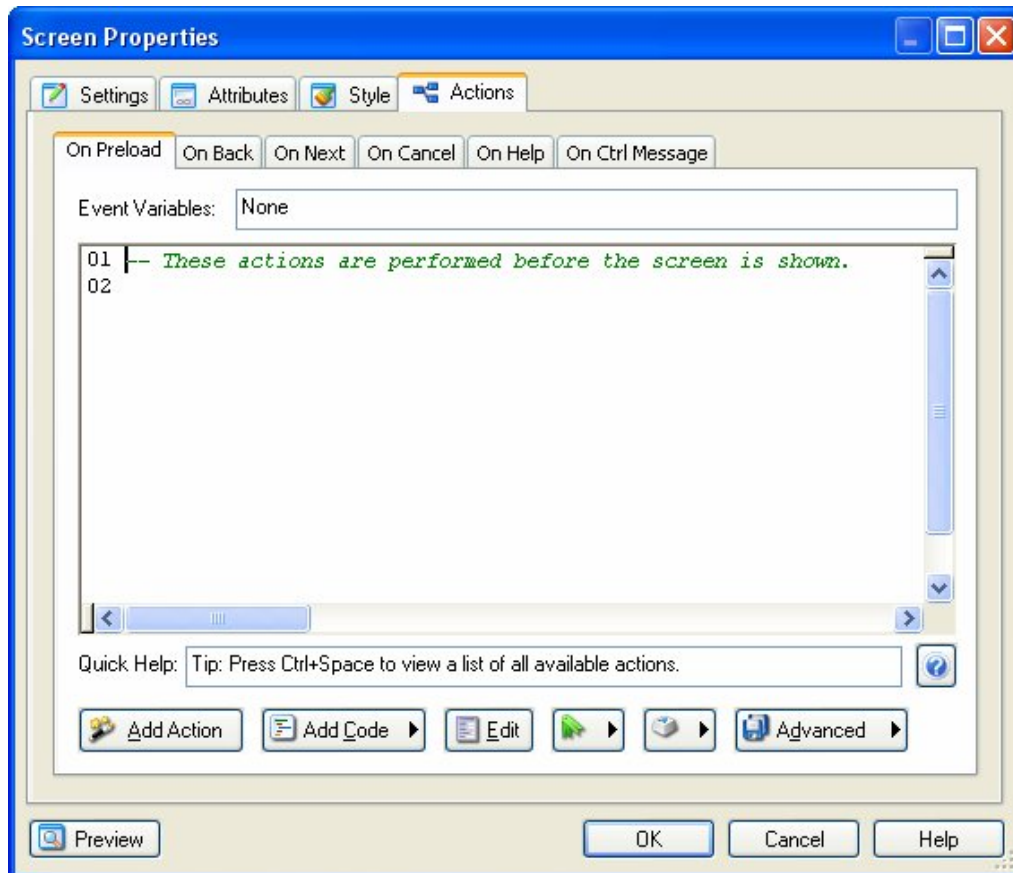
In short, the script editor gives you the best of both worlds: pure scripting capabilities for advanced users and programmers, and an easy-to-use action wizard interface for those who'd rather not use free form scripting.

Script Tabs and Screen Events

The script editor has two visible interfaces, depending on whether you are adding actions to script tabs or screen events.



The script editor on the Client Script tab



The script editor on a screen event tab

The script editor is functionally the same in both places; the only difference is in how you access some of its programming features. The screen event tabs have built-in buttons that allow you to access the most common features of the script editor. The script tabs have no integrated buttons; instead you can access the features via the program menu and toolbars. In both cases, however, you can access the features by using the right-click menu.

Programming Features

The script editor provides many powerful features that make it a useful and accessible tool for programmers and non-programmers alike. Along with the action wizard (covered later under *Using the Action Wizard*), the four most important features of the script editor are: syntax highlighting, intellisense, quick help, and context-sensitive help.

Syntax Highlighting

Syntax highlighting colors text differently depending upon syntax. This allows you to identify script in the script editor as an operator, keyword, or comment with a quick glance.

Note: You can customize the colors used for syntax highlighting via the script editor settings. The script editor settings are accessed on screen event tabs via the advanced button: Advanced > Editor Settings. On script tabs it can be accessed from the program menu by choosing Edit > Advanced > Editor Settings.

Intellisense

Intellisense is a feature of advanced programming environments. It refers to the surrounding script and the position of the cursor at a given moment to provide intelligent, contextual help to the programmer.

Intellisense is a term that has been used with various intended meanings. In TrueUpdate, intellisense manifests itself as two features: autocomplete, and the autocomplete dropdown.

Autocomplete is the editor's ability to automatically complete keywords for you when you press Tab. As you type the first few letters of a keyword into the script editor, a black tooltip will appear nearby displaying the whole keyword. This is the intellisense feature at work. Whenever you type something that the intellisense recognizes as a keyword, it will display its best guess as to what you are typing in one of those little black tooltips. Whenever one of these tooltips is visible, you can press the Tab key to automatically type the rest of the word.

```

01 -----
02 -- The purpose of the Server Script is to detect the installed
03 -- version, decide if an update is available, and then perform
04 -- the update if required.
05 -----
06 Fi
  File

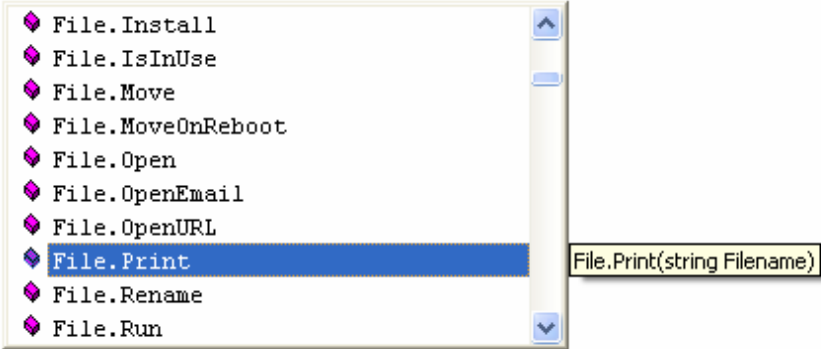
```

Another feature of intellisense is the autocomplete dropdown. By pressing Ctrl+Space while your cursor is in the code window, you can make a drop-down list appear containing the names of all the available actions, constants and global variables. Choosing one of the listed items and pressing Tab or Enter will automatically type the item into the script editor for you.

```

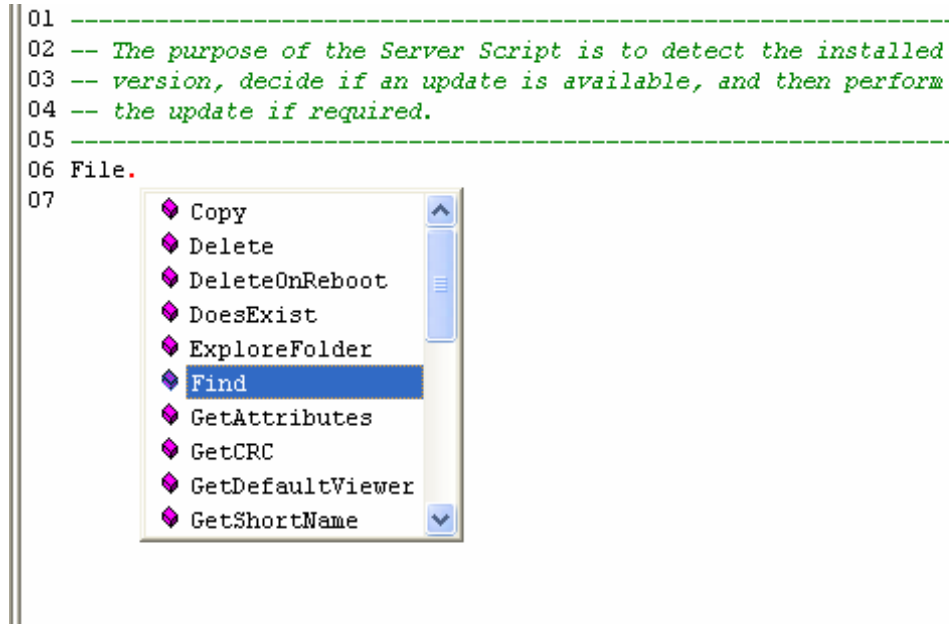
02 -- The purpose of the Server Script is to detect the installed
03 -- version, decide if an update is available, and then perform
04 -- the update if required.
05 -----
06

```



Note: This dropdown cannot be accessed if your cursor is inside a set of quotes (a string).

The autocomplete dropdown is also available for completing action names after the category has been typed. For example, when you type a period after the word File, the intellisense recognizes what you've typed as the beginning of an action name and presents you with a drop-down list of all the actions that begin with "File."



The word will automatically be typed for you if you choose it and then press Tab or Enter. However, you don't have to make use of the dropdown list; if you prefer, you can continue typing the rest of the action manually.

Quick Help

Once you've typed something that the script editor recognizes as the name of an action, quick help is automatically displayed. Quick help is essentially a "blueprint" for the action. It lists the names of the action's parameters, and indicates what type of value is expected for each one. For example, in the case of a Screen.Jump action, the quick help looks like this:

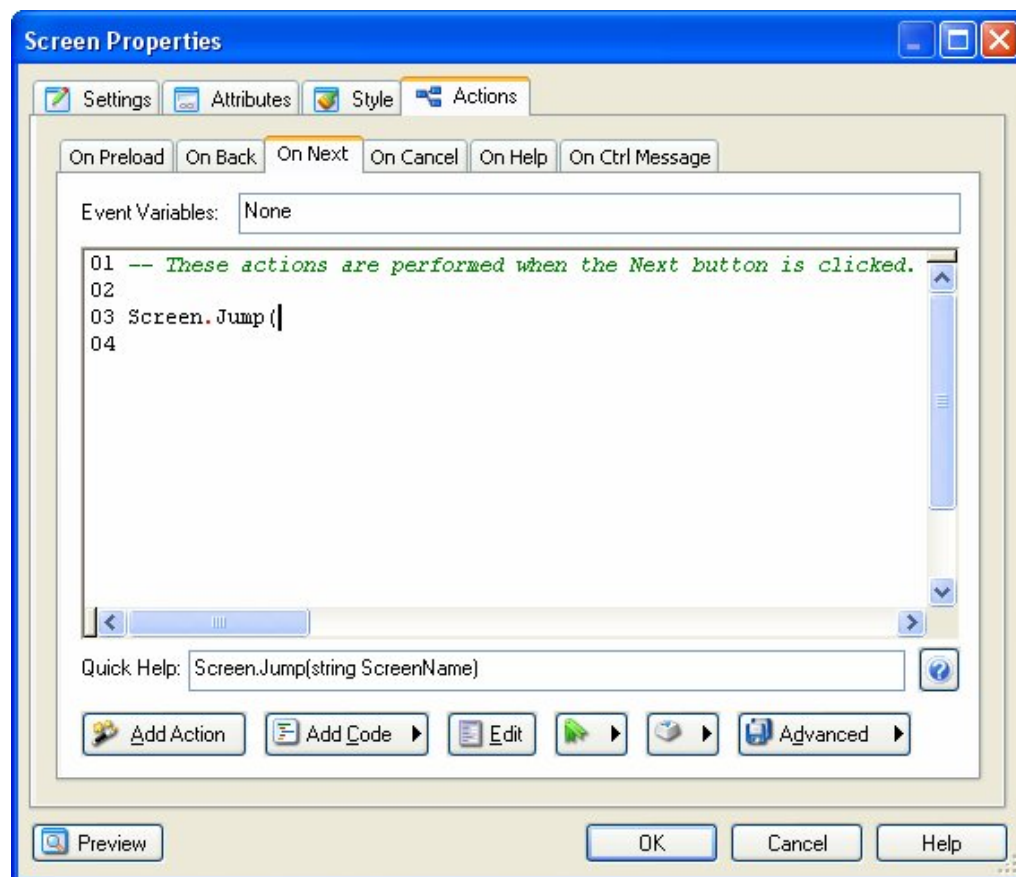
```
Screen.Jump(string ScreenName)
```

This quick help indicates that the Screen.Jump action takes a single parameter called ScreenName, and that this parameter needs to be a string. Strings need to be quoted,

so if you wanted to jump to a screen named Finish, the full action would have to be typed exactly like this:

```
Screen.Jump("Finish");
```

On screen events, the quick help is located towards the bottom of the properties window. When you're working on script tabs, it can be found on the status bar at the bottom of the program window, or on the optional Script Help toolbar. (You can enable the Script Help toolbar by choosing View > Toolbars > Script Help.)



Script help on a screen event tab

Context Sensitive Help

Context sensitive help, as its name suggests, provides help for you based upon what you are currently doing. In the script editor, the context sensitive help lets you jump directly to the current action's topic in the help file.

For instance, if you are typing an action into the script editor and the quick help feature isn't giving you enough information (perhaps you would like to see an example), press the F1 key and the help file will open directly to that action's help topic.

Note: The context sensitive feature is only available when the script editor recognizes the action that the cursor is on. It is easy to know when this is the case; when the script editor recognizes an action, the action's template appears in the quick help.

Client Script

The Client Script tab is a script editor located on the main development window. It contains the script that runs as soon as the TrueUpdate Client application is started. This script is generally referred to as the client script.

Generally the client script should contain any actions needed to initialize and initiate the update process.

A typical client script will introduce the user to the update and then download the server configuration files. In fact, the main function of the client script is to download and run the server script, which is contained in the server configuration files.

Tip: While you're working on the script tabs, you can access all of the script editor features under the Edit and Script program menus.

Server Scripts

The Server Script tab is another script editor located on the main development window. It contains the script that is hosted at the TrueUpdate Server locations. This script is generally referred to as a server script.

The purpose of a server script is to determine whether an update is available, and to actually perform the update—either by running a series of actions, or by launching a separate installer or patch file.

Multiple Server Scripts

TrueUpdate actually supports multiple server scripts, each represented by a separate tab on the program window. Every project begins with a single “main” server script, represented by the Server Script tab.

For larger projects, you may want to divide your actions into multiple server scripts in order to make the project easier to maintain. TrueUpdate allows you to add as many additional server scripts as you need in order to facilitate organizing your code.

The server scripts that you add are treated just like the main server script. In fact, the contents of all the server scripts are stored in the same server configuration file, and they are all downloaded together by the `TrueUpdate.GetServerFile` action. Just as you would run the main server script, you can run any individual server script using the `TrueUpdate.RunScript` action.

Tip: Use multiple server scripts to implement “branching” in different situations. For example, if your TrueUpdate Client needs to perform different steps in order to update different versions of your software, you could create a separate server script for each version, and then use the `TrueUpdate.RunScript` action to call the appropriate script tab from your main server script once the installed version is detected.

Designed For Easy Modification

The server scripts are stored together in one of the server configuration files, which is downloaded by an action in the client script. Because they are downloaded each time your TrueUpdate Client connects to a TrueUpdate Server, you are free to modify the server scripts at any time—even after the TrueUpdate Client has been distributed.

This separation between the client and the server scripts is an important feature of TrueUpdate. It allows you to adjust the update process for your software at any time, without any modifications to the client at all. In other words, it allows you to modify your update process without having to redistribute new clients to your users.

Note: The server scripts are normally what you will modify each time a new version of your software is released (or whenever you need to update your software).

Screen Events

Every screen has a number of events associated with it. These events are “triggered” when something happens to the screen. For example, every screen has an `On Preload` event that is triggered just before the screen is displayed.

For every event that a screen supports, there is a separate tab on the screen's properties dialog. Each tab contains a script editor, where you can edit the script that will be executed when the corresponding event occurs.

When you add an action to a screen's event tab, you are in effect adding the action to that event. At run time, when the event is triggered, all of the actions on the event tab are performed in sequence from top to bottom. In other words, TrueUpdate "runs through" the script for that event. This happens each time the event is triggered.

For example, to make something happen immediately before a screen is displayed, you would add an action to the screen's On Preload event. To make something happen when a screen's Next button is clicked, you would add an action to the screen's On Next event.

Screen events are triggered either by the controls on the screen or by the screen itself. They can be used for navigation (e.g. moving to the next screen), to change what is displayed on the screen (e.g. updating progress text), or to perform the task required of the screen (e.g. downloading a file).

Note: A control is an "object" on a screen that serves a specific purpose, such as receiving mouse clicks, displaying text, or providing a checkbox. Most screens contain several controls. You can think of the controls as the different screen "parts" that allow the screen to do what it does. For more information on controls and which controls exist on the various screen types, please consult the help file.

Using the Action Wizard

The action wizard is a dialog-based way for you to add actions in the script editor. It is designed to guide you through the process of selecting your action and configuring its parameters.

Tip: Even if you prefer typing actions directly into the script editor, don't disregard the usefulness of the action wizard. It is an excellent way to add an action when you aren't familiar with its parameters and the range of values it accepts.

While in a script editor, you can launch the action wizard by pressing Ctrl+W, or by right-clicking and choosing Action Wizard from the context menu.

Note: The action will be added at the current location of the cursor, so take care where you click or right-click before launching the action wizard.

Adding Actions

Here is a brief example that shows how easy it is to add an action. It illustrates how to display a popup dialog with a message on it by adding a `Dialog.Message` action to the Client Script tab.

1) Start a new project and click Cancel when the project wizard appears. Make sure the Client Script tab is selected.

To start a new project, either start TrueUpdate and click “Create a new project” on the welcome dialog, or choose `File > New Project` from the program menu. When the project wizard appears, click Cancel to skip the wizard and go straight to the design environment.

Note: Normally you will want to use the project wizard in order to base your project on a completed structure; however for this example it is easier to start with an almost blank project.

Make sure you have the Client Script tab selected and not the Server Script tab. (You can select the Client Script tab by clicking on it.)

2) Comment out the Screen.Show action by adding two dashes (--) at the start of the line.

Before we continue, let’s disable the default client script’s `Screen.Show` action so it won’t interfere with this example.

Any line in a script that begins with two dashes is treated as a comment. Comments aren’t executed; they are typically used to add helpful notes and explanations to your code. For example, you’ll notice that the first four lines of the Client Script already have comments describing what the Client Script does.

When you add two dashes to the start of the line that has the `Screen.Show` action on it, the line turns green, the default color for comments in the script editor. Adding the two dashes turned the line into a comment. This effectively causes TrueUpdate to ignore that line completely.

You can add two dashes to the start of any line of code to “turn it off” temporarily. Turning a line of code into a comment is known as “commenting out” the code.

Tip: You can “uncomment” a line that is currently commented out by simply deleting the two dashes. Removing the dashes reverts the line to its former “active” status; it is no longer a comment.

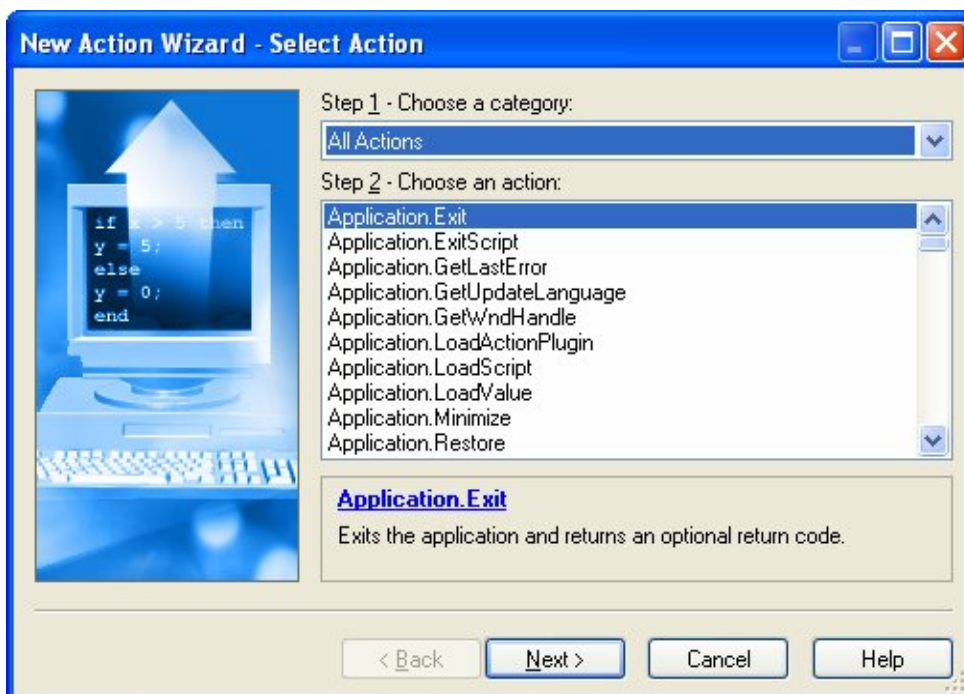
Here is how the Client Script should look when you're done:

```
Client Script | Server Script
01 -----
02 -- The purpose of the Client Script is to
03 -- welcome the user to the update and then
04 -- download and run the Server Script.
05 -----
06
07 --Screen.Show("Client Screens");
08
```

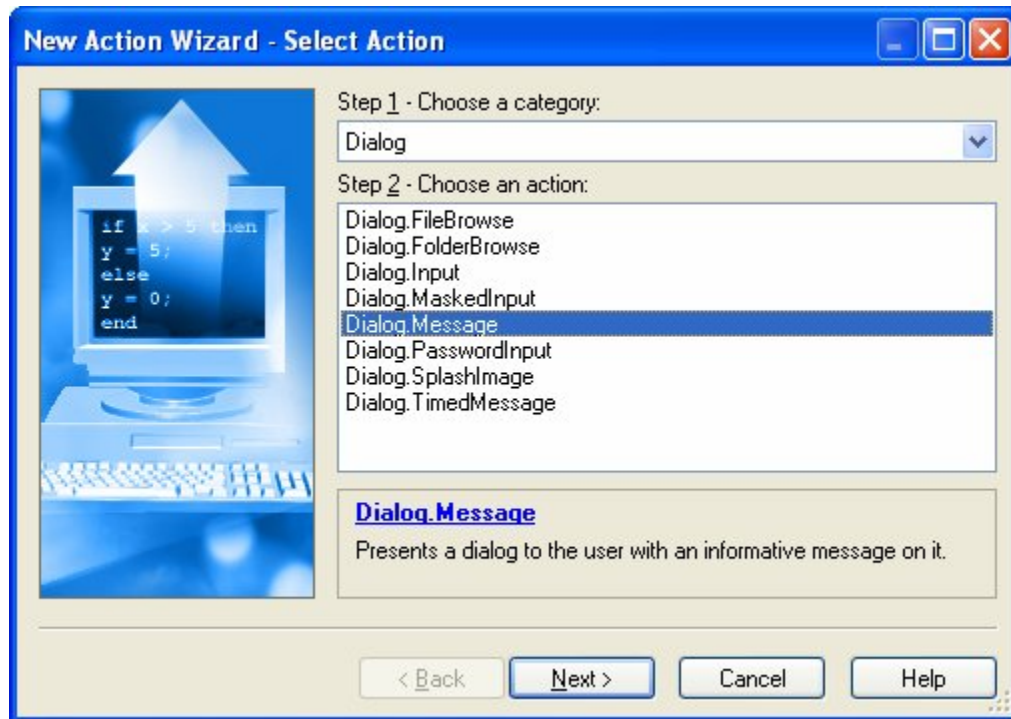
For the purposes of this example, we don't want the client script to display any screens; instead, we will be displaying a message in a popup dialog.

3) Place the cursor on the last line (line 8) and choose Script > Action Wizard from the program menu. When the action wizard appears, switch to the Dialog category and click on the action called Dialog.Message.

The action wizard will walk you through the process of adding an action to the client script. The first step is to choose a category using the drop-down list.



When you choose the Dialog category from the drop-down list, all of the actions in that category will appear in the list below.



To select an action from the list, just click on it. When you select an action in the list, a short description appears in the area below the list. In this description, the name of the action will appear in blue. You can click on this blue text to get more information about the action from the help file.

4) Click the Next button and configure the parameters for the Dialog.Message action.

Parameters are just values that get “passed” to an action. They tell the action what to do. For instance, in the case of our Dialog.Message action, the action needs to know what the dialog’s window title and message should be. You provide this information to the action in its first two parameters.

The first parameter lets you specify the title of the dialog. This is the text that will appear in the dialog window’s title bar.

The second parameter lets you specify the message that will be displayed on the dialog itself.

For now the other parameters are not important, but you should take some time to look at them and their options. (TrueUpdate will automatically fill the other parameters with appropriate default values.)

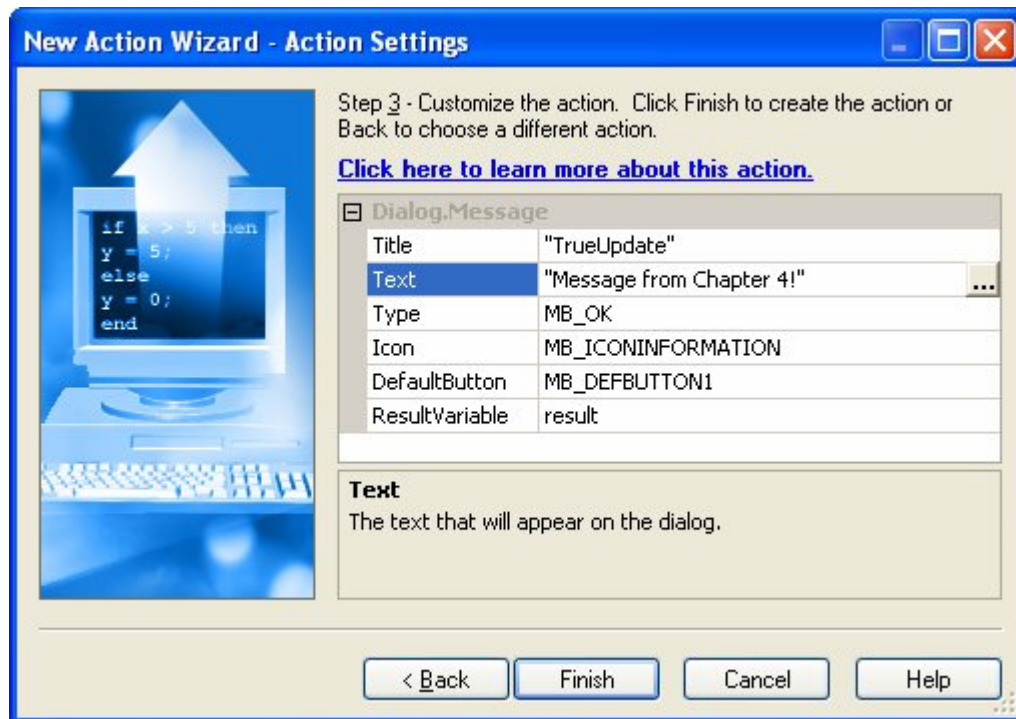
For now, change the title to:

"TrueUpdate"

and the text to:

"Message from Chapter 4!"

Note: Be sure to include the quotes on either side of both parameters. These are string parameters and the quotes are needed for TrueUpdate to properly interpret them.



Once you've set the action's parameters, click the Finish button to close the action wizard. The Dialog.Message action will appear on the Client Script tab.



```
01 -----
02 -- The purpose of the Client Script is to
03 -- welcome the user to the update and then
04 -- download and run the Server Script.
05 -----
06
07 --Screen.Show("Client Screens");
08 result = Dialog.Message("TrueUpdate", "Message from Chapter 4!", MB_OK,
09
```

Note that the parameters you provided are listed between parentheses after the action's name. The parameters are in the same order as they appeared in the action wizard, separated by commas.

5) Build the project and run the TrueUpdate Client. When the client starts, you should see the dialog created by the Dialog.Message action.

Start the publish wizard by choosing Publish > Build from the program menu. The default settings should suffice, so click the Next button to show the list of upload locations (which can be ignored for this example) and then click the Build button to start the build process.

Once the build has completed successfully, make sure the "Open output folder" checkbox is selected, and click the Finish button.

Once the output folder appears, double-click on the client executable to launch it.

You should see the following dialog message appear:



This popup "message box" is the result of the Dialog.Message action.

Click OK to close the message box and allow the TrueUpdate Client to exit.

Editing Actions

There are two ways that you can modify an existing action: you can either edit the action's text directly in the script editor, just like you would edit text in a word processor; or you can use the Action Properties dialog.

The Action Properties dialog is similar to the action wizard, but instead of walking you through choosing a category, it takes you straight to the part where you modify the action's parameters.

The easiest way to bring up the Action Properties dialog is by double-clicking on the action. Or, if you prefer, you can place the cursor within the action and either press Ctrl+E, or right-click and choose Edit Action from the context menu.

Tip: You can tell that the cursor is within an action when the action's function prototype appears in the quick help.

Here is a quick example illustrating how to edit the Dialog.Message action that we created in the previous section.

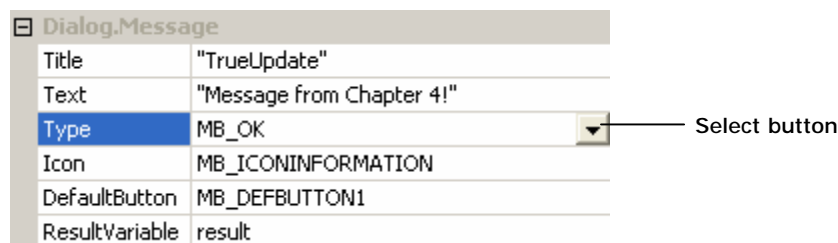
1) Double-click on the Dialog.Message action to bring up the Action Properties dialog.

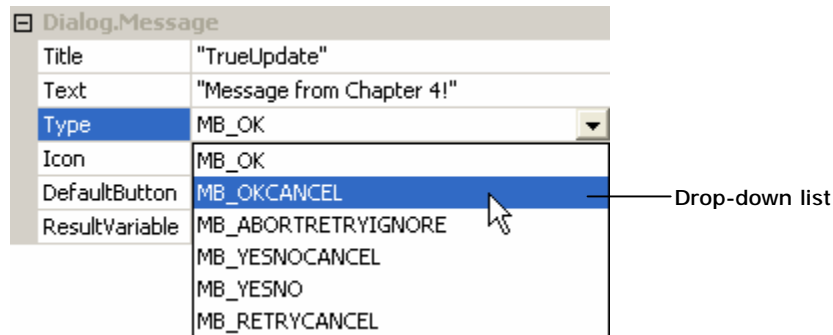
First, make sure the Client Script tab is selected. You should see the Dialog.Message action created in the previous topic.

To edit the action, just double-click it. Double-clicking on the action opens the Action Properties dialog, where you can modify the action's current parameters.

2) Change the Type parameter to MB_OKCANCEL and the Icon parameter to MB_ICONNONE.

To change the Type parameter, first click on the parameter field, then click the select button at the right edge of the parameter field and choose MB_OKCANCEL from the drop-down list.





For the Icon parameter, click the parameter field, click the select button, and choose MB_ICONNONE from the drop-down list.

These changes will add a cancel button to the dialog (MB_OKCANCEL) and get rid of the icon (MB_ICONNONE).

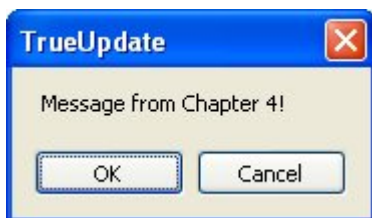
Finally, click OK to finish editing the action. Notice that the changes you made now appear in the script editor.

Constants

MB_OKCANCEL is a *constant*. A constant is a name that represents a value, essentially an “alias” for that value. Constants are often used to represent numeric values in parameters. It’s easier to remember what effect MB_OKCANCEL has than it is to remember what happens when you pass the number 1 to the action.

3) Build the project and run the client. When the client starts, you should see the dialog created by the Dialog.Message action.

When you run the TrueUpdate Client, a dialog will pop up. This dialog is created by the Dialog.Message action on the Client Script tab.



Notice that there is no longer an “information” icon on the dialog, and there is now a Cancel button next to the OK button. This is a direct result of the changes that you made to the action’s parameters.

Getting Help on Actions

You can get help on actions in a variety of different ways in TrueUpdate. For example, when you’re using the action wizard, text is displayed at the bottom of the dialog describing the current action or parameter that you have selected. Additionally, quick access to the help file is provided in the form of a blue text link that you can click on to receive context sensitive help for the action.

In the script editor, the current action’s function prototype is displayed in the quick help field—either on the Script Help toolbar, in the status bar, or directly on a screen’s event tab. In the case of the Script Help toolbar and the event tabs, there is even a help button next to the quick help field that will open the help file directly to that action’s help topic.

You can press the F1 key at any time while working in the script editor to open the TrueUpdate help file. The help file is especially useful when you’re working with actions. It contains a wealth of information on each action, including an overview topic with detailed information about each parameter, and a topic providing one or more working examples for that action.

The overview topic for an action will provide you with the function prototype, which serves as a definition of the action showing what (if anything) the action returns, along with the action’s parameters and their types.

A function prototype defines the types of all of the parameters, the type of the return value (if any), and whether or not any of the parameters have default values.

```
number File.Run ( string  Filename,
                  string  Args = "",
                  string  WorkingFolder = "",
                  number  WindowMode = SW_SHOWNORMAL,
                  boolean  WaitForReturn = false )
```

You can click on any part of the function prototype in the help file to learn more about that part and its particular purpose.

Includes

An *include* is any external script that can be “included” in your project. Also known as a *script file*, an include is simply an external text file that contains a valid TrueUpdate script. By convention, these text files usually have a .lua file extension.

You can add script files to your project on the Includes tab of the Resources dialog, which can be accessed by choosing Project > Includes.

All of the files listed on the Includes tab are stored in the client data file when you build your project. Note that this means adding or removing script files will result in a change to your TrueUpdate Client application.

Script files are very similar to script tabs in TrueUpdate except that instead of the script being kept in the project, it is stored in an external text file.

Note: All of the script editors in TrueUpdate have the ability to export their script to an external file. To save a script from the script editor, simply right-click and choose Advanced > Export from the context menu.

Script files are very useful if you need to share important and complex code between a variety of different projects. By keeping the common code in a single location, your projects will be easier to maintain.

This is more advantageous than simply copying and pasting scripts between projects. Having duplicated script in multiple projects can mean duplicated effort if you want to make changes to the script in all of the projects (e.g. if you discover a bug). For instance, when two projects contain copies of a script, you have to edit both projects in order to make the same changes to both scripts.

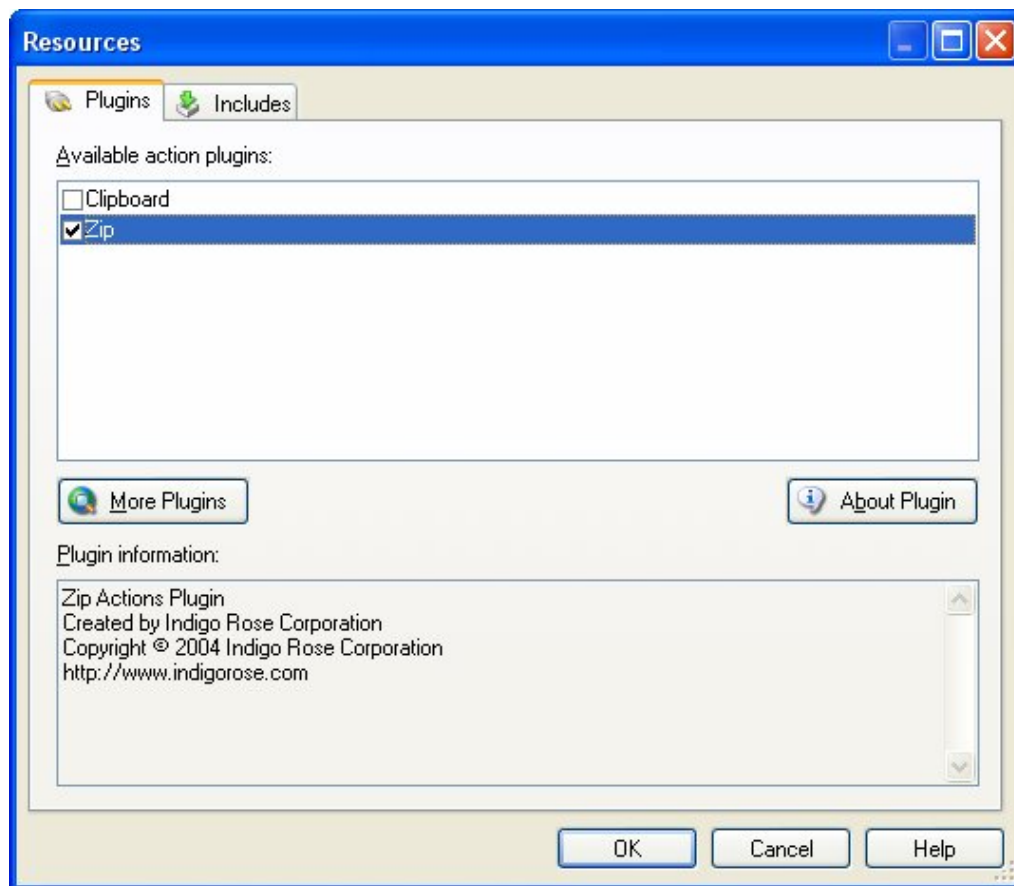
Using an external script file gets around this issue, by allowing multiple projects to share the same script. When using an external script file, each project does not contain a *copy* of the script; rather, each project *references* the *same* script. If you find an error or want to change any of the code, you do not have to edit the script in each project; you simply have to edit the script in the external file. Since each project references the same file, you know that the next time you build a project, it will be using the new script.

Using external script files allows you to maintain your script in a single location: the script file. For this reason, it’s a good idea to use includes whenever you want to share scripts between projects.

Plugins

Plugins are actions that are “added on” to the TrueUpdate program. They are independently developed and distributed and can be integrated into your projects to extend their functionality. You can obtain plugins developed by Indigo Rose as well as those developed by third parties.

Any available plugin can be enabled or disabled in your project on the Plugins tab, which you can access by choosing Project > Plugins.



Note: Only plugins that are installed in the Includes\Plugins folder within the TrueUpdate program directory will be available on the Plugins tab.

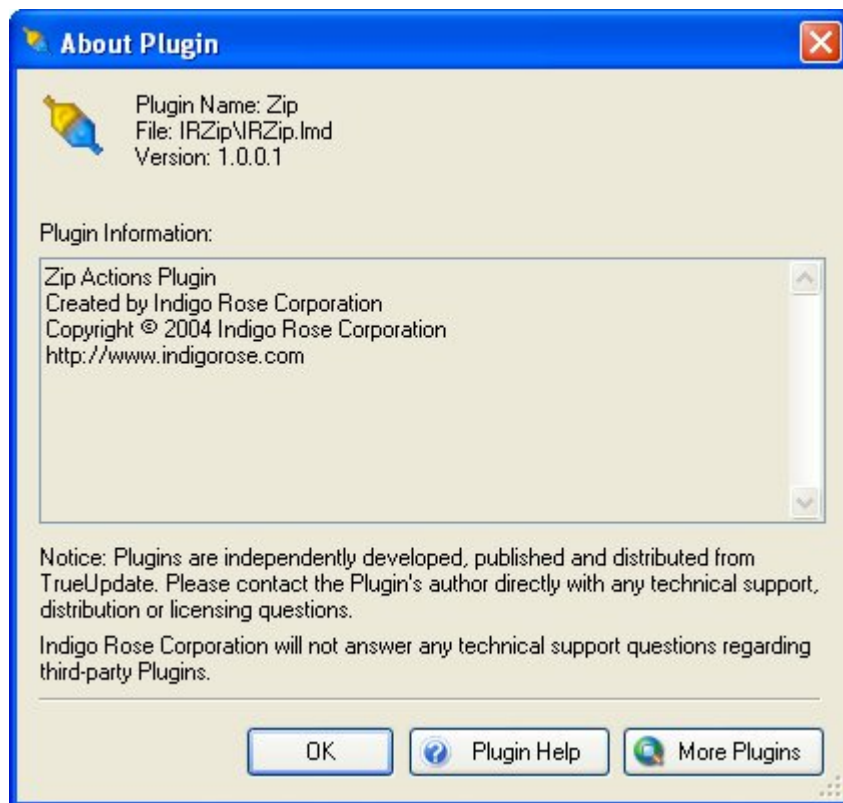
To enable a plugin in your project, simply place a check mark in the checkbox next to

its name on the Plugins tab. Only plugins that are enabled on the Plugins tab will be available in your project.

Once you have enabled a plugin in your project, all of its actions become available to you in the script editor and action wizard. You can even access the plugin's help file in the same way that you would access the help for built-in actions.

Note: Since there is some overhead in terms of file size, it is recommended that you only include plugins that are needed by your update. If you do not place a check mark beside a plugin, it will not be included in the TrueUpdate Client and will not take up any extra space.

For information about a particular plugin's features and how to use them, refer to the plugin's documentation. You can access a plugin's documentation from the Plugins dialog by selecting the plugin, clicking the About Plugin button, and then clicking the Plugin Help button on the About Plugin dialog.



Tip: The More Plugins button on the About Plugin dialog is an easy way to see what plugins are currently available on the Indigo Rose website. Clicking this button will take you directly to the TrueUpdate plugins area of the Indigo Rose website.

Where to Go from Here

Now that you've been introduced to the basics of scripting, you'll probably want to learn more about this important feature of TrueUpdate. Fortunately, there are several resources available to you.

The Scripting Guide

Chapter 11 of this user's guide is a crash course on the scripting language. It covers the basic syntax of the scripting language, and more advanced topics such as looping and functions.

The Project Wizard

The scripts that are generated by the project wizard are excellent real-world examples for you to follow. They were designed to serve as best-practice examples, and include full comments making them easy to follow despite their sophistication. Try choosing different settings in the project wizard, and see how it changes the script as a result.

Project Templates

When you reach the "Update Method" step of the project wizard, you are given the option to use a single installer/patch file, or a custom method. If you choose the custom option and click Next, you will be presented with a list of project templates to choose from. These project templates also contain excellent real-world examples of scripting. Like the other project wizard scripts, they are also fully commented.

Sample Projects

TrueUpdate comes with a number of sample projects that demonstrate very specific update situations. In addition to being good starting points for your own projects, these sample projects contain customized scripts and screens that you will not find in the generic scripts that the project wizard generates.

The sample projects are located in a "Samples" subfolder within the Program Files folder where TrueUpdate was installed.



Chapter 5:

Creating the User Interface

Creating the user interface is an integral part of every update. The user interface is the first thing your end users will see when they run your update. It also serves as a bridge between the information that the user has and that the update wants. Having an easy to use yet fully functional user interface is something that all users of TrueUpdate should be concerned about.

This chapter will introduce you to the user interface and get you well on your way to creating a sharp and consistent look and feel for all of your projects.

In This Chapter

In this chapter, you'll learn about:

- The user interface
- Client vs. Server screens
- Screens
- Themes
- Taskbar settings
- Dialog and Status Dialog actions
- Alternative update methods

The User Interface

You can think of the user interface as any part of the update that the end user will see. When a screen is displayed, the end user is seeing part of the user interface. When the end user clicks the Next button, they are interacting with the user interface.

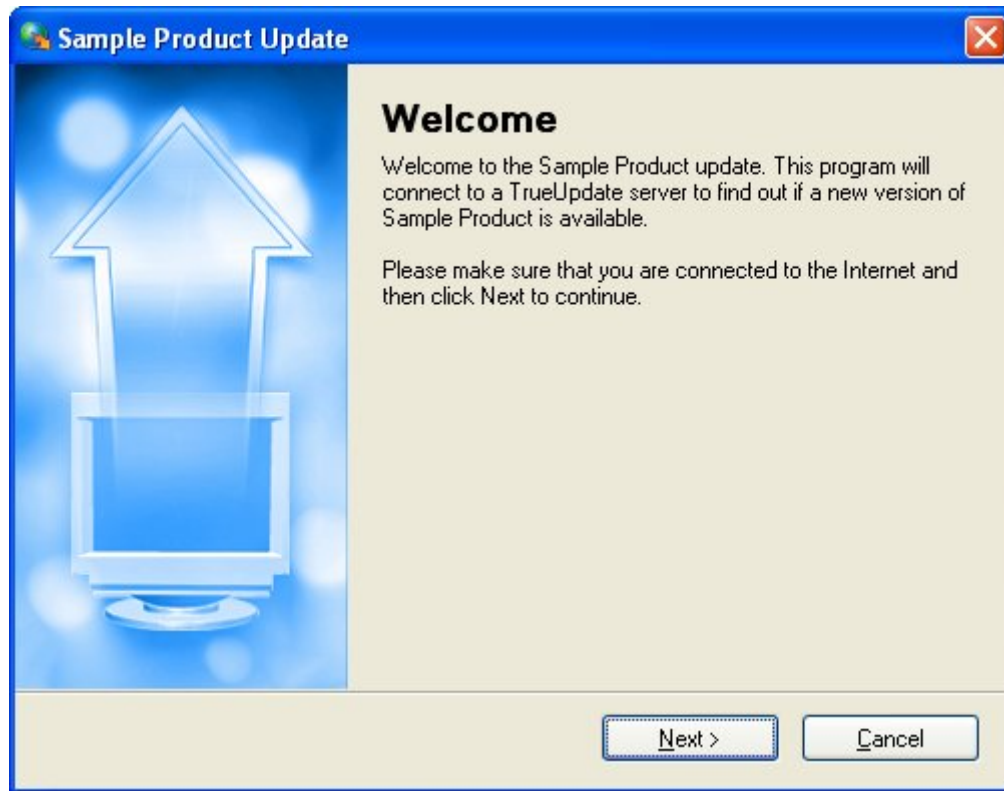
The basic elements of the user interface are:

- Screens
- Themes
- Taskbar visibility
- Any actions that generate a user-interface element (e.g. `Dialog.Message`, `StatusDlg.Show`)

Screens

The most important aspects of your user interface are the screens that you choose to display. Screens are where your end user will actually interact with the update. They allow you to provide important information (such as changes to system requirements) and allow your user to make decisions (such as locating the product to be updated).

Tip: It is possible to create an update project that doesn't use screens for its interface. For example, you can use individual popup "dialogs," or you can have your update remain "silent" and not show any interface at all. However, the most common and most attractive interfaces take advantage of TrueUpdate's screens.



A typical screen in TrueUpdate (the Welcome screen)

Screens are the individual windows that make up your update. When you navigate through an update by clicking the Next and Back buttons, you are navigating from screen to screen.

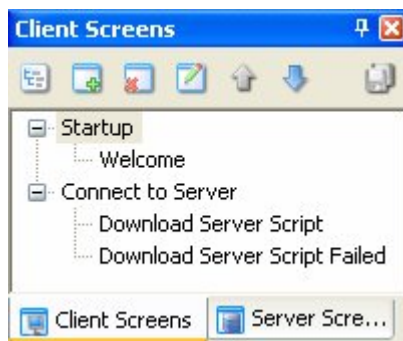
Tip: You can think of the screens in your project as steps in a wizard that walk your user through the process of updating their software.

In general, each screen performs a single task, such as showing a welcome message or letting the end user select which product to update. If you want to perform a major update task, chances are you will need a screen to do so.

The Screen Panes

The two screen panes are where you will configure all of the screens that are used in your project. By default, these two panes are tabbed together in the upper right hand corner of the program window.

There is one screen pane for client-side screens, and one for server-side screens. Since they are tabbed together by default, you can access either pane by clicking on the appropriate tab.



From the screen panes, you can add and remove screens and screen lists, as well as adjust the order of screens in your project. You can also import and export languages for individual screens.

Client Screens

When you build your project, the screens on the Client Screens pane are packaged with the TrueUpdate Client and are accessible to the end user without having to download anything from the Internet. These screens cannot be updated without updating the client itself. Whenever the TrueUpdate Client detects a newer version of itself on the server, it will update itself automatically; however, this self-update process requires the client to restart.

In order to make the update process appear as seamless to the user as possible, it is preferable to avoid unnecessary changes to the client screens after the client has been distributed to your customers. For this reason, you should only include screens on the Client Screens pane that will be used by the client script and that need to be shown before the server script has been downloaded.

Examples of typical client screens are:

- Welcome
- Download Server Script
- Download Server Script Failed
- Enter Password

Server Screens

The screens on the Server Screens pane are stored in a server configuration file. These screens are accessible to the end user only after the client has downloaded the configuration file—in other words, after the server script has been downloaded. This generally happens as the result of a `TrueUpdate.GetServerFile` action, which typically occurs every time the TrueUpdate Client is run.

Because the screens on the Server Screens pane are downloaded along with the server script, you are free to change them after your client has been distributed. Changes to the server script and server screens do not require the client to restart; any such changes will appear perfectly seamless to the user. In fact, you could completely replace all the server screens and the user would have no indication aside from the fact that this time, different screens are shown.

Examples of Server Screens are:

- Update Available
- Update Successful
- Download File (HTTP)
- Already At Target

Screen Lists

All screens in your project are organized into screen lists. The purpose of screen lists is twofold: they provide a way to group screens together, and they allow you to define separate *sequences* of screens.

A typical screen list will contain a sequence of screens that work together as a unit. For example, you might want to group all of the screens that deal with downloading

your patch files within one screen list. This allows you to treat those screens as a single unit. Using a single Screen.Show action, you can show the list, and the screens within that list will display sequentially.

Note: The Screen.Show action takes two parameters: the name of a screen list, and the name of a screen within that list. If you omit the second parameter, the action will begin showing the first screen in the list, and only “return” once it reaches the last screen in the list.

In general, the order in which screens appear within a list on the screen pane will be the order in which they appear during the update. The screen that is at the top of the list will appear first and the screen that is at the bottom of the list will appear last.

To change the order of your screens, simply select a screen in the list, then click the Up or Down button to place it in the desired location in the list.

Tip: You can use the Screen.Show action to show screens in the middle of a list by specifying which screen to start at. In this case, the screens would be displayed sequentially from that point forward. This can be useful if you want to “skip” some of the initial screens in a list in specific circumstances.

Adding Screens

Adding a screen to your project is easy. Simply select the screen list that you want to add it to, and click the Add button at the top of the screen pane. Note that screens can only be added to screen lists, and cannot exist on their own; a screen cannot be added until there is at least one screen list to add it to.

Note: To add a client-side screen, ensure that the Client Screens tab is selected. To add a server-side screen, ensure that the Server Screens tab is selected.

Clicking the Add button brings up the screen gallery where you can select from a variety of screen types. Once you’ve selected the type of screen you want, simply click OK to add it to the screen list.

Tip: Most of the screens in the screen gallery contain default scripts for their events. If the default behavior of a screen doesn’t suit your needs, remember that you can customize it by modifying the default scripts.

Removing Screens

To remove a screen from your project, simply select it on the screen pane and click the Remove button, or press the delete key.

Tip: If you remove a screen from your project by accident, you can undo the deletion by pressing Ctrl+Z.

Editing Screens

To edit a screen's properties, just select it in the list and click the Edit button.

Clicking the Edit button opens the Screen Properties dialog where you can edit and customize all of the settings for that screen.

Tip: You can also edit a screen by double-clicking on its name in the list.

Showing Screens

At run time, only screens that are explicitly shown by your script are presented to the user. In order to show a screen, you must use a Screen.Show action. This action takes two parameters: the name of a screen list, and (optionally) the name of a specific screen in that screen list.

For example, to show the first screen in a screen list, you would use the Screen.Show action and specify the name of the screen list you wish to show:

```
Screen.Show("List01");
```

The above action would show the first screen in the list named List01. Assuming each screen in that list had a Screen.Next() action in its On Next event script, each screen in that list would be shown in sequence as the user clicked the Next button until the end of the "List01" screen list was reached.

Alternatively, you can specify a starting point within a list by indicating which screen should be shown first:

```
Screen.Show("List01", "Screen04");
```

The above action would show Screen04, and (once again assuming the screen handles the On Next event normally) every screen succeeding it in List01.

Note: You can only show the screens on the Server Screens tab after the server configuration files have been downloaded using the TrueUpdate.GetServerFile action. Attempting to show a server screen without first getting the server configuration files will result in an error.

Screen Properties

The Screen Properties dialog is where you can edit the properties of a specific screen. All Screen Properties dialogs have the same four tabs (although the specific content on these tabs may differ depending on the screen type): Settings, Attributes, Style and Actions.

Settings

The Settings tab allows you to edit properties that are specific to the selected screen. Each screen type has different settings that are specific to that type of screen.

For example, a Check Boxes screen will have settings that apply to check boxes on its Settings tab.

For more information on the specific screen settings, please see the TrueUpdate help file.

Attributes

The Attributes tab contains settings that are common to all screens. The only difference that you will find between the Attributes tabs of different screen types is that Attributes tabs for progress screens lack options for the Next, Back, and Help buttons. This is because these buttons don't exist on progress screens.

In general, the Attributes tab is where you can configure which banner style to use, the name of the screen, and the navigation button settings.

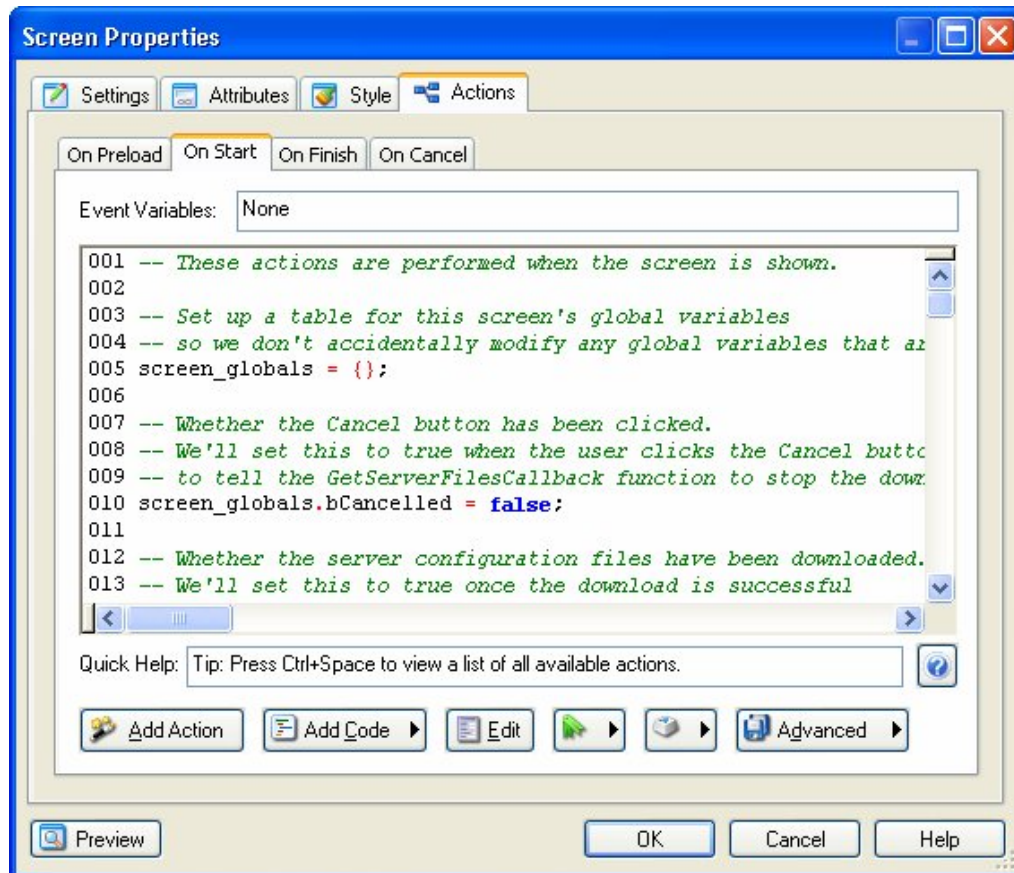
Style

The Style tab is where you can override the project theme on a per-screen basis. By default the project theme is applied to all screens throughout your project; however, you might feel that a particular screen needs something a bit different in order to stand out. You can use the Style tab to override any of the theme settings on a specific screen. Note that the changes will only be applied to that screen.

Actions

The Actions tab is where you can edit the actions associated with the screen's events.

For more information on actions and events, please see the help file and Chapter 4.



The Language Selector

The Settings and Attributes tabs both have a language selector in the bottom right corner. The language selector is a drop-down list containing all of the languages that are currently enabled in the project. It is used for creating multilingual updates.

Selecting a language in the list allows you to edit the text that will be used on the screen when that language is detected.

Session Variables

Session variables play a large part in the way that screens work and how they display their text. Anytime you see something in TrueUpdate that looks like %ProductName%, you are looking at a session variable.

Note: A session variable is essentially just a name (with no spaces) that begins and ends with %.

Session variables are very similar to normal variables in that they serve as “containers” for values that may change. We say that values are “assigned to” or “stored in” session variables. When you use a session variable, its name (e.g. %ProductName%) is replaced at run time by its value (e.g. “TrueUpdate”). Session variables are basically *placeholders* for text that gets inserted later.

Session variables are often used in the default text for screens. They are automatically expanded before the screen is displayed, so instead of seeing %ProductName% on the screen, the end user will actually see the name of your product that you entered in the project wizard, or on the Session Variables tab of the Project Settings dialog.

Session variables are also used to store return values when screens or controls need them.

Tip: Session variables can be created and changed at run time using actions like SessionVar.Expand, SessionVar.Get, SessionVar.Remove, and SessionVar.Set.

For more information please see Chapter 7, which discusses session variables in more detail.

Screen Navigation

Screen navigation can be thought of as the path that the user takes through the screens in your update. The user navigates forward through various screens by clicking the Next button, and backward through the screens by clicking the Back button.

Screen navigation is basically a linear path from the top screen in a screen list to the bottom screen. Generally, the order of your screens in a screen list is exactly the order in which the navigation will proceed. Although there are other ways to control the path through the screens (e.g. using actions to create a “branching” path), in most cases the default behavior is all that is needed.

Note: The user cannot normally move *between* screen lists using the navigation buttons. If you want them to be able to move back into a previous screen list, or forward into the next screen list, you will need to use additional Screen.Show actions in your script. For example, when Next is clicked on the last screen in a screen list, TrueUpdate does not display the next screen list, but instead returns back to the location where Screen.Show was called from. In order to show the next screen list, you would need to follow that first Screen.Show action with a second one.

How Screen Navigation Works

In its simplest form, screen navigation is when the user moves forward or backward through the update by clicking the Next and Back buttons. By default, this moves the end user down or up through the screens in a screen list.

This is actually accomplished using actions. Each screen has Screen.Next and Screen.Back actions on its On Next and On Back events which are performed when the Next and Back buttons are clicked. If necessary, you can modify or override the default behavior of any screen by editing or replacing the default actions with your own. Most of the time, however, you will not even need to know that the actions are there.

Navigation Buttons

Navigation buttons are the Back, Next, and Cancel buttons that are usually visible along the bottom (or “footer”) of each screen. The Next button moves the user down a screen list from the top to the bottom; the Back button moves up through the screen list; and the Cancel button stops the user’s navigation by canceling the entire update.

The settings for these buttons can be found on the Attributes tab of the screen properties dialog for each screen. There you can change the text, enabled state and visible state of these buttons.

The two options for the visibility state are self-explanatory; they make the button either visible or invisible. The two options for the enabled state make the button enabled or disabled. If a button is in the enabled state, it looks and functions like a normal button; it will depress when the user clicks on it, and the text is displayed in its normal color (usually black). When a button is in the disabled state, however, it will not respond to the user’s mouse, and is typically drawn in less prominent gray shades (also known as being “ghosted” or “grayed out”).

Each navigation button has an event that will be fired when the button is clicked. These events can be found on the Actions tab of the screen properties dialog.

Note: A Help button is also available on the footer of each screen but is generally not considered a navigation button.

Navigation Events

An event is something that can happen during the update. When an event is triggered (or “fired”), any actions that are associated with that event are performed. Note that an event must be triggered in order for its actions to be performed. If an event is not triggered, the actions associated with it will not be performed.

Each event represents something that can happen while your update is running. For example, all screens have an On Preload event, which is triggered just before the screen is displayed. To make something happen before a screen is displayed, you simply add an action to its On Preload event.

All of the three navigation buttons have an event that will be fired when they are clicked. The events are “On Back” for the Back button, “On Next” for the Next button and “On Cancel” for the Cancel button.

In the case of the three navigation buttons, navigation actions are executed when their respective events are fired. This allows the end user to navigate through the screens from the beginning to the end.

There are other events that are associated with screens but aren’t necessarily related to screen navigation:

- On Preload – just before the screen is displayed
- On Help – when the help button is selected
- On Ctrl Message – triggered when a control on the screen fires a control message (for example, when the user clicks on a button, or when an item is selected in a list box)

Navigation Actions

There are seven navigation actions available to you in TrueUpdate: Screen.Back, Screen.End, Screen.Jump, Screen.Next, Screen.Previous, Application.ExitScript, and Application.Exit. The most common of the seven actions are Screen.Next and Screen.Back.

When the Next button is clicked, the user is attempting to navigate from the current screen to the next screen in the update. The easiest way to implement this behavior is to insert the Screen.Next action on the On Next event. This is done by default for all screens.

The same holds true for the Back button; when the Back button is clicked, the user is attempting to move backwards in the update to the previous screen. To implement this behavior, a Screen.Back action needs to be executed when the On Back event is fired.

Note: The Screen.Back action moves backward through the screen history in the same way that a Back button does in a web browser: it sends you “back” to the previously viewed screen. To move up one screen in the screen list, use the Screen.Previous action, which sends you to the previous screen *in the list*.

In certain situations, simply moving down a screen list is not the appropriate behavior; instead, jumping to a specific screen in the list is necessary. You can accomplish this by using a Screen.Jump action. If the goal is to jump to the next list in the update, first a Screen.End action can be used to jump past all of the screens in the current screen list, and then a Screen.Show action can be used to show screens from the next list.

To interrupt screen navigation—which usually occurs when the Cancel button is clicked—you can use an Application.ExitScript action. The Application.ExitScript action immediately exits from the current script—essentially skipping any actions that follow and going straight to the end of the script. In other words, it forces the immediate interruption of the current screen event.

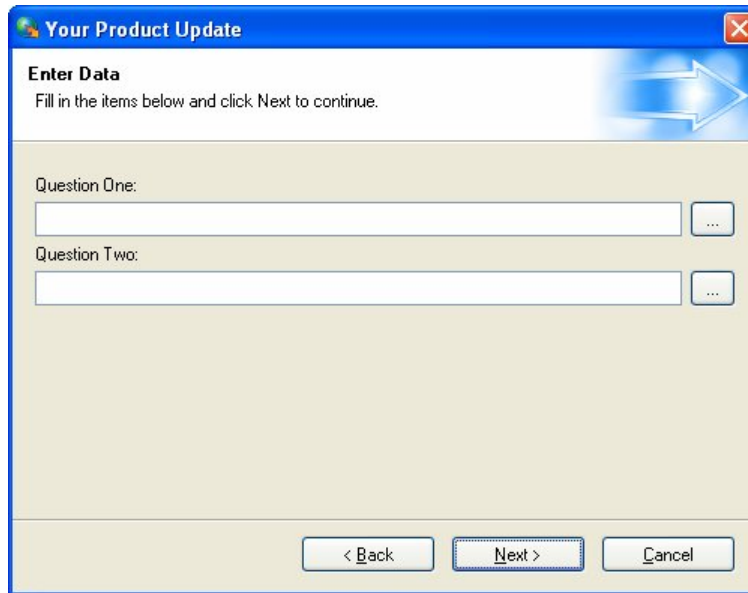
The Application.Exit action goes one step further: it causes your entire update to exit as soon as the action is performed. In other words, it not only stops the current script, it stops the entire TrueUpdate Client application.

Tip: You can find detailed information on these actions in the help file.

Screen Controls

A control is an “object” on a screen that serves a specific purpose, such as receiving mouse clicks, displaying text, or providing an option in the form of a checkbox, radio button, etc. Most screens contain several controls. You can think of the controls as the different screen “parts” that allow the screen to do what it does. In other words, controls are the parts of a screen that your users directly interact with.

Note: Most controls are designed to display information to the user or receive the user's input.



An Edit Fields screen showing three types of controls: labels, edit fields, and buttons

TrueUpdate provides a number of actions that allow you to get and set the properties of the controls on a screen. You can use these actions to change the information that is displayed on a control, or to retrieve the information that the user provided.

For example, if you need to determine whether a checkbox is selected, you can use a `DlgCheckBox.GetProperties` action to investigate the control's "checked" property.

Similarly, if you wanted to change the contents of a list box according to the user's input on a previous screen, you could modify the list box's items using actions like `DlgListBox.AddItem` and `DlgListBox.DeleteItem`.

Tip: The control-related actions in TrueUpdate begin with "Dlg," which is short for "Dialog." In programming terms, TrueUpdate's screens are implemented as *dialog windows*. The "Dlg" prefix was chosen for its familiarity to Windows programmers.

You can also use actions to programmatically show and hide controls, or enable and disable them. For example, you could hide a button on a screen if the user's input on the previous screen made that button unnecessary. Or you could make one checkbox

only become enabled after the user selects another checkbox, essentially making the second checkbox a “sub-option” of the first.

Note: For more information on screen controls and the related actions, please consult the help file.

Screen Layout

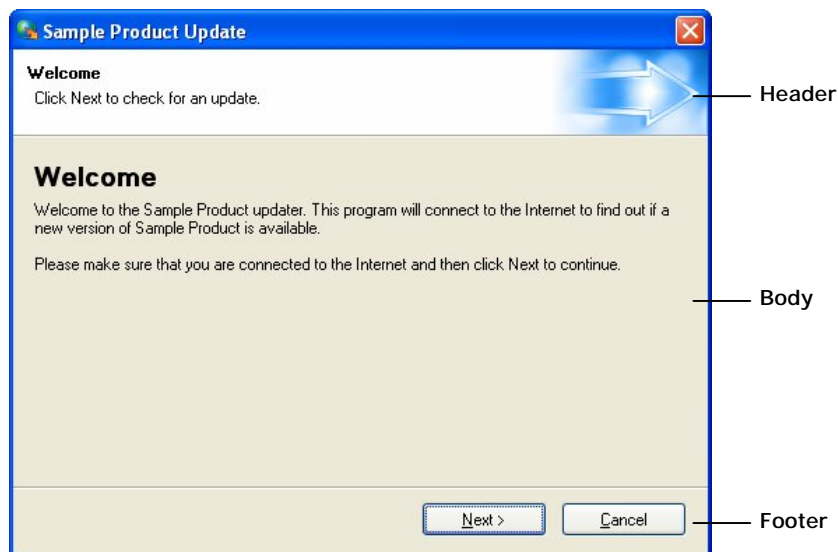
In TrueUpdate choosing a layout for your screens and their controls is incredibly easy. You can switch between all three banner styles (top, side, and none) on any screen that you like, and the controls on your screens will dynamically position themselves ensuring that all of your information is visible.

Note: A control can be thought of as any visible element on a screen, from edit fields, to radio buttons, to static text controls. However, when the term “control” is used, it does not generally refer to the navigation buttons or banner text.

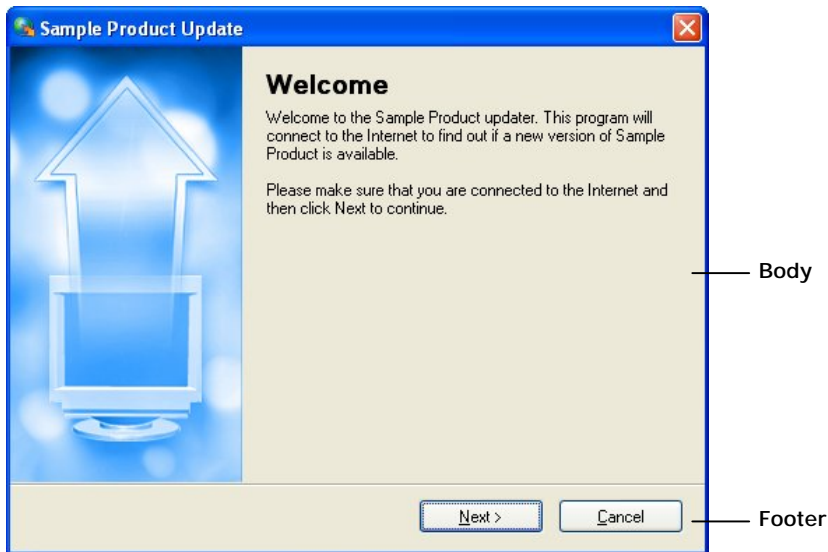
Header, Body, Footer

Screens in TrueUpdate are divided into three basic parts: the header, the body and the footer.

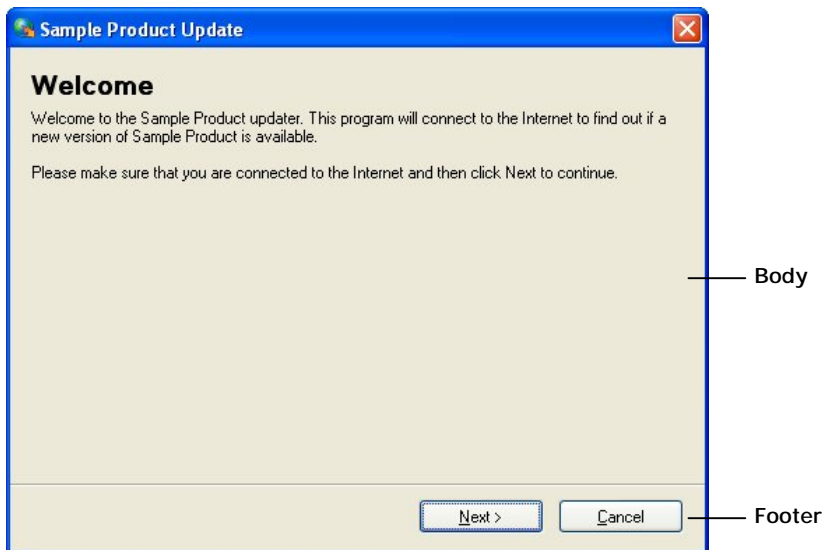
The header runs across the top of each screen and can be thought of as the area that the top banner fills.



The footer is similar to the header area except that it runs along the bottom of each screen. This is the area of the screen where the navigation buttons are placed.



Now that you know what the header and the footer are, you can think of the body as the rest of the screen. The body of a screen takes up the majority of each screen and will contain most of the screen's information.



Banner Style

In TrueUpdate the term banner refers to an area of the screen that is special and somewhat separate from the rest of the screen. You can use the banner area to display some descriptive text, an image, or both.

There are three different types of banner styles available in TrueUpdate: none, top, and side.

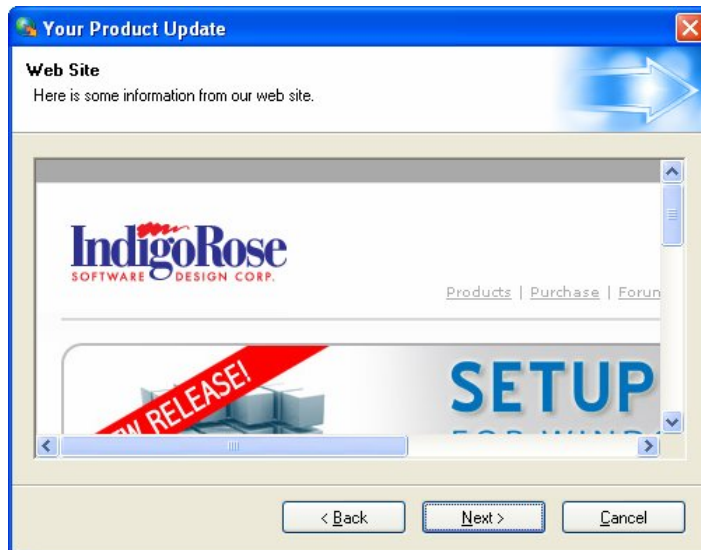
The none banner style is the easiest of all three styles to understand since it means that there will be no banner displayed on the screen.



The none banner style

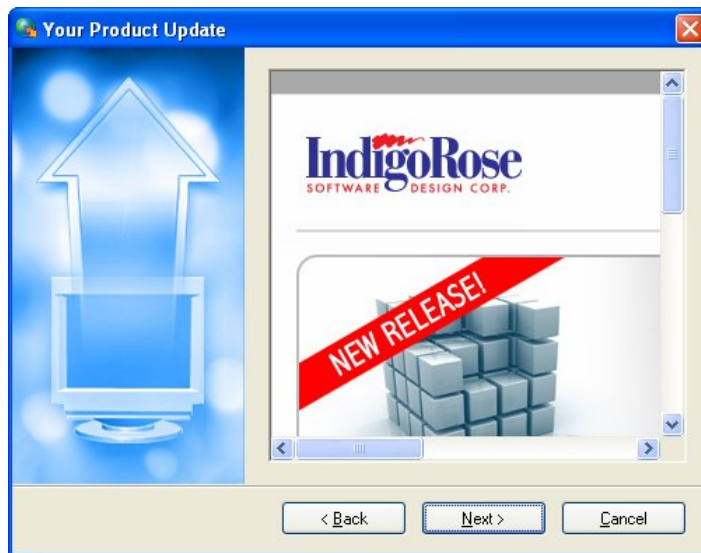
The top banner style has a long thin banner (or “header”) across the top of your screen. This is the style that you will probably apply to the majority of the screens in your project. The top banner style supports two lines of text referred to as the heading text and the subheading text. This text is usually used to describe the current screen and/or provide some information about what is required of the user.

The top banner style also supports an image that will be placed on the right hand side of the banner. The image is drawn starting from the upper right corner and extending towards the lower left. If the top banner image is taller than the banner area, it will extend down into the body of the screen. In fact, the top banner image can be as large as the body and cover the whole screen. Any area of the top banner that is not covered by the image will be painted with a color according to the project theme.



The top banner style

The side banner style has an image that runs down the left side of the screen, forming a vertical banner alongside the body.



The side banner style

The side banner image is drawn in the upper left corner of the screen, starting from the upper left and extending towards the lower right. Like the top banner image, it can be larger than the banner area and extend into the body.

Tip: For more detailed information on how the screens in TrueUpdate are drawn, and for an example of how to use the side and top banner images to create interesting backdrops for your screens, see *How Screens Are Drawn* in the help file.

Dynamic Control Layout

One of the best features in TrueUpdate is the dynamic control layout ability of screens. TrueUpdate will dynamically reposition the controls on your screen so that the maximum amount of information stays visible.

Dynamic control layout means that controls will resize and layouts will adjust automatically as you type. You no longer have to manually place your controls and you won't find yourself locked into a pre-determined amount of space.

The dynamic repositioning of controls takes place within an area called the control area. The control area of a screen occupies a sub-section of the body of a screen; its size is controlled by the global theme.

Offsets	
Top Banner	15, 15, 15, 15
Side Banner	15, 15, 15, 15
No Banner	15, 15, 15, 15
Banner Text X	10
Banner Text Y	10

The control area for each banner style is defined in the theme settings using offsets from the top, bottom, left and right sides of the body.

The best part of the dynamic control layout feature is that it works without any effort on your part. Simply fill your screens with all of the information and controls that you want, and TrueUpdate will re-position all the controls so that everything is visible and a visually appealing look is achieved.

This is not to say that you do not have any control over how controls will be displayed on your screen; in fact, it's just the opposite. Many screens (Edit Fields, Checkboxes, etc.) allow you to add as many as 32 controls to your screen, which TrueUpdate will dynamically position. You have the ability to choose how many columns you want the controls displayed in, whether they are distributed horizontally or vertically and, in the case of the Edit Fields screen, how many columns each control spans!

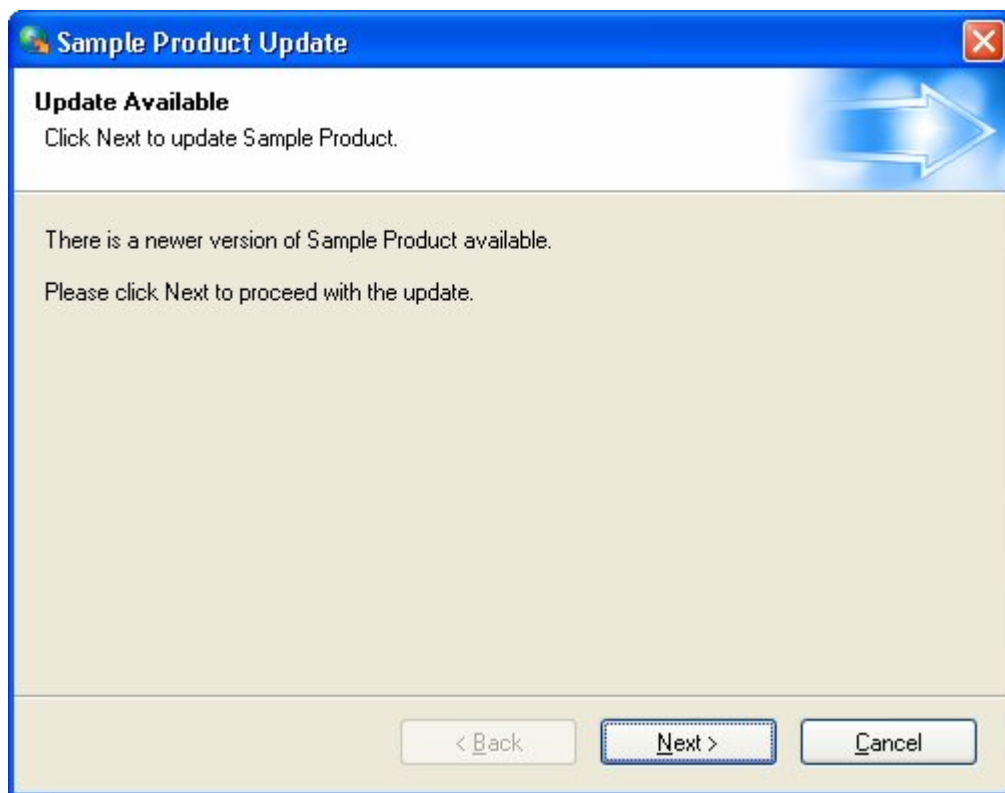
The best way to understand the dynamic control layout feature is to actually use it yourself. Try playing around with the settings of a Check Boxes or Select Folder screen and observe how TrueUpdate positions your controls to achieve the best look possible.

Themes

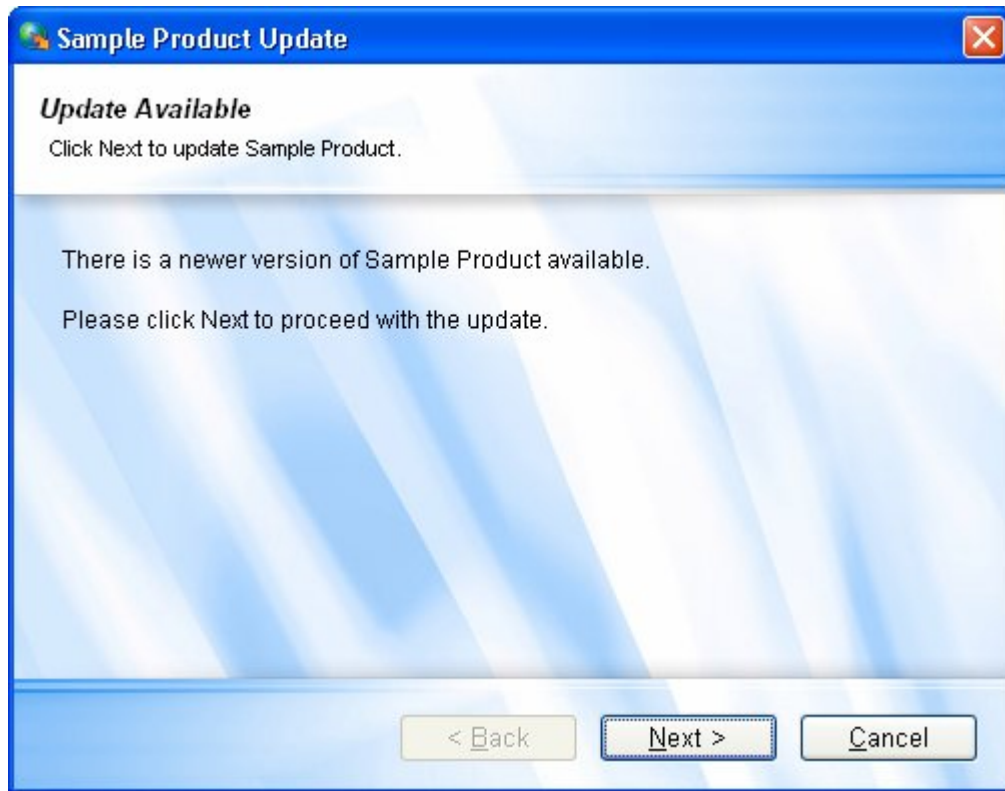
A theme is a group of settings and images that control the way your update looks. You've probably encountered themes before when using other applications or even Windows XP. Themes do not change *what* is displayed; instead, they change *how* it is displayed.

Themes in TrueUpdate determine the general appearance of your screens and the controls they contain. Rather than managing the position of screen controls or the banner style used, themes determine the color and font of screens and controls. Themes are project-wide and affect all screens in the project unless intentionally overridden on the style tab of the screen's properties.

Themes provide an easy way to change the look and feel of your screens and controls across your entire project.



An Update Available screen with the "Default" theme applied



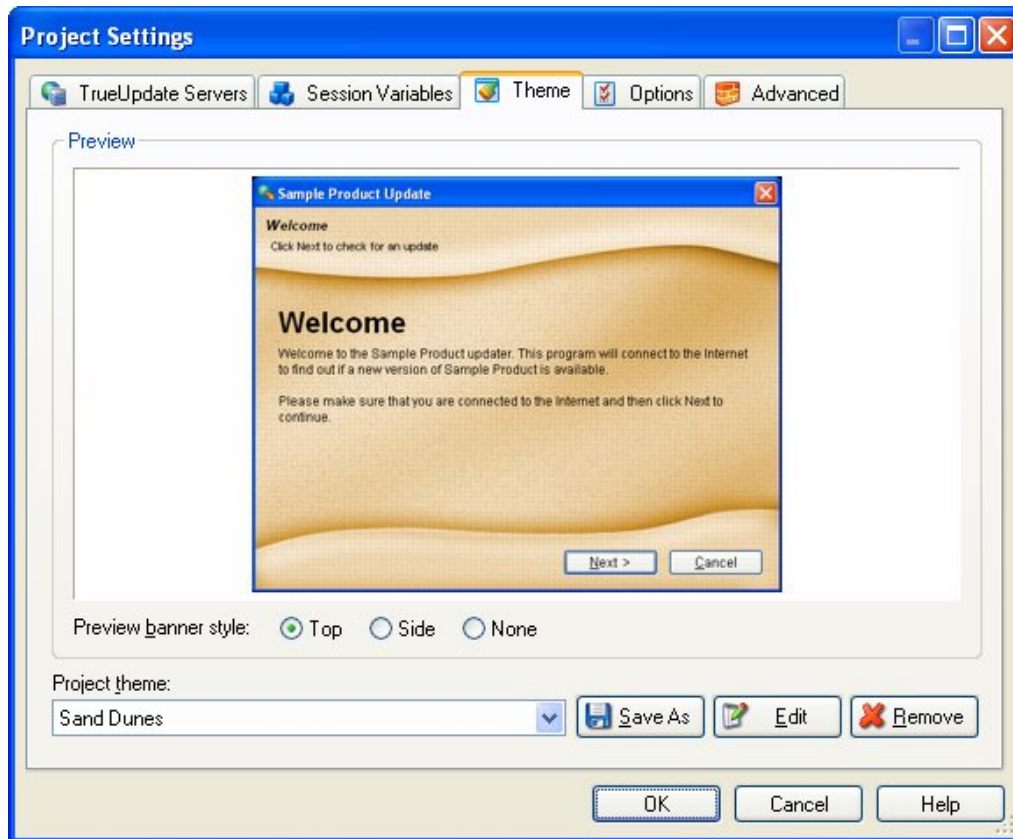
The same screen with a custom theme applied

Choosing a Theme

You can choose a theme for your project on the Theme tab of the Project Settings dialog, which can be accessed by choosing Project > Theme.

The drop-down list on the Theme tab contains a list of all the themes that are available in the project. Selecting a theme in this drop-down list will apply the theme to all of the screens in your project. For your convenience, a preview of the currently selected theme is displayed on the Theme tab as soon as you make a selection.

As shown above, themes affect the appearance of screens and their controls. For example, choosing a theme that colors static text controls purple will result in *all* static text controls being purple, and choosing a theme that colors static text controls black will result in *all* static text controls being colored black.



Creating a Custom Theme

TrueUpdate allows you to create your own custom themes. This provides you with an easy way to share the same custom look and feel between multiple projects.

Here is a brief step-by-step guide to help you in the creation of a custom theme.

1) Start a new project and save a copy of a pre-existing theme.

Start a new project by choosing File > New Project from the menu, then open up the theme settings by choosing Project > Theme.

The first step in creating a custom theme is to select an existing theme to base your new theme upon. If you cannot find a suitable theme, simply choose the default theme.

Once you have selected a theme to start from, use the Save As button to save a copy of it under a new name. Choose a name that describes the theme you plan to make; this theme is what you will be modifying in order to create your new theme.

2) Edit your new theme in the Theme Properties dialog and click OK to save your changes.

Make sure that your new theme is selected and click the Edit button to bring up the Theme Properties dialog. Here you will be able to edit all of the properties of your theme. Once you have made all of your changes, simply click the OK button and the changes to your theme will automatically be saved.

Now you have a working theme that will be available to you in all your TrueUpdate projects.

Note: If you are not happy with the changes made while editing your theme, simply click the Cancel button and your changes will not be saved.

Overriding Themes

As stated earlier, project themes affect every screen in your project. While in the vast majority of situations this is the desired effect, there may be a few instances where this is not exactly what you want. Fortunately, TrueUpdate allows you to override any or all of the theme settings on any of your screens.

As mentioned in the Screen Properties section, each screen has a Style tab associated with it. If you look at the Style tab you will notice that it looks identical to the Theme Properties dialog except that it has a checkbox in the top left corner labeled “Override project theme.”

Choosing the override project theme option will enable the theme settings and allow you to make changes to the theme settings strictly for the current screen. The changes you make on the Style tab will not affect any of the other screens in your project.

Note: If you decide that you want to go back to the project theme on a screen where you have overridden it, simply go to the Style tab and uncheck the Override project theme checkbox. There is no need to re-create the screen.

Other Options

There are a few other options in TrueUpdate that relate to the user interface. These options may not be as important as screens or themes, but they just might provide the elements necessary to perfect your project's look and feel.

Taskbar Settings

The taskbar is the bar that runs across the bottom of all modern Windows operating systems beginning with the START button on the left. When a program is running, its icon and name will generally appear in a button in the taskbar.



TrueUpdate allows you to choose whether or not to show an icon in the task bar and to choose what that icon will be. Both of these settings can be found on the Options tab of the Project Settings dialog: Project > Options.

To choose a custom icon, enable the “Use custom icon” option, click the browse button, and locate the icon of your choice.

To hide the taskbar icon, simply select the appropriate option (i.e. “System tray” or “Hidden”) in the Taskbar Visibility section of the Options tab.

Actions

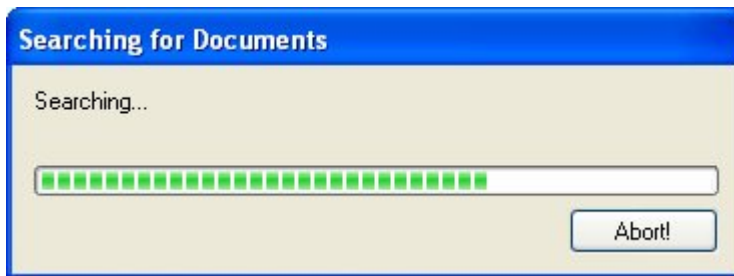
Some of the actions available to you in TrueUpdate are capable of showing user interface elements. These actions can be divided into two main categories: Dialog actions and Status Dialog actions.

Dialog actions are used to show pop up dialogs to the user. Examples include the Dialog.Message action that lets you display a message in a dialog, and the Dialog.TimedMessage action that lets you show a dialog with a message for a specific amount of time.



A typical message dialog

Status dialogs are the other main user interface elements that are available to you through scripting. Status dialogs are mainly used to show progress during a lengthy event like an HTTP.Download action or a File.Find action.



A status dialog

Status dialogs are shown and configured using actions like `StatusDlg.SetMessage`, `StatusDlg.ShowProgressMeter`, and `StatusDlg.Show`.

Note: It is generally recommended that progress be shown in a more integrated manner by using a progress screen; however, there are situations where a status dialog may be more appropriate.

Tip: For more information on the Dialog and StatusDlg actions, please consult the help file.

Alternative Interfaces

Silent Updates

Although the user interface is the most important aspect of most updates, there may be occasions when a “silent” update is preferred. A silent update requests no input from the user, and performs its work invisibly.

For example, if your update will be run regularly on system startup and can function completely using hard-coded values, you may prefer your update to act entirely in the background, unseen by the user—whether to avoid annoying the user, or to prevent the user from interrupting the update process.

There are two main steps in creating a silent update. The first, and least complicated, is ensuring that the update runs in a ‘hidden’ mode. This is accomplished by selecting the Hidden option in the Taskbar Visibility section of the Options dialog (which can be accessed by choosing Project > Options). This will prevent the TrueUpdate Client from displaying an icon on the Windows taskbar while it is running.

Next, your update must be configured in such a way that it requires no input from the user. This means that everything must be anticipated and handled in your scripts. Any paths must be hard-coded in or determined automatically (e.g. registry keys, INI files, etc.) without any input from the user.

As a backup plan, you could configure your update to ask the user for any information that it cannot determine automatically. This negates the purpose of a silent update, but may be preferable to the update failing all together.

Dialog-based Updates

A dialog-based update uses popup dialogs for its user interface, essentially replacing the full screens of a wizard style interface with an interface made up of small, simple dialogs or “message boxes.” This approach aims to offer a streamlined interface to the user, presenting a minimum amount of information without any of the graphical panache of TrueUpdate’s fully themed screens.

The dialog interface style is most often used by developers who wish to incorporate TrueUpdate directly into another application, and who want the update process to appear tightly integrated into their product. The dialog-style updates you can create

with TrueUpdate use standard Windows message boxes that will appear just like any standard messages and dialogs that are displayed by your own application.

Tip: Another way to make the update process appear integrated with an application is to create a custom theme. Using custom themes, it is possible to make an update with a wizard-style interface that closely matches the look of a particular application.

In a dialog-based update, you do not use screens or any of the screen actions. Instead, you perform all required actions in the client script and server scripts. The client script would check for a connection to the Internet and download the server script. Then, your server script would perform all required update functions, without calling upon any screens.

Any information that needs to be presented to the user or received from the user is done using actions like `Dialog.Message` and `Dialog.Input`. If you want to display the progress of a lengthy action (like `File.Copy` or `HTTP.Download`), you can use actions such as `StatusDlg.Show`, `StatusDlg.SetMeterPos` and `StatusDlg.Hide`.

Chapter 6:

TrueUpdate Servers

One of the best features of TrueUpdate is that it uses standard server technology. It doesn't require any proprietary hardware or software—all you need is a location where you can host the three server configuration files. These locations are called TrueUpdate Servers.

In This Chapter

In this chapter, you'll learn about:

- What are TrueUpdate Servers?
- Types of TrueUpdate Servers
- Adding, removing and editing servers
- Using multiple servers for redundancy
- Why TrueUpdate provides scalability

What Are TrueUpdate Servers?

In TrueUpdate, the term *TrueUpdate Server* refers to the location of your server configuration files. In other words, a TrueUpdate Server is any location where the TrueUpdate Client can download your server configuration files. This can be a standard HTTP server, a standard FTP server, or even a folder on your local area network.

The server configuration files allow the TrueUpdate Client to update its client-side data and executable files if newer versions are available, allow it to determine whether an update is required, and tell it the exact steps to take in order to perform the update. They are generated whenever you build a project. Once you upload the server configuration files to a location where the TrueUpdate Client can find them, that location becomes a TrueUpdate Server.

Tip: You can set up multiple, redundant TrueUpdate Servers by hosting the configuration files at more than one location.

Each TrueUpdate Client contains a list of all the locations where it can download the configuration files. The items in this list are also called TrueUpdate Servers. In this context, a TrueUpdate Server is the collection of settings that describe a location where the server configuration files can be found and the method that should be used to download them. These settings include a name that uniquely identifies the TrueUpdate Server along with the location (e.g. a URL) and any other connection details that are required to access the files, such as a username and password.

Tip: TrueUpdate can automatically upload your server configuration files to your TrueUpdate Servers via FTP, SFTP, or direct file copy. You can configure these automatic upload locations on the Upload tab of the Build Settings dialog, which you can access by choosing Publish > Settings from the program menu.

Types of TrueUpdate Servers

TrueUpdate provides a variety of methods for downloading server configuration files. All file access is handled using standard protocols—TrueUpdate doesn't require any special hardware or specific types of servers.

Each download method is represented as a different “type” of TrueUpdate Server.

The four types of TrueUpdate Servers are listed below.

HTTP Server

An HTTP TrueUpdate Server is a location (folder) on any Web server. For this type of server, the TrueUpdate Client will download the server configuration files from the Web site using the HTTP protocol. TrueUpdate uses Internet Explorer's connection settings, so even if a client is behind a proxy server, they should have no problems provided they are able to use Internet Explorer successfully.

HTTPS Server

An HTTPS TrueUpdate Server is a location (folder) on a secure Web server. For this type of server, the TrueUpdate Client will download the server configuration files from the server using the HTTP protocol exchanged over an SSL (Secure Socket Layer)-encrypted session. This is known as the "secure hypertext transfer protocol." As with HTTP, TrueUpdate also uses Internet Explorer's connection settings for HTTPS connections, so the same likelihood of achieving a successful connection across a proxy server exists for this method as well.

Note: This type of location requires that you have access to a secure Web server.

FTP Server

An FTP TrueUpdate Server is a location (folder) on an FTP server from which the TrueUpdate Client will download the server configuration files using FTP (file transfer protocol).

LAN/Local Server

As its name suggests, a LAN/Local server is a location (folder) either on a local drive, a mapped network drive, or a network path. In this case, the TrueUpdate Client copies the server configuration files from the TrueUpdate Server location on the network.

Note: This type of server is most useful when you are using TrueUpdate to update applications within your organization. Keep in mind that those without access to your local area network will be unable to download files from this type of TrueUpdate Server.

Adding, Removing and Editing Servers

All TrueUpdate Servers can be added and configured through the TrueUpdate Servers tab of the Project Settings dialog. You can access this dialog by choosing Project > TrueUpdate Servers from the program menu.

You can add a TrueUpdate Server by clicking the Add button and selecting the appropriate server type from the menu. Once you have selected a server type, a properties dialog will appear allowing you to specify how the server will be accessed. Each server's properties depend upon what type of server was selected. With that in mind, properties generally include a unique name or identifier that is used to refer to the TrueUpdate Server, the folder path or address, and any other connection settings that are necessary to access it.

Once a TrueUpdate Server's settings have been configured through its properties dialog, clicking OK adds it to the list on the TrueUpdate Servers tab. If the server's settings need to be adjusted later, you can highlight the desired TrueUpdate Server and click the Edit button to open its properties dialog. Also, note that any TrueUpdate Server can be easily removed. Simply highlight the desired TrueUpdate Server and click the Remove button.

Note: TrueUpdate supports as many TrueUpdate Servers as you require.

TrueUpdate Server Redundancy

Although most updates can be served from a single location, many developers find it useful to have one or more backup servers in case of problems with the primary server. TrueUpdate makes this easy to accomplish by allowing you to set up multiple TrueUpdate Server locations.

TrueUpdate's approach to handling redundancy is straightforward: simply add more TrueUpdate Server locations to the list, and see that the server configuration files are uploaded to those locations. If a server is unavailable for any reason, the TrueUpdate Client will move on to the next one until it can establish a connection and download the configuration files.

In fact, TrueUpdate is designed to support redundancy to any level. You can add as many TrueUpdate Server locations as you need, and you can mix the server types as well—for example, you could have a primary HTTPS server, followed by a pair of HTTP servers and an FTP server as backups.

Tip: If you're designing an update for a mission critical application, use TrueUpdate's support for redundancy to ensure that, given appropriate access to the Internet or your internal network, your TrueUpdate Client can always access a TrueUpdate Server.

By default, any project generated through the project wizard is designed to run through the list of TrueUpdate Servers defined in the project until the server configuration files have been downloaded successfully. In other words, if the first server fails, it will automatically try the next location. The default project achieves this functionality by using a `TrueUpdate.GetUpdateServerList` action to get a list of all of the servers, and then using a `TrueUpdate.GetServerFile` action to download the server configuration files from each TrueUpdate Server in turn.

While simply going through the list of TrueUpdate Servers in the order they are defined on the Project Settings dialog is the default behavior, this is by no means the only option available to you. Since the default server interaction is scripted, it is possible for you to modify it even further. For example, you may want to use different TrueUpdate Servers in specific cases. You could change the primary server for clients distributed in different geographic locations, for instance—letting the client automatically choose the appropriate TrueUpdate Server location whether its run from North America, Europe, or Asia. By customizing the script, you can have full control over the entire process. This allows you to choose the best server for each particular client.

TrueUpdate Server Scalability

From the ground up, TrueUpdate was created to be fully scalable and fault-tolerant. In addition to being able to easily configure the client application to access redundant servers, *you* control the underlying server technology. Because TrueUpdate uses standard Internet protocols such as HTTP, HTTPS, FTP and LAN, you are free to take advantage of the full range of server technologies available to you. Unlike other update services that use proprietary server hardware and software, TrueUpdate allows you ultimate control over load-balancing and distributed processing of client/server requests.

Another exciting feature of TrueUpdate is the client's ability to update itself with the newest client executable or data file. That way, if any changes are made to the client—such as an updated list of TrueUpdate Servers—these changes can be propagated through to the existing clients that have already been distributed. While we generally recommend not modifying the client after it has been distributed, this feature

can be a lifesaver if you run into a situation where you require additional servers or need to change existing ones.

Note: In order for the client to update itself, it must be able to access at least one of its previously defined TrueUpdate Servers. In other words, you cannot simply update the TrueUpdate Server list on your end, switch servers entirely, and expect the client to update itself. Before the client makes use of any new servers, it must know what those new servers are.

Chapter 7:

Session Variables

When designing an update, it is often desirable to make parts of it dynamic. For example, the user might input a value on one screen that you'd like to display on the next. Or you might want to display a path on a screen (as the default value in an edit field, perhaps), but the path includes a folder like "My Documents" that is likely to have a different location on each user's system.

Although you could use regular script variables along with actions to manipulate the screen text at run time, session variables allow you to accomplish the same result in a more direct way: by simply including "placeholders" in your screen text that will automatically be replaced by specific values before the screen is shown.

In this chapter you will learn everything there is to know about session variables in TrueUpdate.

In This Chapter

In this chapter, you'll learn about:

- Built-in and custom session variables
- Setting session variables
- Adding session variables
- Removing session variables
- Using session variables on screens
- Expanding session variables

What Are Session Variables?

Session variables are designed to handle dynamic data during the update process. Essentially “placeholders” for changeable text, session variables give you an easy way to insert dynamic values into the text that appears on your screens.

They also give you an easy way to compose paths to locations that cannot be known in advance, such as the path to the user’s My Documents folder. For example, you can use a session variable like %MyDocumentsFolder% in a path to be replaced by the appropriate full path at run time.

Like regular variables, session variables allow you to “store” information in them, acting like named “containers” that you can assign values to. The main difference between session variables and the “regular” variables you use in scripts is simply that session variables in a screen’s text are automatically expanded before the screen is shown. This makes them especially useful for displaying dynamic text on screens.

Even though all session variables are functionally identical, there are two distinct categories of session variables that can be used in TrueUpdate: built-in session variables, and custom session variables.

Built-in Session Variables

For convenience, TrueUpdate contains a variety of built-in session variables for values that are commonly used in projects. These variables are automatically assigned appropriate values when the TrueUpdate Client application is started.

Most of the built-in session variables hold information that has been gathered from the user’s system. For example, since the path to the Windows folder can differ between systems, a session variable named %WindowsFolder% is provided which automatically contains the correct path.

Note: Many of these values are also available in the form of global variables that you can use directly in your scripts, e.g. _WindowsFolder and _ProgramFilesFolder. There are also actions like Shell.GetFolder that you can use to get additional system paths. The built-in session variables are provided primarily for use in paths and default values that are displayed on screens.

Here is the list of built-in session variables, in alphabetical order:

%ApplicationDataFolder%

The path to the Application Data folder on the user's system. This folder serves as a common repository for application-specific data. Typically, this path is something like "C:\Documents and Settings\YourName\Application Data." On Windows Vista, it would return something like "C:\Users\YourName\AppData\Roaming."

%ApplicationDataFolderCommon%

The path to the all-user Application Data folder on the user's system. This folder serves as a common repository for application-specific data. Typically this is something like "C:\Documents and Settings\All Users\Application Data." On Windows Vista, this returns "C:\ProgramData."

%CommonFilesFolder%

The user's Common Files folder. Typically, this is something like "C:\Program Files\Common Files."

%CompanyName%

Your company's name. The value of this variable is set on the Session Variables tab of the Project Settings dialog.

%Copyright%

The copyright message for your product. The value of this variable is set on the Session Variables tab of the Project Settings dialog.

%DAOPath%

The path to the user's DAO (Data Access Objects) directory.

%DesktopFolder%

The path to the user's Desktop folder. On Windows NT/2000/XP/Vista, this is the path from the per-user profile.

%DesktopFolderCommon%

The path to the user's Desktop folder. On Windows NT/2000/XP/Vista, this is the path from the All Users profile. On a non-Windows NT system, this is the same as %DesktopFolder%.

%FontsFolder%

The path to the user's font directory (e.g. "C:\Windows\Fonts").

%MyDocumentsFolder%

The user's personal (My Documents) folder on their system. Usually this is something like "C:\Documents and Settings\YourName\My Documents" on Windows 2000/XP and "C:\My Documents" on Windows 98/ME, and "C:\Users\YourName\Documents" on Windows Vista.

%ProductName%

The name of the product that you are updating. The value of this variable is set on the Session Variables tab of the Project Settings dialog.

%ProgramFilesFolder%

The user's Program Files folder (e.g. "C:\Program Files").

%RegOwner%

The name of the registered user of the system.

%RegOrganization%

The organization of the registered user of the system.

%SourceDrive%

The drive that the TrueUpdate Client executable was run from (e.g. "C:" or "D:").

%SourceFolder%

The full path to the folder that the update executable was run from (e.g. "C:\Downloads" or "D:\").

%SourceFilename%

The full path, including the filename, for the current update executable. For example, if the user was running "update.exe" from "C:\Downloads," %SourceFilename% would be expanded to "C:\Downloads\update.exe."

%StartFolder%

The path to the user's Start menu folder. On Windows NT/2000/XP/Vista, this is the path from the per-user profile.

%StartFolderCommon%

The path to the user's Start menu folder. On Windows NT/2000/XP/Vista, this is the path from the All Users profile. On a non-Windows NT system, this is the same as %StartFolder%.

%StartProgramsFolder%

The path to the Programs folder in the user's Start menu. On Windows NT/2000/XP/Vista, this is the path from the per-user profile.

%StartProgramsFolderCommon%

The path to the Programs folder in the user's Start menu. On Windows NT/2000/XP/Vista, this is the path from the All Users profile. On a non-Windows NT system, this is the same as %StartProgramsFolder%.

%StartupFolder%

The path to the user's Startup folder. On Windows NT/2000/XP/Vista, this is the path from the per-user profile.

%StartupFolderCommon%

The path to the user's Startup folder. On Windows NT/2000/XP/Vista, this is the path from the All Users profile. On a non-Windows NT system, this is the same as %StartupFolder%.

%SystemFolder%

The path to the user's Windows System folder (e.g. "C:\Windows\System").

%SystemDrive%

The drive that the user's Windows System directory is located on (usually "C:").

%TempFolder%

The path to the user's Temp folder.

%WindowsFolder%

The path to the user's Windows folder (e.g. "C:\Windows").

Custom Session Variables

You can define your own session variables to supplement the built-in session variables that are automatically provided in TrueUpdate. The session variables that you define are known as "custom" session variables.

Custom session variables can be used everywhere that built-in session variables can be used; in fact, they are functionally identical. The only difference is that the built-in session variables are automatically created for you in each project, whereas custom session variables don't exist until you assign a value to them.

Setting Session Variables

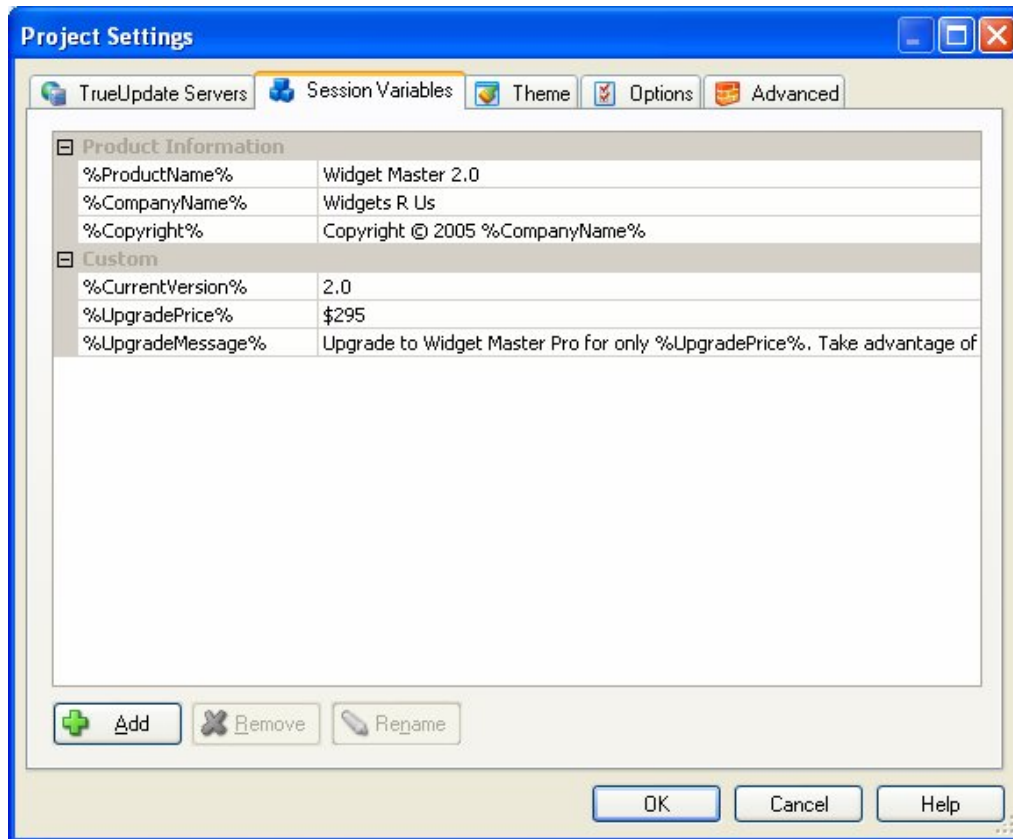
Each session variable consists of a name, e.g. "%ProductName%," and a value that you assign to it, e.g. "Widget Master 2.0." When a session variable is expanded at run time, its name is replaced by the value that is currently assigned to it. (For example, "Thank you for using %ProductName%" would become "Thank you for using Widget Master 2.0.")

There are two ways you can assign a value to a session variable: you can set its initial value on the Session Variables tab, or you can use an action to set its value anywhere in your project.

Using the Session Variables Tab

The Session Variables tab provides a convenient location for setting the initial value of session variables at startup. It is primarily used for values that need to be displayed on the client screens and that don't need to be determined dynamically at run time using actions. In other words, it is where you can specify values that you know in advance and that you want to display on the earliest screens in your project.

You can access the Session Variables tab by choosing Project > Session Variables from the program menu.



All session variables defined on the Session Variables tab are stored in the client data file. This means that any changes to the list of session variables will in turn change the client data file. TrueUpdate handles changes to the client data file automatically—in fact, a copy of the latest client data file is stored at each TrueUpdate Server location so the TrueUpdate Client applications can make sure they are always using the latest version. In order for a TrueUpdate Client to update its data file, however, it needs to restart itself. If you foresee needing to change the values of these session variables often, you should consider whether they could be defined in your Server Script using actions instead. Since changes to the Server Script aren't stored in the client data file, they don't cause the TrueUpdate Client application to restart.

Tip: Generally, session variables that aren't displayed on the client screens should be created from your Server Script so they can be changed without affecting the client data file.

The Session Variables dialog contains two categories: Product Information and Custom.

The Product Information category contains three built-in session variables for values that are commonly displayed on screens, such as the product name and the name of the company that produced it. To set the value of one of these variables, simply edit the appropriate field in the right-hand column.

The Custom category is where you can add, remove and edit your own session variables to supplement the ones in the Product Information category. To add a custom session variable, click on the Add button at the bottom of the dialog.

Tip: As the session variable list grows, it may help to hide portions of the list. Each category can be expanded or collapsed by clicking the “+” icon on the left hand side of the category text.

Using Actions

An action is also available to set the value of a session variable. This action is called SessionVar.Set. It allows you to set the value of an existing session variable that was defined on the Session Variables tab, or to create a brand new one.

The SessionVar.Set action can be used with any event (i.e. in any script) throughout the project. The function prototype for this action is:

```
SessionVar.Set(string VariableName, string Value)
```

For example, if you want to assign the value “My Value” to a session variable named %MyVar%, the action would look like this:

```
SessionVar.Set("%MyVar%", "My Value");
```

After the above action is performed, all occurrences of the text %MyVar% on future screens will be replaced with the text “My Value.”

Note: The SessionVar.Set action works with *all* session variables, including built-in session variables like %MyDocumentsFolder%. It is possible to overwrite a built-in session variable’s value using the SessionVar.Set action, so be very careful when setting session variables with actions. Under normal circumstances, there should be no reason to modify the values of built-in variables.

Removing Session Variables

When you remove a session variable from your project, TrueUpdate will no longer recognize the variable's name as special placeholder text. For example, removing the session variable %ProductName% causes the name to revert back to its actual characters. In other words, the text “%ProductName%” ceases to be anything other than the letters %, P, r, o, d, u, c...and so on. After the session variable is “removed,” there is no longer a value associated with the name, and no expansion occurs.

There are two methods for removing session variables from your project: using the Session Variables tab, or using actions.

Using the Session Variables Tab

Similar to adding session variables, removal of session variables can also be accomplished from the Session Variables tab. However, only those in the Custom category can be removed from your project. To remove a custom session variable, click on the desired session variable name in the list to highlight it, and then click on the Remove button. The session variable will be removed from the list.

Using Actions

Session variables can also be removed at any point during your update with an action. The action used to remove a session variable is called SessionVar.Remove and can be found in the “SessionVar” action category. The function prototype for this action can be seen below:

```
SessionVar.Remove(string VariableName)
```

For example, if you want to remove a session variable called %MyVar%, the action script would look like the following:

```
SessionVar.Remove( "%MyVar%" );
```

Note: Since both custom and built-in session variables behave the same, it is possible to remove a built-in session variable using the SessionVar.Remove action. For this reason, extra care should be taken when removing session variables with actions.

Using Session Variables on Screens

The main use of session variables is for the dynamic expansion of text strings on screens. One example of a valuable use of session variables is when you need to use a value on multiple screens, such as a product version number. While you can certainly enter the text directly for each screen, if that string changes in the future, it would require finding every location where it is used in order to change its value. Using a session variable in place of that text would only require the modification of the session variable's value in one location.

Another valuable use of session variables is for gathering data on one screen that needs to be displayed on another screen. In this case, the values are not known until some point during the update, and therefore could not be directly entered at design time.

Tip: Session variables can also be useful in multilingual updates for custom messages that you wish to display depending on the language detected or chosen.

When Are Session Variables Expanded?

Session variables are automatically expanded before each screen is shown—specifically, before each screen's On Preload event. Any session variable that is used on a screen will automatically display the value it contained *before* that screen was shown.

This means that if you change the value of a session variable in a screen's On Preload event, in most cases the old value will still appear. There are a few exceptions, such as some static text controls which will automatically be “refreshed” after the On Preload event and will therefore display their new values. As a general rule, however, the On Preload event is already “too late” for any changes to a session variable to be made if you want the new value to automatically appear on the screen.

One way to get the current value from a session variable is to expand it “manually” using the SessionVar.Expand action. SessionVar.Expand allows you to retrieve the current value of a session variable at any point in your project. In fact, you can use SessionVar.Expand on a screen's On Preload event to retrieve a session variable's value, and then use actions to replace the screen text with new text that includes the current value.

Expanding Session Variables in Scripts

Session variables are often used on screens that gather information from the user. For example, the Edit Fields screen stores the user's input in separate session variables—one session variable for each edit field on the screen. This is fine if you simply want to display the user's input on another screen; in that case, all you need to do is include the appropriate session variables in that other screen's text. If you want to use the inputted values in your scripts, however, you need a way to expand the session variables in your script. This can easily be accomplished using the `SessionVar.Expand` action.

The function prototype for this action is:

```
string SessionVar.Expand(string Text);
```

Basically, the `SessionVar.Expand` action takes a string of text and gives you back the same text, but with all of the session variables in the string expanded. In other words, it returns a copy of the text in which all of the session variables have been replaced by their current values.

For example, if a session variable called `%MyName%` contains the string “They call me nobody,” you can access the string using the following action script:

```
strContents = SessionVar.Expand( "%MyName%" );
```

In the above line of script, the variable `strContents` would receive an expanded version of “%MyName%.” The end result is that the value stored in `%MyName%` (“They call me nobody”) would be assigned to `strContents`.

However, `SessionVar.Expand` isn't limited to retrieving the contents of a single session variable. It will happily expand a string containing several session variables—or even one containing no session variables at all. (In the latter case, the string it returns will be an exact copy of the original string.)

For example, using `SessionVar.Expand` on the string “When asked for his name, all he said was: %MyName%” would return the entire string with the expanded contents of the session variable `%MyName%`:

When asked for his name, all he said was: They call me Nobody

In addition, when the `SessionVar.Expand` action expands a string, it performs a *recursive* expansion. Session variables within session variables are expanded as well. This means that if the value in a session variable is a string that has a session variable

in it, the “internal” session variable will also be expanded. You can think of the expansion as being a “loop” that continues until there are no more session variables left to expand.

For example, consider the following two session variables:

%verb% - whose value is “flying”

%message% - whose value is “Look at me, I’m %verb%!”

After the following action script is executed:

```
strContents = SessionVar.Expand( "%message%" );
```

...the contents of the variable strContents would be:

Look at me, I’m flying!

As you can see, %message% was replaced by “Look at me, I’m %verb%!” and then %verb% was replaced by “flying.”

Expanding Without Recursion

TrueUpdate also contains an action that will prevent the recursive expansion of session variables. This action is called SessionVar.Get and can also be found in the “SessionVar” actions category. The function prototype is:

```
string SessionVar.Get(string Text);
```

Using the previous example, let’s say we only wanted to expand the contents of %message%, without expanding %verb%. In that case, the following action script could be used:

```
strContents = SessionVar.Get( "%message%" );
```

...and the contents of the variable strContents would be:

Look at me, I’m %verb%!

As you can see, the SessionVar.Get action would expand the %message% session variable, but wouldn’t go any further; the %verb% variable would remain unexpanded.

Expanding after On Preload

Session variables are automatically expanded before each screen is shown. This automatic expansion happens before any of the screen's events are triggered—even before the earliest screen event, On Preload. This means that if you use the SessionVar.Set action to change the value of a session variable from On Preload, the new value will not appear on the screen, because at that point the session variables have already been expanded. (There are exceptions to this rule, such as the static text controls on some screens, but it is safer to assume that it is true in all cases.)

In order to change the text on a screen from within that screen's events, you must use a different method. Luckily, there are two ways in which you can expand session variables on the current screen.

The first and most straightforward method is to formulate a new text string with the session variable in it, expand that string using SessionVar.Expand, and then assign the expanded string to the desired screen control.

For example, the following script would expand a string that contains two session variables and then replace the text on a scrolling text screen with the new text:

```
NewText = SessionVar.Expand("First: %FirstName%\r\nLast: %LastName%");
DlgScrollingText.SetProperties(CTRL_SCROLLTEXT_BODY, {Text=NewText});
```

Although this method works well, it requires you to write and edit the text within your script, instead of composing the text directly on the Settings tab for that screen. This isn't too difficult if your project only supports one language. However, if you're creating a multilingual project, you'll need to use control structures to assign different text to the screen that is appropriate for the user's system language. This would make your scripts much longer and more difficult to maintain.

An alternative method is to use an action to retrieve the original text for the screen, which still contains the session variables in their unexpanded form. This allows your script to essentially "re-expand" the text that you entered on the Settings tab.

Here is a version of the previous script that uses the Screen.GetLocalizedString action to retrieve the original unexpanded text for the scrolling text control:

```
OriginalText = Screen.GetLocalizedString(IDS_CTRL_SCROLLTEXT_BODY);
NewText = SessionVar.Expand(OriginalText);
DlgScrollingText.SetProperties(CTRL_SCROLLTEXT_BODY, {Text=NewText});
```

The key to this second approach is the `Screen.GetLocalizedString` action, which retrieves the text for a specific screen control as it was entered on the Settings tab, automatically choosing the appropriate text for the language on the user's system.

When your project supports multiple languages, it is much easier to edit the screen text directly on the Settings tab, where you can use the language selector to switch between all of the languages that your project supports.

Tip: If you would like to see an advanced example of session variables in use, examine some of TrueUpdate's built-in screens. For example, the Select Drive screen uses actions to set, update, and display the session variable `%SpaceAvailable%`.

Chapter 8:

Languages

The Internet has opened many new markets to software developers whose past products would usually have supported only their own local language. In the international marketplace, it is important to not only offer software in a variety of languages, but also to keep this software current.

As you'll see in this chapter, you can use TrueUpdate to create an update that will automatically display messages and prompts in your user's native language. With integrated language selection built into all screen dialogs, TrueUpdate also makes it very easy to modify your existing translations at any time.

In This Chapter

In this chapter, you'll learn about:

- Internationalizing your project
- How language detection works
- The language manager
- Language files
- Setting the default language
- Adding and removing languages
- The language selector
- Localizing screens and actions
- Customizing error messages and prompts
- Advanced techniques, such as using actions to determine the current language and “changing” the current language for testing purposes

Internationalizing Your Project

TrueUpdate has the ability to automatically detect the user's system's language and to display messages and screens in that language. The developer has full control over which languages are supported in their project and over the content presented to the user.

Language text is mainly used for messages generated throughout the update and on screens, both of which can be easily translated for multilingual updates.

TrueUpdate allows you to localize your application in two areas:

- Common error messages, status messages and prompts
- Application-specific screens

The following sections of this chapter will look more closely at how to achieve this localization.

Run-time Language Detection

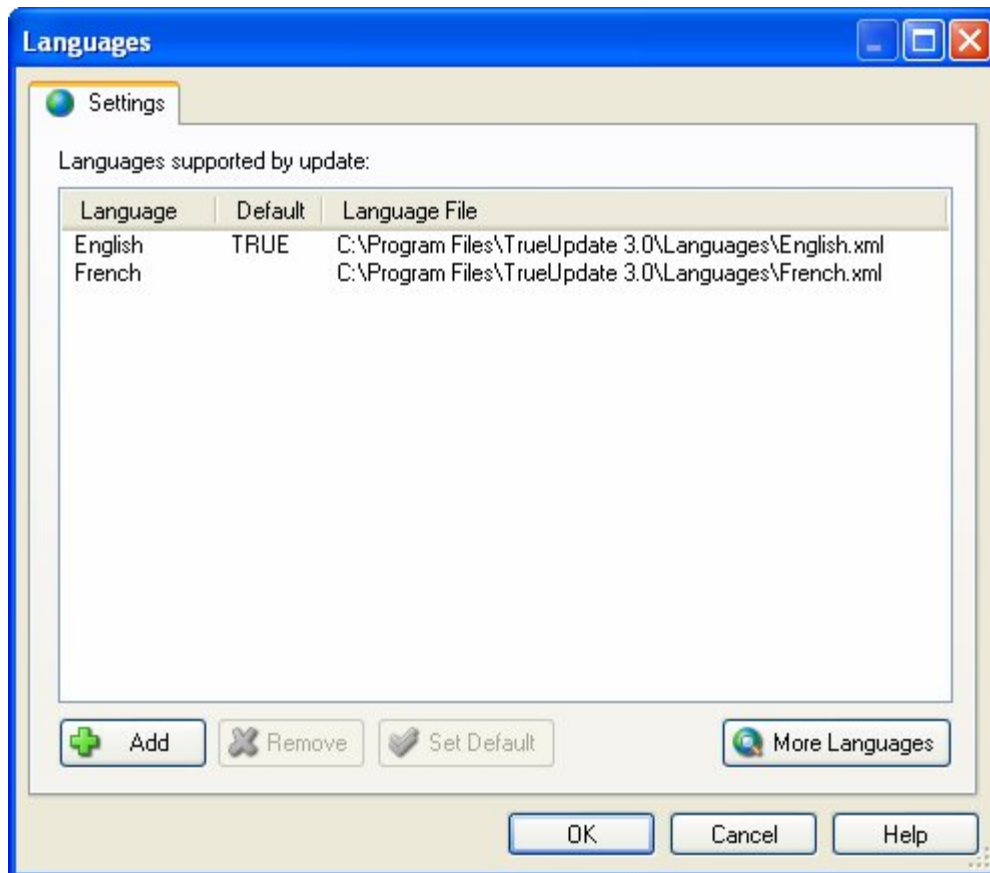
The language that the TrueUpdate Client detects is based on the user's regional and language settings. These settings allow Windows users to configure which language is displayed, which input locale is used and which keyboard layout is supported in the Windows operating system environment. These settings are usually configured when Windows is installed and can usually be changed from the Windows Control Panel. For example, in Windows XP, a user can select Start > Settings > Control Panel and start the Regional and Language Settings control panel application.

Each language in Windows has a constant language identifier. A language identifier is a standard international numeric abbreviation for the language that is used in a country or geographical region. Each language identifier is made up of a primary and secondary language ID. (There is a complete list of primary and secondary language IDs in the TrueUpdate help file.) TrueUpdate maps all known languages and sub-languages according to the language identifiers used by Windows.

Tip: TrueUpdate maps language identifiers to language names in a file called `language_map.xml` located in TrueUpdate's Language folder (usually "`C:\Program Files\TrueUpdate 3.0\Languages`"). You can look at this file to see which primary and secondary language IDs are mapped to which languages. It is NOT recommended that you modify this file unless you have a very specific reason to do so.

The Language Manager

The languages that are supported by your client are all configured from the Language Manager. You can access the language manager by selecting **Project > Languages** from the menu.



Although you can localize messages in several areas of the design environment, the Language Manager is the only place where you can control the project's default language as well as which languages are supported by your update. For example, if you are editing one of your screens and decide that you would like to add a German translation, you will have to use the Language Manager to do so. Once you add German support here, it will be available in all other areas of your project.

When you add a language to your project, you are indicating that you want the client to recognize that particular language identifier on a system and to use specific messages for that language when identified. Conversely, if a system's language is not represented in the languages list, it does not mean that the update will not run on that system; rather, it means that the update will use the default language.

Default Language

Every project must have a default language. The default language is the one that will be used when the client encounters a system language that is not represented in the languages list.

For example, let's suppose that you have English, French and German support in your update with English as the default language. If your user runs the update on a Greek system, the user will see the English messages since you did not specifically include support for the Greek language.

Note that the default language must be one that has a corresponding language file (see the next section).

Language Files

A language file is an XML file that contains all of the "internal" error messages, status messages and prompts that are used by the TrueUpdate Client. Language files do not contain project-specific messages, such as the text that you enter on screens.

The language files are located in TrueUpdate's Languages folder (usually "C:\Program Files\TrueUpdate 3.0\Languages"). They are named according to the English name for the language they represent. Each file contains a language map that identifies which language the file is responsible for and all of the built-in messages that will be used for that language.

Not all languages have a pre-configured language file. If you add a language to your

project that does not have a language file, that language will use the same messages as the default language.

Getting Additional Language Files

If you need a language file that is not shipped with TrueUpdate, please visit the Indigo Rose website (www.indigorose.com) and user forums (www.indigorose.com/forums/) where new language files are made available from time to time.

Making Your Own Language File

If after consulting the Indigo Rose web site you still can't find the language file you need, you can always make one yourself. To make a new language file, simply make a copy of the existing language file that you want to translate from, rename it to the new language name, change the language map in the file accordingly, and then translate the messages.

To clarify this process, here is an example of how to create a French language file:

1. Open Windows Explorer to TrueUpdate's Languages folder (usually "C:\Program Files\TrueUpdate 3.0\Languages").
2. Make a copy of English.xml and name it French.xml.
3. Open French.xml in a text editor such as Notepad.
4. Open language_map.xml from the Languages folder in a text editor as well.
5. Locate the section that maps French in the language_map.xml file. It should look like this:

```
<Language>
  <Name>French</Name>
  <Primary>12</Primary>
  <Secondary>
    <ID>1</ID>
    <ID>2</ID>
    <ID>3</ID>
    <ID>4</ID>
    <ID>5</ID>
    <ID>6</ID>
  </Secondary>
</Language>
```

6. Copy the above section from language_map.xml and paste it in place of the `<Language></Language>` section of the French.xml file. This will allow TrueUpdate to recognize this file as a language file for the French language.
7. Translate all messages in the `<Messages></Messages>` section to French. Do not change any actual XML tags. For example:

`<MSG_SUCCESS>Success</MSG_SUCCESS>`

becomes:

`<MSG_SUCCESS>Succès</MSG_SUCCESS>`
8. Save the file and re-open TrueUpdate. The new language will now be available.
9. Feel free to visit the Indigo Rose user forums (www.indigorose.com/forums/) and share the file with others so they can use your translations in their projects.

Adding Languages

To add a new language to your project, click the Add button in the Language Manager. This will open the Add New Language dialog.

Simply select the language that you want to add and click OK. You will then see the language appear in the languages list.

What Happens When You Add a New Language

When you add a new language to your project, the following happens automatically:

- TrueUpdate searches the Languages folder for an appropriate language file for the language. If one is not found, the new language is set to use the default language's language file.
- The newly added language is added to all screens. That is, all screens have messages added to them for the new language. If a translated language file for that particular screen already exists, it will be used. Otherwise, the messages from the default language will be replicated for the new language.
- The language is added to the list of languages that you can select from in the language selectors throughout the design environment.

Of course, you will still need to go into the screen dialogs to verify and/or translate the text for the new language.

Removing Languages

To remove a language from the project, select it in the Language Manager's list and click the Remove button.

Note that the default language cannot be removed. In order to remove a language that is currently being used as the default language for a project, you will need to make another language the default language first.

When you remove a language, all of the translations for that language are removed from the screens in the project. Therefore, use caution when removing languages.

The Language Selector

Once a language is added to your project, it is available for translation. When you're editing the text on a screen, you can select the language that you want to edit by using the *language selector*. The language selector is simply a drop-down list that lets you choose the current language (for editing purposes) of the screen.

For example, if your project supports English, French and German, the language selector on every screen's properties dialog will let you choose whether to edit the English, French, or German text.



Selecting a language in the language selector replaces the editable text on the dialog with the text for that language. Any changes that you make to the screen text will be restricted to that language.

Note: If you select a newly added language for which there is no language file, the editable text will initially be the same as the text for the default language.

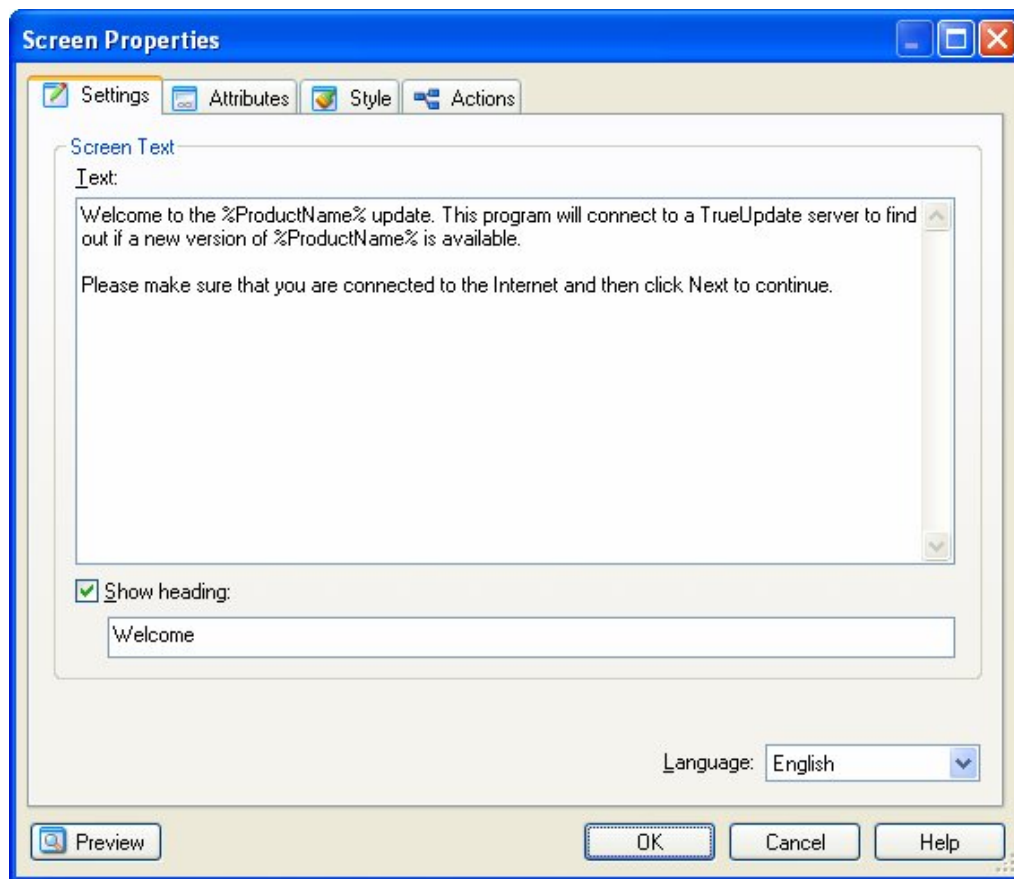
Only the languages that have been added to the project will appear in the language selector. If you want to work on a language that is not in the drop-down list, you will first have to add it to the project using the Language Manager.

Localizing Screens

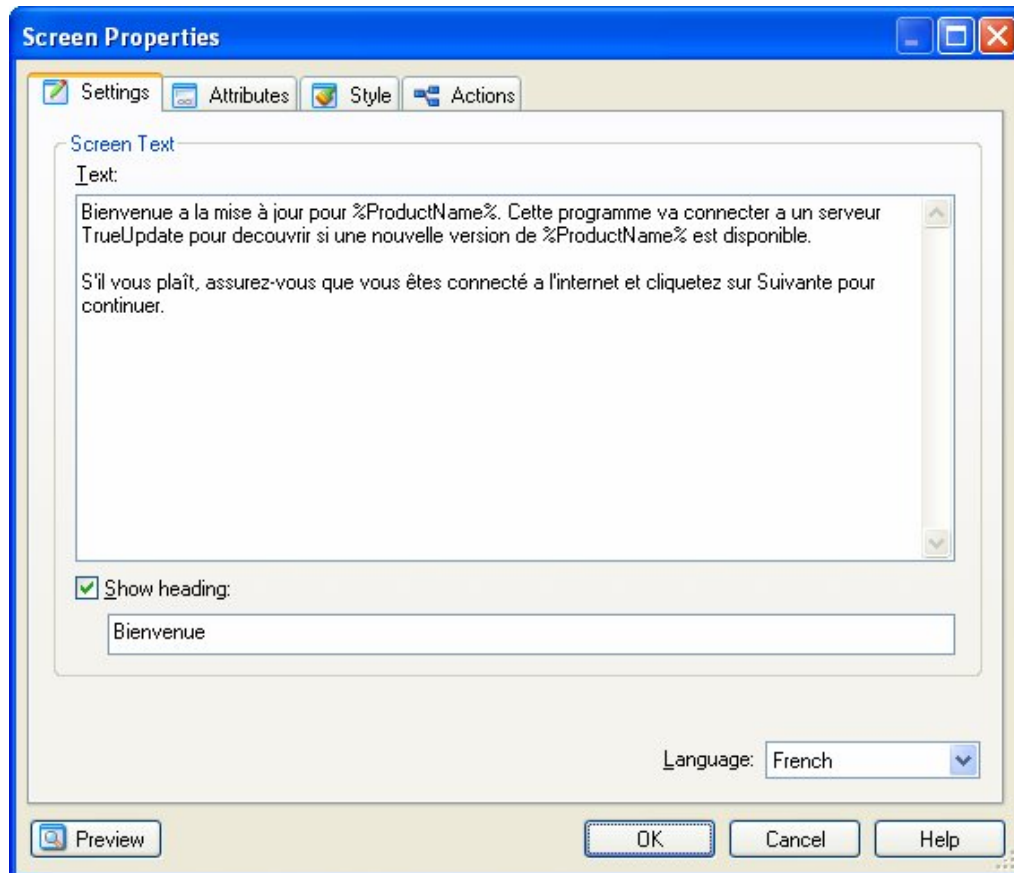
To localize a screen, open the specific screen's properties. Next, select the language that you want to enter text for in the language selector. Then, simply type the text that you want for the various fields in that language.

Note: The Screen ID field on the Attributes tab cannot be translated. The Screen ID is a unique identifier for the screen and is never displayed to the user.

Here is an example of a Welcome screen with English text:



And here is the same screen with French text:



Notice that in the second screenshot, “French” has been selected in the language selector near the bottom of the dialog. The text that you enter always corresponds to the language that is selected in the language selector.

Tip: If the language that you want to translate to doesn’t appear in the language selector, you need to add support for that language to your project. This is done by adding the language in the language manager. For more information, see *The Language Manager* beginning on page 155 and *Adding Languages* on page 158.

Importing and Exporting Screen Translations

There may be times when you want to have your screens translated by a third party translator. If the translator owns a copy of TrueUpdate, you can simply send them your project file, have them translate the screens using the method explained above, and then have them send the project file back to you.

However, it may be that the translator does not own TrueUpdate or that you need to work with the project in other ways while the translation is taking place. TrueUpdate has a solution for this situation. You can:

1. Select the screen that you want to have translated in either the client or server screens list.
2. Click the Advanced button (this will open a popup menu).
3. Select Export Language and then the language that you want to export to.
4. Choose a location to save the file to.
5. Send the exported file to your translator.
6. When you receive the file back, select the screen in the Screens list.
7. Click the Advanced button and then choose Import Language from the popup menu.
8. Locate the translated file and select Open.
9. You will now have the new translated strings in your screen.

Customizing Error Messages and Prompts

If you want to change the default error messages, prompts or status messages, you can edit the appropriate XML file in the Languages folder using a text editor such as Notepad. However, this is not generally recommended; the messages that you change will be propagated to all projects that are built after the changes are made.

Note also that if you choose to change the default messages, your modified language file may be overwritten by a future update to TrueUpdate. To avoid this, you may want to rename any language files that you modify, for example renaming english.xml

to my_english.xml. (In cases where there is more than one language file for a given language, TrueUpdate will use the last one that it finds in the Languages folder.)

```
<Messages>
  <MSG_SUCCESS>Success</MSG_SUCCESS>
  <MSG_ERROR>Error</MSG_ERROR>
  <MSG_NOTICE>Notice</MSG_NOTICE>
  <MSG_WARNING>Warning</MSG_WARNING>
  <MSG_YES>Yes</MSG_YES>
  <MSG_NO>No</MSG_NO>
  <MSG_YES_TOALL>Yes to All</MSG_YES_TOALL>
  <MSG_TO>to</MSG_TO>
  <MSG_FROM>from</MSG_FROM>
  <MSG_BROWSE>Browse...</MSG_BROWSE>
  <MSG_OK>OK</MSG_OK>
  <MSG_CANCEL>Cancel</MSG_CANCEL>
  <MSG_PATH>Path</MSG_PATH>
  <MSG_SEARCH_MASK>Search</MSG_SEARCH_MASK>
  <MSG_SEARCH_ALL>All Files</MSG_SEARCH_ALL>
  <MSG_SEARCH_FILE>Searching for file</MSG_SEARCH_FILE>
  <MSG_SIZE_BYTES>bytes</MSG_SIZE_BYTES>
  <MSG_SIZE_KILOBYTES>KB</MSG_SIZE_KILOBYTES>
  <MSG_SIZE_MEGABYTES>MB</MSG_SIZE_MEGABYTES>
  <MSG_SIZE_GIGABYTES>GB</MSG_SIZE_GIGABYTES>
  <MSG_BITSPERPIXEL>BPP</MSG_BITSPERPIXEL>
  <MSG_CONFIRM>Confirm</MSG_CONFIRM>
  <MSG_CONFIRM_ABORT>The update is not finished! Do you really want to abort?</MSG_CONFIRM_ABORT>
  <MSG_CONFIRM_CONTINUE>Are you sure you want to proceed with the update?</MSG_CONFIRM_CONTINUE>
  <MSG_NOT_ENOUGH_FREE_SPACE>There is not enough free space to update %ProductName% on
  <MSG_COPYING>Copying</MSG_COPYING>
  <MSG_DELETING>Deleting</MSG_DELETING>
  <MSG_SEARCHING>Searching</MSG_SEARCHING>
```

An excerpt from english.xml

Advanced Techniques

There are a number of advanced techniques that you can use to manipulate the language and the translated language strings in your update at run time. Most of these methods are accomplished using actions. This section covers a few of these advanced techniques.

Determining the Current Language

There are two actions that can be used to retrieve information about the user's language ID: System.GetDefaultLangID and Application.GetUpdateLanguage. Although both actions return a table of information containing language IDs, it is important to know the difference between the two.

System.GetDefaultLangID

System.GetDefaultLangID is used to get the primary and secondary language ID for the language that the user employs in Windows. This is absolute and cannot be changed with any other actions. For example, if this action returns 10 as the primary ID and 13 as the secondary ID, you will know that the user's Windows system is configured for Spanish (Chile). (There is a complete list of primary and secondary language IDs in the TrueUpdate help file.)

The information returned by this action can be used in cases where you want to make specific choices about what to do based on the user's absolute system language. For example, if you have a Web site that is translated into several different languages, you might want to use a series of if...then statements to open the appropriate site:

```
-- Determine the absolute system language
local tblSysLang = System.GetDefaultLangID();

-- Set a default
local strURL = "http://www.yourcompany.com/english";

-- See if we should go to a different site
if (tblSysLang.Primary == 7) then
    strURL = "http://www.yourcompany.com/german";
elseif (tblSysLang.Primary == 10) then
    strURL = "http://www.yourcompany.com/spanish";
elseif (tblSysLang.Primary == 16) then
    strURL = "http://www.yourcompany.com/italian";
end

-- Go to the Web site
File.OpenURL(strURL);
```

Application.GetUpdateLanguage

Application.GetUpdateLanguage is used to retrieve the primary and secondary language ID that is actually being used by the TrueUpdate Client. Note that Application.GetUpdateLanguage may return a different result than System.GetDefaultLangID if the language being used by the client differs from the language your user employs in Windows.

For example, let's say that you have three languages in your project: English (which is the default language), French, and German. Suppose a user from Chile runs the update on their system. Even though their system uses primary ID 10 and secondary ID 13 (which would be returned by System.GetDefaultLangID and corresponds to a dialect

of Spanish), `Application.GetUpdateLanguage` would return a primary ID of 9 and a secondary ID of 1, which is the update's default language (English). The TrueUpdate Client would be using the default language because Spanish was not added to the project at design time.

The value returned by `Application.GetUpdateLanguage` will always be a value for a language that you added to your project using the Language Manager. It will never contain values for languages that were not explicitly included in the project.

Changing the Current Language

Normally, if TrueUpdate detects a language on the user's system that is not supported in the Language Manager, the default language will be used. However, you may prefer to have your update use a different language in such situations. You can accomplish this by using the `Application.SetUpdateLanguage` action.

The `Application.SetUpdateLanguage` action allows you to directly set the primary and secondary language IDs that will be used for the update. Calling this action changes all subsequent error and status messages as well as the text shown on the screens. It effectively "forces" the TrueUpdate Client to act like it detected that language in the first place.

For example, let's say that your project supports two languages: English (which is the default language) and Simplified Chinese. Since English is the default language, it will be used whenever the client is run on anything other than Simplified Chinese.

However, you might prefer that the client use Simplified Chinese if the user runs the client on a system configured to use Traditional Chinese.

In other words, you want to override the default language rule and force your client to use Simplified Chinese whenever Traditional Chinese is detected. You can do so easily by placing the following short script at the beginning of your client script, before any screens or dialogs are shown:

```
-- Determine the absolute system language
local tblSysLang = System.GetDefaultLangID();
if (tblSysLang.Primary == 4) and (tblSysLang.Secondary == 1) then
    -- Traditional Chinese on user's system,
    -- so use Simplified instead
    Application.SetUpdateLanguage(4, 2);
end
```

Testing Different Languages

You may also want to use the `Application.SetUpdateLanguage` action for testing purposes. For example, if you are running an English version of Windows, you might want to see how your update will look on an Italian system. Because your system is running in English, the client will always choose English as the language to display when you run it on your system. However, you could force the client to use Italian by putting the following script at the beginning of your client script:

```
Application.SetUpdateLanguage(16, 1);
```

You could even modify your client script to check for a custom command line option, so you could force your TrueUpdate Client to use a different language at any time. This could be useful for testing purposes, or to handle any language detection issues that are discovered after the client has been distributed.

Tip: The global variable `_CommandLineArgs` can be used to determine what arguments were passed to the TrueUpdate Client executable.

Localizing Actions

You may have noticed that there is no language selector when editing scripts in TrueUpdate. Any text you enter directly into a script will remain the same, regardless of what language is detected on the user's system.

However, it is possible to use actions to detect the current language, and to use multiple “if” statements to specify different text for different languages.

For example, let's say that your project supports English and French and you want to show a dialog box using actions that will be localized according to those languages. The following script first determines the language that the client is using, and then displays one of two possible greetings:

```
local tblSysLang = Application.GetUpdateLanguage();

if (tblSysLang.Primary == 9) then
    Dialog.Message("Welcome",
                  "Welcome to the update");
end

if (tblSysLang.Primary == 12) then
    Dialog.Message("Bienvenue",
                  "Bienvenue à la mise à jour");
end
```

The `Application.GetUpdateLanguage` action returns a table containing the primary and secondary language ID that is being used by the TrueUpdate Client. Note that this may or may not be the exact language that was detected on the user's system; rather, it is the appropriate language that the TrueUpdate Client has chosen from the list of supported languages in the project. In other words, it is the user's system language if that language is supported by the update; otherwise, it is the default language.

Tip: You can get the user's *actual* system language by using the `System.GetDefaultLangID` action. However, it is usually preferable to use the `Application.GetUpdateLanguage` action so the scripted language behavior will match the "automatic" language behavior of the screens and error messages in the project.

Working with Existing Translated Messages

TrueUpdate allows you to get and set translated messages from the language files and screens at run time. This is done through several actions.

TrueUpdate.GetLocalizedString

This action allows you to retrieve the localized text for a general message from the language files at run time. The message will be returned in the current update language. For example, the default language files provide messages to confirm if the user wants to abort the update. Here is a way to show a dialog that confirms if the user wants to abort the update in the current language:

```
local strTitle = TrueUpdate.GetLocalizedString("MSG_CONFIRM");
local strPrompt =
TrueUpdate.GetLocalizedString("MSG_CONFIRM_ABORT");
local nResult = Dialog.Message( strTitle
                                , strPrompt
                                , MB_YESNO
                                , MB_ICONQUESTION
                                , MB_DEFBUTTON2 );

if(nResult == IDYES)then
    Application.Exit();
end
```

Note: The message IDs (like `MSG_CONFIRM`) for the localized message strings can be found in the XML language files, e.g. `English.xml` in the TrueUpdate Languages folder.

TrueUpdate.SetLocalizedString

This action allows you to change the value of a localized string. This can be useful if you want to override the default value of an error message from script, so you don't have to permanently change your language file.

For example, let's say that you want to change the message that is displayed if the user tries to cancel the update. By default it is "The update is not finished! Do you really want to abort?", but you want to change it to: "Stopping now is not a good idea. Are you sure?"

```
TrueUpdate.SetLocalizedString("MSG_CONFIRM_ABORT",  
    "Stopping now is not a good idea. Are you sure?");
```

Screen.GetLocalizedString and Screen.SetLocalizedString

These actions are used to get and set the value of a localized string from the current screen's message file. Every editable text item on a screen has a corresponding string ID that is used internally to retrieve the appropriate text for the current language when the screen is displayed.

These actions can be used to create new, temporary localized strings that are only valid on the current screen. In other words, they permit you to define your own localized strings (using custom string IDs) for use on the current screen.

For example, you could create custom localized error messages for use in a screen's event scripts, without having to implement "if...then" branching to choose the appropriate translated text wherever an error message can be displayed.

Note: The `Screen.GetLocalizedString` and `Screen.SetLocalizedString` actions work the same way that `TrueUpdate.GetLocalizedString` and `TrueUpdate.SetLocalizedString` do, except that they will only access strings used on the current screen.

Chapter 9:

Security

Any time data is exchanged over the Internet security is an issue. TrueUpdate normally relies on Internet communication to update software; therefore, it is also subject to Internet security issues.

This chapter will explain the security measures that have been implemented in TrueUpdate and outline ways in which you can further enhance your update application's security.

In This Chapter

In this chapter, you'll learn about:

- Client side security
- How the server files are secured
- Client-server communication
- The secure protocols supported by TrueUpdate
- Custom client-server communication
- Important security considerations

Security in TrueUpdate

The security of your product is only as strong as its weakest link. For that reason, measures have been taken in the design and development of TrueUpdate to provide as much security as is practical for the product.

Since TrueUpdate was designed to be flexible enough to accommodate your specific updating needs, there are security measures that can only be implemented by you, the developer. In this chapter, we will examine TrueUpdate's built-in security features and the additional measures you can take to further secure your software updates.

Client Side Security

TrueUpdate takes steps to ensure that the update files you distribute with your software are secure. The details of your update application, such as scripts and project settings, are hidden from casual inspection.

In general, you will distribute two files with your update application: the client executable file and the client data file. Below is a short overview examining how these files and the file updating process are protected.

Client Executable File

The client executable file does not contain any information specific to your update application. It is a runtime executable that is used by all TrueUpdate users. For this reason there is no special protection applied to it.

Client Data File

The client data file contains your client script, screens and most of the project settings you configured at design time. Since some of this information can be sensitive, the file is compressed and encrypted to prevent unauthorized inspection or tampering.

At run time the scripts, screens and settings in the client data file are read directly into memory, without being written to the hard drive (even temporarily) in any unencrypted form. The only files that are temporarily created by the TrueUpdate Client are not sensitive in nature, and are deleted as soon as the update application shuts down.

Client File Updating

Every time you publish your update, the MD5 hash of the current client data file is stored in the server data. This allows the client application to automatically determine whether its data file is up to date. If the MD5 hash of the client data file on the user's system does not match the one that the server file was published with, the client data file will automatically be downloaded and updated. So, if your client data file is tampered with on the client system (e.g. if it is infected with a virus, or damaged by the user), it will be removed and updated the next time an update is requested.

How MD5 Security Works

An MD5 hash is a digital signature that can be used to uniquely identify any piece of data. It is the result of a special calculation that can be applied to any kind of data. This calculation process is strictly one-way. Although you can calculate the MD5 hash for any value, you *cannot* later retrieve the value from the MD5 hash. It is currently impossible to reverse the process.

So, if you calculate the MD5 hash of a file, that MD5 hash will always be the same for that exact file. If even one bit of data in the file is modified, the MD5 hash will be different. This method is a great way to ensure data integrity.

Note that you can calculate and validate MD5 hashes on your own files using the Crypto plugin available for TrueUpdate.

Server Files Security

Your TrueUpdate server files are especially vulnerable because they are usually publicly available on your TrueUpdate servers. For this reason, there are strong security measures in place to protect them.

Types of Server Files

When you publish your update to a TrueUpdate Server, several files are uploaded. The files all use your custom server file prefix for file names and have the file extensions .ts1, .ts2 and .ts3.

Server Data File

The server data file has the extension .ts1 on the TrueUpdate Server. This file contains all of your server scripts and screens. This file is the most critical security concern as it contains information about how you update your software and may also contain sensitive passwords and file locations.

The server data file is encrypted using private-key Blowfish encryption. The private key is the unique, product-specific key that you specify in TrueUpdate at design time (see Project > Options from the menu.) This private key is stored in the client data file, which is also encrypted using a different encryption scheme.

What is Blowfish Encryption?

Here is a definition from the official Blowfish Web site (www.schneier.com):

“Blowfish is a symmetric block cipher that can be used as a drop-in replacement for DES or IDEA. It takes a variable-length key, from 32 bits to 448 bits, making it ideal for both domestic and exportable use. Blowfish was designed in 1993 by Bruce Schneier as a fast, free alternative to existing encryption algorithms. Since then it has been analyzed considerably, and it is slowly gaining acceptance as a strong encryption algorithm. Blowfish is unpatented and license-free, and is available free for all uses.”

Only update clients that were built with the same encryption key that was used to encrypt the server data at build time can read the server file. For this reason, it is very important not to lose the encryption key used for an update project. It is also important not to change the encryption key after your update application has been distributed to end users.

Tip: The longer your encryption key, the harder it is to break. It is recommended that your encryption key be at least 20 characters long.

Updated Client Data File

The updated client data file has the extension .ts2 on the TrueUpdate Server. It contains the latest version of your client data file, and is placed on the TrueUpdate Server in case the user does not have the most recent version of the client data on their system at run time. If the TrueUpdate Client sees that the .ts2 file is newer than its existing client data file, it will download the .ts2 file and replace the client data file with the newer version.

Since the client data file is already encrypted, the updated client data file doesn't require any additional security measures.

Updated Client Executable File

The updated client executable file has the extension .ts3 on the TrueUpdate Server. It is a compressed version of the client executable. Like the updated client data file, the updated client executable is used to update the user's TrueUpdate Client application when the user does not have the most recent version of the client on their system.

Since this file does not contain any of your product-specific information, no special protection has been applied to it.

Client-Server Communication

As you probably know by now, the TrueUpdate Client application normally needs to communicate with a TrueUpdate Server in order to update your software. The question is, what kind of communication takes place?

There is no “default” communication between your TrueUpdate Client and your TrueUpdate Servers. If you build a TrueUpdate project that has an empty client and server script, no client-server communication takes place. All client-server communication is specified through script, whether it is created manually or with the project wizard.

The most common way that your projects will interact with your TrueUpdate Servers is through the action TrueUpdate.GetServerFile. This action downloads the server file you have specified from a TrueUpdate Server and then runs the server script. All of this communication takes place through the protocol specified for the TrueUpdate Server (HTTP, HTTPS, FTP, or LAN).

The TrueUpdate Client never sends information about the client or client system to the TrueUpdate Server. It merely downloads the specified server file, decrypts it and then runs the script. It may also do a self-update if necessary.

Of course, TrueUpdate provides a wide variety of actions that you can execute to perform all sorts of client-server communication over several protocols. The following sections will look more closely at some of these available options.

Secure Protocols

TrueUpdate provides you with several secure protocols that can be used in both the design and run-time phases of development.

SFTP

TrueUpdate allows you to use the SFTP (Secure File Transfer Protocol) to upload server files at publish time. SFTP is a transfer method that uses SSH to transfer files using AES, DES, and Blowfish encryption. Unlike standard FTP, it encrypts both commands and data, preventing passwords and sensitive information from being transmitted in the clear over the network. It is functionally similar to FTP, but because it uses a different protocol, you cannot use a standard FTP client to talk to an SFTP server, nor can you connect to an FTP server with a client that supports only SFTP. TrueUpdate's implementation of SFTP uses SSH2.

HTTPS

The secure hypertext transfer protocol (HTTPS) is a communications protocol designed to transfer encrypted information between computers over the World Wide Web. HTTPS is HTTP using a Secure Socket Layer (SSL). A secure socket layer is an encryption protocol invoked on a Web server that uses HTTPS.

The TrueUpdate Client can transfer files and data from a Web site using the actions HTTP.DownloadSecure and HTTP.SubmitSecure. These actions will ensure that the connection between your client and your server is secure. Note that your Web server must be specially configured in order to support HTTPS.

Custom Client-Server Communication

There may be times when you want your update client to communicate with a Web server in a way that is not built into TrueUpdate. Some examples are:

- validating the user's serial number against a Web database before providing updates
- tracking which customers have updated the software and when they last updated in a Web-based database

- allowing the user to renew a software subscription through your update client. The client will gather payment data and then submit it to your Web server through the HTTPS protocol for payment processing
- submitting user feedback from the TrueUpdate client to a Web script that in turn emails the information to you

Fortunately, TrueUpdate comes with built-in actions to facilitate this kind of communication, namely HTTP.Submit and HTTP.SubmitSecure. These two actions allow you to send data to a script on your Web server and then receive a response from the server. These actions should work with any type of standard Web server, such as Apache or Microsoft IIS, and with any type of scripting language that allows CGI communication, such as ASP, PHP, Perl, etc.

By providing a method to interface with Web scripts, TrueUpdate gives you the flexibility to add further security features to the client-server communication process beyond the built-in actions it comes with. This ability to utilize any Web scripts you develop greatly adds to the extensibility of TrueUpdate.

Important Considerations

When planning product security, it is important to keep the following in mind: even the best security measures can be circumvented. This is clearly illustrated by successful breaches of highly sophisticated security systems such as DVD encryption and secure government communications.

In addition to securing your installer, update application, and perhaps your installed application, you may want to consider controlling the distribution of your installation files. After all, a software thief can't steal something he or she can't find.

One final point to keep in mind is that *too* much security can be a bad thing. While it is important to secure your application to prevent against unauthorized use, it is equally important to allow legitimate users a painless update process.



Chapter 10:

Building and Distributing

Once you have designed your update system, you need to build and distribute it. Fortunately, TrueUpdate makes this as seamless as the actual design process.

From automatically uploading to your chosen TrueUpdate Server locations, to a handy list describing what you need to do after the build, TrueUpdate makes building your project fast and easy.

In This Chapter

In this chapter, you'll learn about:

- The build process
- The publish wizard
- Build Settings
- Uploading to your server
- Integrating the client into your software
- Distributing the client
- Launching your client
- Testing
- Log files

The Build Process

Once you have finished creating your TrueUpdate project, you must use it to generate the actual TrueUpdate Client application and server configuration files. This is similar to a programmer compiling source code into a working executable.

When building your update, TrueUpdate will go through the following steps in the order listed below:

1. Run any pre-build programs that have been specified.
2. Collect all of the fonts needed for the screens in your update.
3. Convert all of the images used by the screens.
4. Collect all of the required language files.
5. Collect all includes (external script files) that have been specified.
6. Collect all action plugins that have been specified.
7. Collect any external files that have been specified (e.g. a custom icon).
8. Copy the run-time engine into the specified output directory.
9. Create the client data file by compressing all of the collected files and saving all necessary client information.
10. Create the server configuration files (*.ts1, *.ts2, and *.ts3).
11. Upload the server configuration files and the server information file (*.tsx) to any automatic upload locations you've defined in the project.
12. Run any post-build programs that have been specified.

Note: When TrueUpdate uploads the files to your automatic upload locations, it only uploads the files that it needs to (the ones that have changed since the last time your project was built). For instance, the .ts2 file will only be uploaded if it differs from the .ts2 file that is already on the server, and the .ts3 file will only be uploaded if its file version is newer than the file version of the .ts3 file on the server.

The Publish Wizard

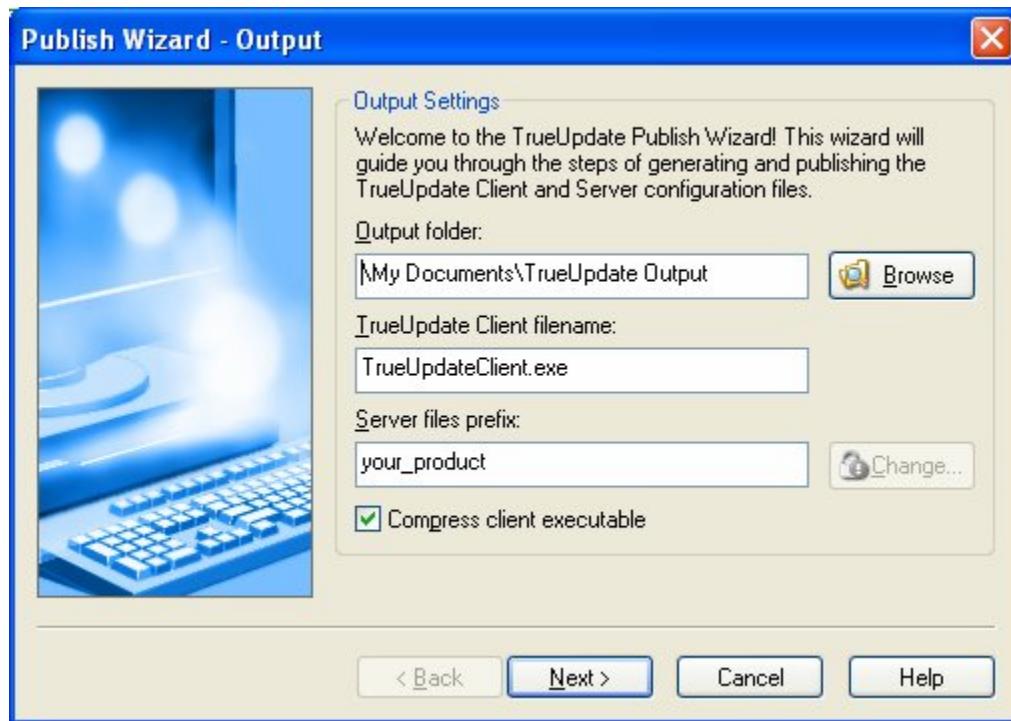
With the goal of making the build process as seamless as possible, TrueUpdate includes a publish wizard to assist you. The publish wizard can be accessed by choosing Build from the Publish menu.

The first step in the publish wizard allows you to specify three important items: the output folder, the name of the TrueUpdate Client executable, and the server files prefix.

The output folder is simply the location where your TrueUpdate Client and server configuration files will be generated on your local system.

The TrueUpdate Client filename determines the name of the client executable that you will distribute to your users.

The server files prefix is the name that will be used for the server configuration files, i.e. the name that will be used for the .ts1, .ts2, and .ts3 files.



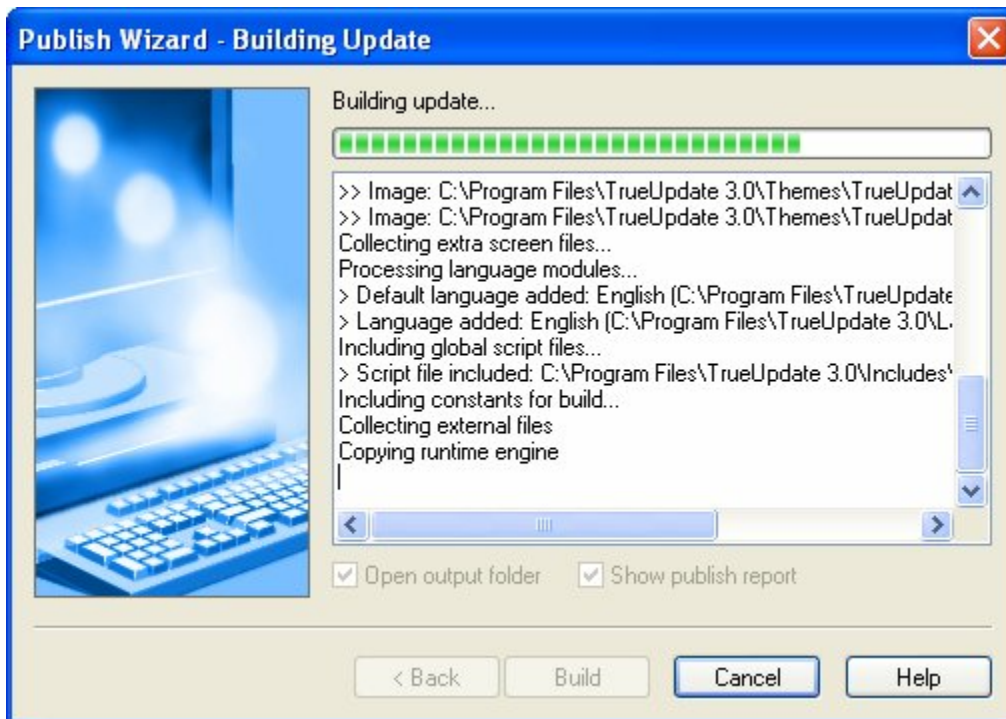
Note: The first time you build your project, you will be able to edit the server files prefix directly. However, on all subsequent builds the server files prefix field will be disabled; to edit the server files prefix, you will need to click the Change button. Clicking the Change button displays a warning before allowing you to change the value, because changing the server files prefix will “break” an existing distribution. The TrueUpdate Client needs to know what the server files prefix is in order to download the server files. If you change the server files prefix, you will need to redistribute the TrueUpdate Client application to all of your users.

The second step in the publish wizard prompts you to configure your upload locations. You can use this feature to automatically upload the server configuration files to your TrueUpdate Server locations. You can also use it to upload the files to other locations, such as a backup file server or a folder on your network for testing.

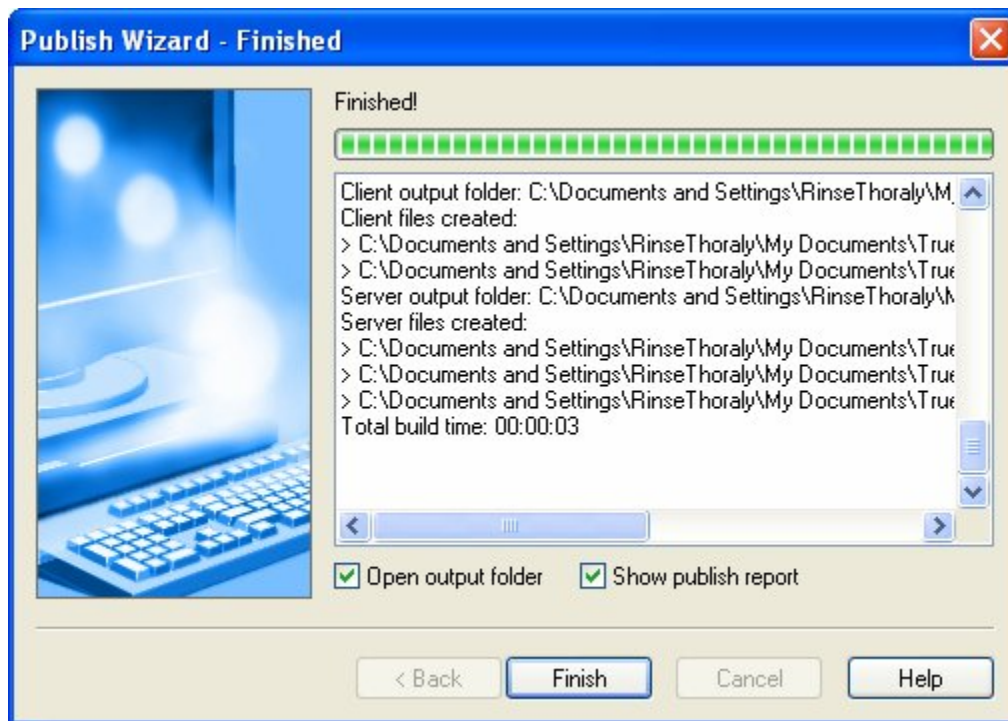


Note that you can add and edit upload locations in this step, and you can enable or disable the individual upload locations by checking or unchecking the corresponding checkbox.

Once you've selected the locations that you want to upload the server files to, you can click the Build button to start the build process.



Once the build process is complete, the publish wizard will display a summary of the build process and any errors or warnings that occurred during the build.



There are two options below the summary that control what happens after you click Finish. One determines whether TrueUpdate will automatically open the output folder for you (for example, if you wanted to immediately test the update, you won't have to navigate to the output folder yourself). The other controls whether to display a publish report listing the files that have been output by the build process and what you need to do next.

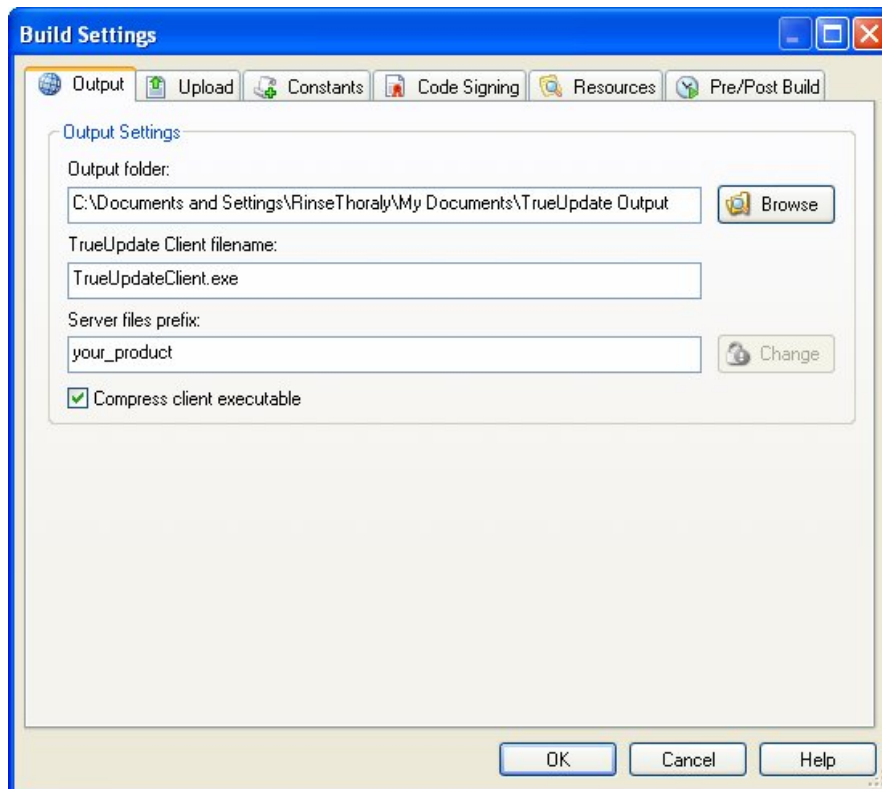
When you're finished reading the summary, you can click Finish to exit the publish wizard.

Build Settings

The Build Settings dialog allows you to specify default settings for the publish wizard and to configure additional features that affect the build process or the generated files. You can access the Build Settings dialog by choosing Publish > Settings from the program menu.

Output

The Output tab is where you can set the output folder, the TrueUpdate Client filename, and the server files prefix.



The output folder is where all of your update files will be built to. This can be any location on a local hard drive or on the local network. All of the files that TrueUpdate uses are built to this location, including both client files and server configuration files.

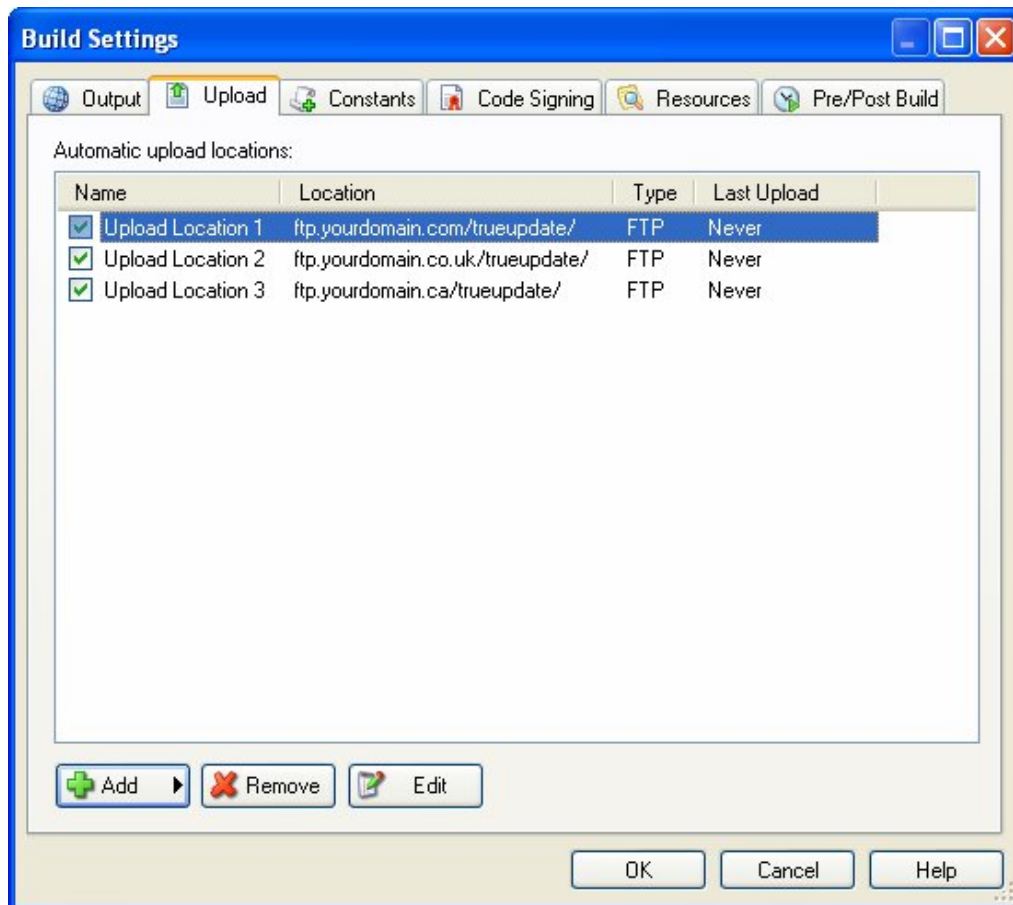
There are two client files that get built: an executable file (the “client executable”) and a data file (the “client data file”). The name that you enter in the TrueUpdate Client filename field is the full name that will be used for the executable file.

TrueUpdate automatically chooses a name for the client data file based on the executable file’s name. For example, if you were to specify “Update.exe” in the TrueUpdate Client filename field, the data file would be named “Update.dat.” If you were to specify “myupdate.exe,” the data file would be named “myupdate.dat.”

Upload

A great feature in TrueUpdate is the ability to have your server configuration files automatically uploaded for you. This can save you a lot of time when distributing your updates, especially if you have more than one server to upload to.

You can configure as many or as few upload locations as you want, but in general each upload location will correspond to a TrueUpdate Server used in your project.



Note: When specifying upload locations it is not necessary to specify any file names. You only need to specify the folder where the files will be uploaded to.

Upload locations come in three different varieties that correspond to the upload methods used: FTP, SFTP, and File Copy.

FTP

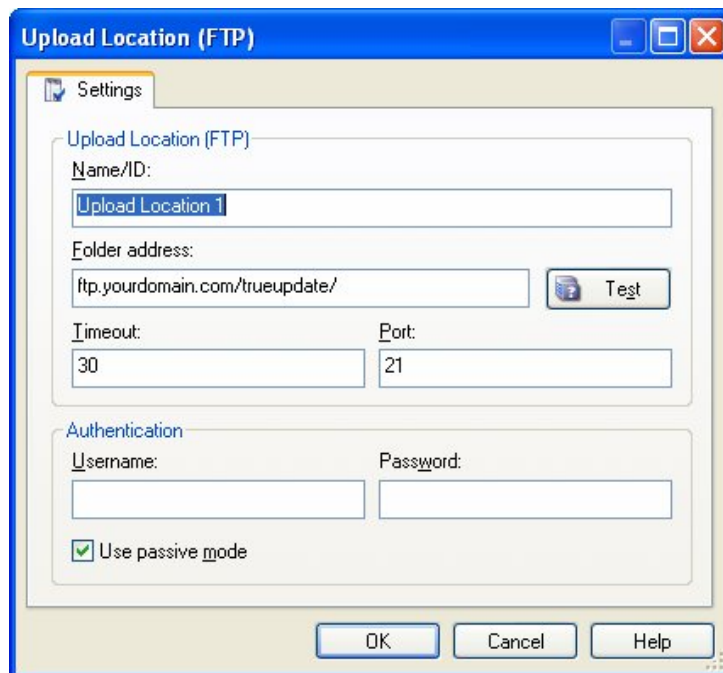
Uses the File Transfer Protocol to upload your server files to a specific folder on an FTP server.

SFTP

Uses the Secure File Transfer Protocol to upload your server files to an FTP server. The SFTP upload method is very similar to the FTP upload method except that it encrypts the transfer of data between the client and server.

File Copy

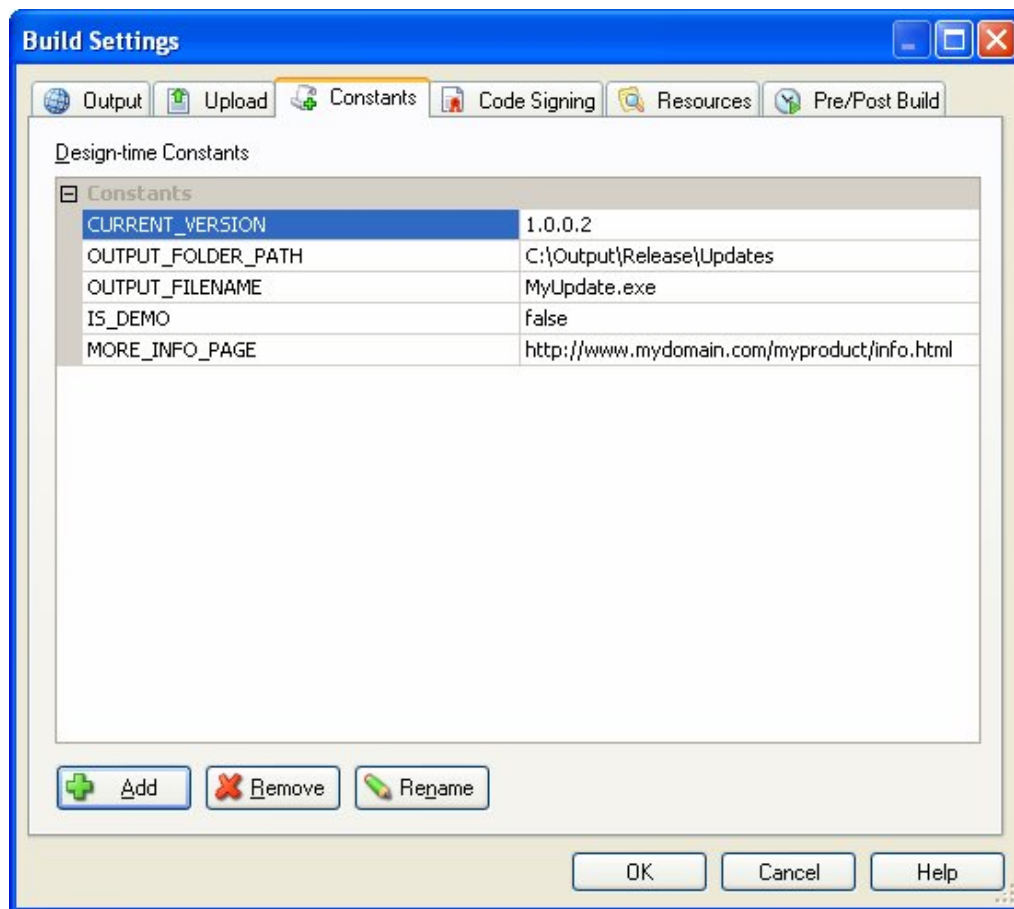
Copies the server files to a writable folder on a local storage device (e.g. hard drive, floppy drive, USB key) or a LAN location (i.e. a folder on the local area network). Normally used with LAN/Local TrueUpdate Servers or internal Web/FTP servers.



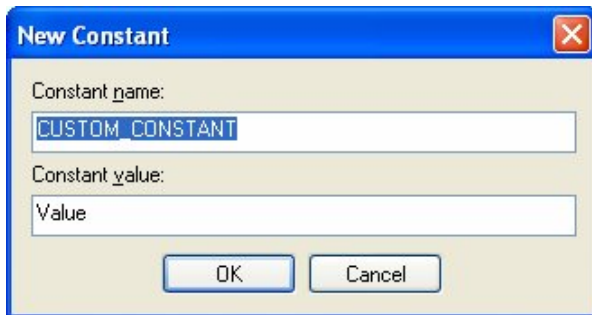
Constants

Constants are essentially design time variables. They are similar to session variables, but instead of being expanded as the user runs the update, they are expanded when you build the project. In other words, when you build your project, all of the constants are automatically replaced by the values assigned to them.

You can define constants on the Constants tab of the Build Settings dialog, which you can access by choosing Publish > Settings.



Clicking the Add button on the Constants tab displays a dialog where you can name the new constant and give it a value.



Each constant has a name that will be replaced by this value throughout the project when the project is built. It's exactly like a big search-and-replace operation that happens whenever you build the project.

Since each constant is essentially just a name that gets replaced with different text, you can use them just about anywhere. You can use them on screens, in file paths, in actions...pretty much anywhere that you can enter text.

Note: Design-time constant names can be in any format you like. One recognizable format which we recommend is to write the constant name in all capitals, using underscores in place of spaces between words, like so: `LATEST_FILE_VERSION`.

Unattended Builds

An unattended build is when a project (in this case a TrueUpdate project) is built with little or no developer interaction, often by another software process. Unattended builds are usually used when many different tasks need to be accomplished in sequence with the TrueUpdate build process. In these cases a software application or a simple batch file is used to automate the tasks. An example would be using a program to compile the source code for an application, building that application's installer using Setup Factory, and then building the update using TrueUpdate.

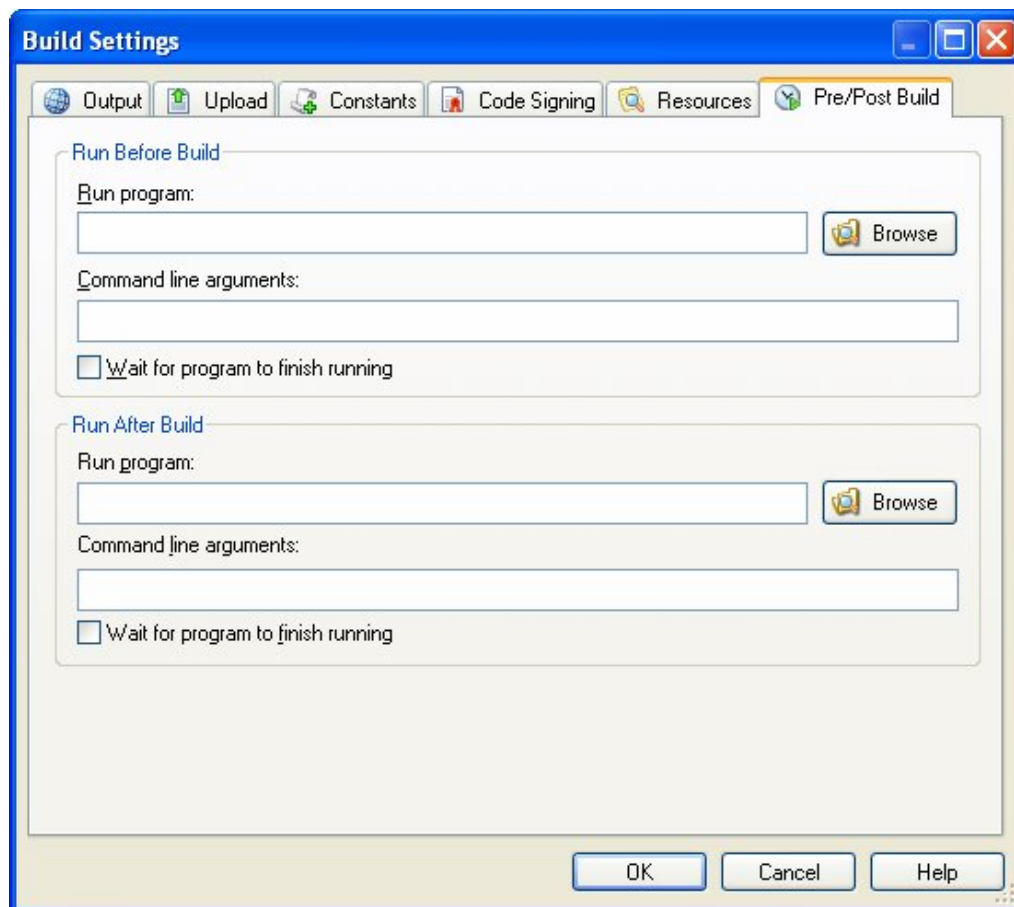
Constants are extremely useful for performing unattended builds because they can be changed using a command line argument. You can even use constants to specify the output location, and the server file prefix. The possibilities are endless.

For example, you could create a batch file that would generate a unique update for every one of your customers, with the customer's name showing up on one or all of the screens during the update.

For more information on performing unattended builds of your project, search for "Unattended Build Options" in the TrueUpdate help file.

Pre/Post Build Steps

There may be instances where you want to run a program either before or after your project has been built. TrueUpdate makes this process easy. On the Pre/Post Build tab of the Build Settings dialog, you can set a program to Run Before Build, and a program to Run After Build.



The uses of this feature are limited only by your imagination. You could, for example, chain several builds together by having one project call another through the Run After Build option. Or you could run an “automated backup” batch file that compresses all of the server files into a single zip file, and then copies the zip file to a folder on your network. The possibilities are limitless.

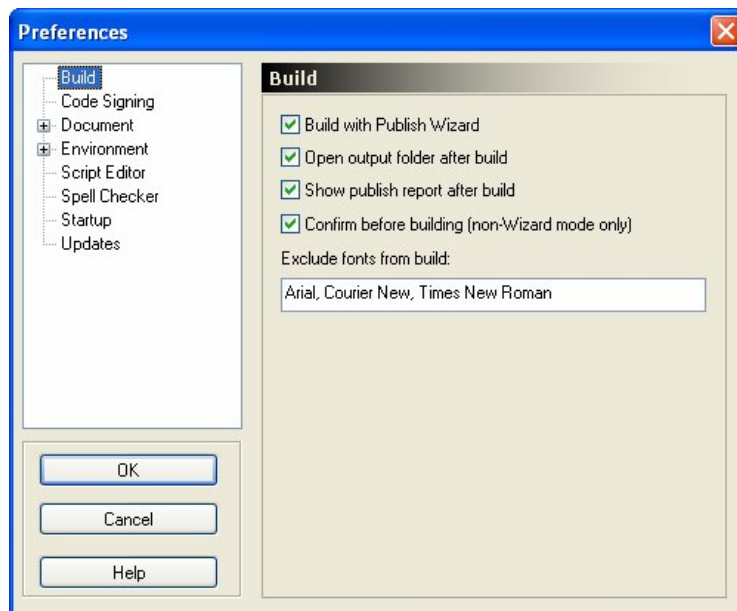
You can specify the path to the desired program in either of the Run Program fields. Any command line arguments that are needed should be specified in the Command Line arguments fields. If you want the design environment to pause while the other program is open, check the Wait for program to finish running checkbox.

Build Preferences

TrueUpdate includes a number of preferences that affect how TrueUpdate handles the build process. You can find these preferences by choosing Preferences from the Edit menu and selecting the Build category.

The build preferences allow you to control whether TrueUpdate opens the output folder, whether it shows the publish report by default, whether it displays the publish wizard before building, and (when not using the wizard) whether a confirmation dialog is displayed before the build process begins.

You can also specify a list of fonts to exclude from the build process. Fonts take up additional space in your update so it is often a good idea to exclude any fonts that you know your end user will already have. By default this field includes *Arial*, *Courier New*, and *Times New Roman*.



Integrating the Client into your Software

Once you've built the TrueUpdate Client, you need to integrate it into your software. Integrating the client simply means including it with your software distribution and providing one or more ways for your users (or your software) to initiate an update.

Note: The TrueUpdate Client can be thought of as the client executable and the client data file.

Integrating TrueUpdate can be as simple as putting a shortcut on the Start menu that will launch the client executable, or it can be as sophisticated as having your main application run the client executable every 15 days. You could provide an item in your application's menu to initiate an update, or even a button on your application's toolbar. The level of integration is up to you.

Note: The TrueUpdate Client program is an executable that can be run normally from Windows.

Integrating TrueUpdate into your application involves two basic steps: adding the client files to your software distribution, and providing one or more ways to initiate an update.

Step 1: Adding the Client Files

The TrueUpdate client consists of two files: a client executable, and a single data file. By default the client executable is named TrueUpdateClient.exe and the data file is named TrueUpdateClient.dat. Both files are generated when you build your update.

Note: A complete list of files that you need to distribute can be found in the publish report that is generated each time you build your update.

Exactly how you add the client files to your distribution depends on the distribution method you use. It can be as simple as including the two files in a zip archive, or as sophisticated as using a professional installation tool like Setup Factory.

You can install the files anywhere on the user's system, but it's usually best to install them in the same place as your application. This way the location of your software can easily be determined at run time by using the built-in variable `_SourceFolder`, or the built-in session variable `%SourceFolder%`.

If you'd prefer to have the client files in a different folder, remember to provide some way for the TrueUpdate client to determine the path where your software was

installed. For example, you could write the path to a Registry key or INI file with your installer and then use actions to read that information into a variable.

Step 2: Triggering TrueUpdate

Initiating an update is simply a matter of running the TrueUpdate client. In fact, your users could check for an update simply by double-clicking on the client executable.

Of course, there are several more sophisticated ways to initiate an update. Here are some of the different ways TrueUpdate could be started:

Using a Start Menu item

Install a shortcut to the TrueUpdate client in the user's Start menu.

Using an icon on the Desktop

Give your users the option to install a shortcut on their Desktop so they can initiate an update at any time. You could use the standard TrueUpdate client icon for this purpose, or perhaps the same icon you use for your application.

Using a menu item in your program

Provide an item in a program menu like Help > Check for Updates that will launch the client executable.

Using a toolbar button in your program

Provide a "Check for Updates" button on one of your program's toolbars.

Automatically on starting Windows

Install a shortcut in the user's Startup folder to automatically run the TrueUpdate Client every time the user's system is started. You could also create a small "launcher" program that checks the number of times the system has been booted since the last update was performed and only launch TrueUpdate periodically. You could even set up an event with the Windows scheduling service if it's active on the user's system.

Automatically when your program starts

Launch TrueUpdate silently from your program when it's started. You could check the system date and only launch the client if your application was last

started more than a certain number of days ago. Or keep a counter of “application starts” in the Registry, and only check for an update after a certain amount of times that the user launches your program. You can even let your user specify how often they want the checks to be performed.

As a stand-alone executable, the TrueUpdate client gives you plenty of freedom to initiate the update process in a way that fits the style of your application. Which method (or methods) you choose to provide is entirely up to you.

Source Code

Here are some source code samples using ShellExecute to call the TrueUpdate Client from Visual Basic, C/C++, and C#.

Calling ShellExecute from Visual Basic

```
Private Declare Function ShellExecute Lib "shell32.dll" _
    Alias "ShellExecuteA" (ByVal hwnd As Long, _
    ByVal lpOperation As String, ByVal lpFile As String, _
    ByVal lpParameters As String, ByVal lpDirectory As String, _
    ByVal nShowCmd As Long) As Long

Private Const SW_NORMAL = 1

Private Sub RunTrueUpdateClient()

    ' Execute the program
    Dim lReturn As Long

    lReturn = ShellExecute(Me.hwnd, "open", _
        "C:\\Program Files\\Widget Designer\\TrueUpdateClient.exe", _
        "", 0, SW_NORMAL)

    If lReturn <= 32 Then
        MsgBox "Error executing client"
    End If

End Sub
```

Calling ShellExecute from C/C++

```
BOOL RunTrueUpdateClient()
{
    HWND hWindow;
    int nResult;

    hWindow = GetSafeHwnd();
    nResult = ShellExecute(hWindow, "open"
        , "C:\\Program Files\\Widget Designer\\TrueUpdateClient.exe"
        , "", NULL, SW_NORMAL);

    if(nResult > 32) // values above 32 indicate success
    {
        return TRUE; // update executed
    }
    else
    {
        return FALSE; // update not executed
    }
}
```

Using Process components and the .Net framework from C#

```
using System;
using System.Diagnostics;
using System.ComponentModel;

public bool RunTrueUpdateClient()
{
    bool bReturn = true;
    try
    {
        Process UpdateProcess = new Process();
        UpdateProcess.StartInfo.FileName =
            "C:\\Program Files\\Widget Designer\\update.exe";
        UpdateProcess.StartInfo.WindowStyle =
            ProcessWindowStyle.Normal;
        //The Start method returns true on success and
        //false on failure
        bReturn = UpdateProcess.Start();
    }
}
```

```

        catch (Win32Exception e)
        {
            //An error has occurred
            bReturn = false;
        }

        return bReturn;
    }

```

Testing Your Update

One of the most important and often overlooked steps when creating an update is testing it after it has been built. You should test your update on as many computers and operating systems as possible. Try it on every operating system that your product supports. Windows 95, Windows 98, Windows 2000, Windows ME, Windows NT, Windows XP and Windows Vista all have both slight and major differences between them and should be thoroughly tested. It is also important to test the way that your update responds to different OS configurations, different service packs, monitor resolutions, color depth, and font sizes. If your update needs a lot of hard drive space, be sure to test it on systems with a very limited amount of hard drive space.

If you have made use of TrueUpdate's multilingual support, be sure to test out each language in your update. If you are using multiple TrueUpdate servers, be sure to test each update server individually to ensure that they are all working properly.

It is very important to test before you distribute. An 'internal build' of your update is much easier to recall than is a public release should a problem arise.

Tip: For information related to tracking down script-related issues, see *Debugging Your Scripts* in Chapter 11.

Log Files

TrueUpdate's built-in logging function can quickly become a developer's best friend. It is enabled by default, and contains all the information that you as a developer will need to diagnose an update gone bad. Everything that happens during the update is logged, from which screens were displayed to what version of the update engine is being used.

Beyond its basic components, the update log file is fully customizable; as the developer, you can choose what information is logged and what information is not. If you require even more detail than what the built-in logging capabilities provide, you can add custom lines to the log at any point during your update.

Using log files is invaluable to developers for diagnosing problems. Let's face it, no matter how much testing is done, there will always be one end user who hits a snag. Utilizing log files, you can find out exactly where that snag occurred, and fix the problem. As a result, you save on support costs and keep your customer happy.

Ultimately, the choice of whether or not to use the log file, and how much information should be in that log file, is up to you.

Here is an example of what a log file might look like:

```
[12/28/2004 16:38:11] Success    Update started: C:\output\tu20\TrueUpdateClient.exe
[12/28/2004 16:38:11] Notice    Update engine version: 3.0.0.0
[12/28/2004 16:38:11] Notice    Product: Your Product, version %ProductVer%
[12/28/2004 16:38:11] Success    Language set: Primary = 9, Secondary = 1
[12/28/2004 16:38:11] Success    Include script: _TU20_Global_Functions.lua
[12/28/2004 16:38:11] Success    Language set: Primary = 9, Secondary = 1
[12/28/2004 16:38:12] Success    Run client data event: Client Script
```

Chapter 11:

Scripting Guide

One of the powerful features of TrueUpdate is its scripting engine. This chapter will introduce you to the new scripting environment and language.

TrueUpdate scripting is very simple, with only a handful of concepts to learn. Here is what it looks like:

```
a = 5;
if a < 10 then
    Dialog.Message("Guess what?", "a is less than 10");
end
```

(Note: this script is only a demonstration. Don't worry if you don't understand it yet.)

The example above assigns a value to a variable, tests the contents of that variable, and if the value turns out to be less than 10, uses a TrueUpdate action called "Dialog.Message" to display a message to the user.

New programmers and experienced coders alike will find that TrueUpdate is a powerful, flexible yet simple scripting environment to work in.

In This Chapter

In this chapter, you'll learn about:

- Important scripting concepts
- Variables
- Variable scope and variable naming
- Types and values
- Expressions and operators
- Control structures (if, while, repeat, and for)
- Tables (arrays)
- Functions
- String manipulation
- Debugging your scripts
- Syntax errors and functional errors
- Other scripting resources

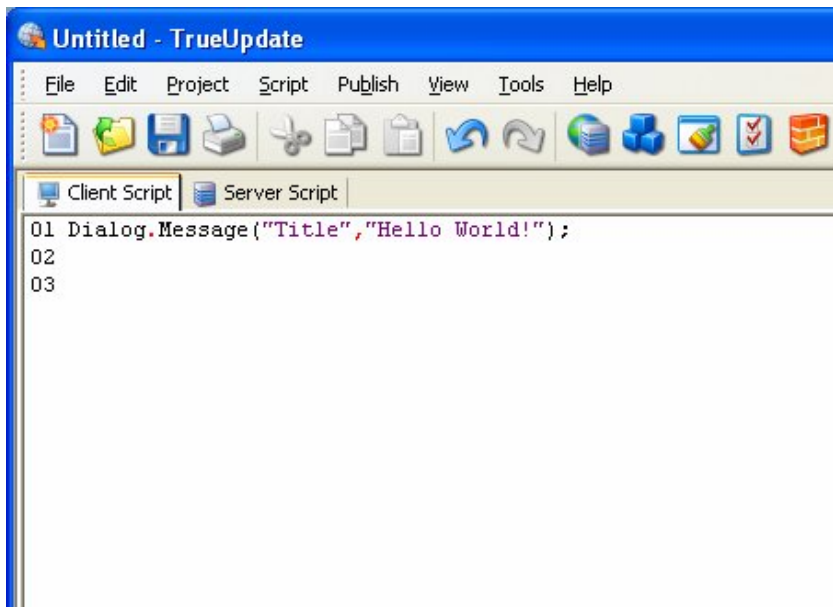
A Quick Example of Scripting in TrueUpdate

Here is a short tutorial showing you how to enter a script into TrueUpdate and preview the results:

1. Start a new project.
2. In your project's Client Script, add the following code:

```
Dialog.Message("Title", "Hello World");
```

It should look like this when you're done:



3. Choose Publish > Build from the program menu, and go through the publish wizard.
4. Once your project has finished building, run the created file (typically TrueUpdateClient.exe).

When the Client Script is run, the script you entered will be performed. You should see the following dialog appear:



Congratulations! You have just made your first script. Though this is a simple example, it shows you just how easy it is to make something happen in your TrueUpdate application. You can use the above method to try out any script you want in TrueUpdate.

Note: If you are working with actions that interact with screens, you must perform these actions from a screen event.

Important Scripting Concepts

There are a few important things that you should know about the TrueUpdate scripting language in general before we go on.

Script is Global

The scripting engine is global to the runtime environment. That means that all of your events will “know” about other variables and functions declared elsewhere in the product. For example, if you assign “myvar = 10;” on a particular screen event, myvar will still equal 10 when the next event is triggered. There are ways around this global nature (see *Variable Scope* on page 204), but it is generally true of the scripting engine.

Script is Case-Sensitive

The scripting engine is case-sensitive. This means that upper and lower case characters are important for things like keywords, variable names and function names.

For example:

```
ABC = 10;  
aBC = 7;
```

In the above script, ABC and aBC refer to two different variables, and can hold different values. The lowercase “a” in “aBC” makes it completely different from “ABC” as far as TrueUpdate is concerned.

The same principle applies to function names as well. For example:

```
Dialog.Message("Hi", "Hello World");
```

...refers to a built-in TrueUpdate function. However,

```
DIALOG.Message("Hi", "Hello World");
```

...will not be recognized as the built-in function, because DIALOG and Dialog are seen as two completely different names.

Note: It’s entirely possible to have two functions with the same spelling but different capitalization—for example, GreetUser and gREeTUSeR would be seen as two totally different functions. Although it’s definitely possible for such functions to coexist, it’s generally better to give functions completely different names to avoid any confusion.

Comments

You can insert non-executable comments into your scripts to explain and document your code. In a script, any text after two dashes (--) on a line will be ignored. For example:

```
-- Assign 10 to variable abc  
abc = 10;
```

...or:

```
abc = 10; -- Assign 10 to abc
```

Both of the above examples do the exact same thing—the comments do not affect the script in any way.

You can also create multi-line comments by using `--[[` and `]]` on either side of the comment:

```
--[[ This is  
a multi-line  
comment ]]  
a = 10;
```

You should use comments to explain your scripts as much as possible in order to make them easier to understand by yourself and others.

Delimiting Statements

Each unique statement can either be on its own line and/or separated by a semi-colon (;). For example, all of the following scripts are valid:

Script 1:

```
a = 10  
MyVar = a
```

Script 2:

```
a = 10; MyVar = a;
```

Script 3:

```
a = 10;  
MyVar = a;
```

However, we recommend that you end all statements with a semi-colon (as in scripts 2 and 3 above).

Variables

What Are Variables?

Variables are very important to scripting in TrueUpdate. Variables are simply “nicknames” or “placeholders” for values that might need to be modified or re-used in the future. For example, the following script assigns the value 10 to a variable called “amount.”

```
amount = 10;
```

Note: We say that values are “assigned to” or “stored in” variables. If you picture a variable as a container that can hold a value, assigning a value to a variable is like “placing” that value into a container. You can change this value at any time by assigning a different value to the variable; the new value simply replaces the old one. This ability to hold changeable information is what makes variables so useful.

Here are a couple of examples demonstrating how you can operate on the “amount” variable:

```
amount = 10;
amount = amount + 20;
Dialog.Message("Value", amount);
```

This stores 10 in the variable named amount, then adds 20 to that value, and then finally makes a message box appear with the current value (which is now the number 30) in it.

You can also assign one variable to another:

```
a = 10;
b = a;
Dialog.Message("Value", b);
```

This will make a message box appear with the number 10 in it. The line “b = a;” assigns the value of “a” (which is 10) to “b.”

Variable Scope

All variables in TrueUpdate are *global* by default. This means that they exist project-wide, and hold their values from one script to the next. In other words, if a value is

assigned to a variable in one script, the variable will still hold that value when the next script is executed.

For example, if you enter the script:

```
foo = 10;
```

...into a screen's On Preload event, and then enter:

```
Dialog.Message("The value is:", foo);
```

...into a different screen's On Preload event, the second script will use the value that was assigned to "foo" on the first event. As a result, when the second screen is encountered, a message box will appear with the number 10 in it.

Note that the order of execution is important...in order for one script to be able to use the value that was assigned to the variable in another script, that other script has to be executed first. In the above example, the first screen's On Preload event is triggered *before* the second screen's On Preload event, so the value 10 is already assigned to foo when the second screen's event script is executed.

Local Variables

The global nature of the scripting engine means that a variable will retain its value throughout your entire project. You can, however, make variables that are non-global, by using the special keyword "local." Putting the word "local" in front of a variable assignment creates a variable that is local to the current script, function, or block of code.

For example, let's say you have the following three scripts in the same project:

Script 1:

```
-- assign 10 to x  
x = 10;
```

Script 2:

```
local x = 500;  
Dialog.Message("Local value of x is:", x);  
x = 250; -- this changes the local x, not the global one  
Dialog.Message("Local value of x is:", x);
```

Script 3:

```
-- display the global value of x
Dialog.Message("Global value of x is:", x);
```

Let's assume these three scripts are performed one after the other. The first script gives `x` the value 10. Since all variables are global by default, `x` will have this value inside all other scripts, too. The second script makes a *local* assignment to `x`, giving it the value of 500—but only inside that script. If anything else inside that script wants to access the value of `x`, it will see the local value instead of the global one. It's like the “`x`” variable has been temporarily replaced by another variable that looks just like it, but has a different value.

(This reminds me of those caper movies, where the bank robbers put a picture in front of the security cameras so the guards won't see that the vault is being emptied. Only in this case, it's like the bank robbers create a whole new working vault, just like the original, and then dismantle it when they leave.)

When told to display the contents of `x`, the first `Dialog.Message` action inside script #2 will display 500, since that is the local value of `x` when the action is performed. The next line assigns 250 to the local value of `x`—note that once you make a local variable, it completely replaces the global variable for the rest of the script.

Finally, the third script displays the global value of `x`, which is still 10.

Variable Naming

Variable names can be made up of any combination of letters, digits and underscores as long as they do not begin with a number and do not conflict with reserved keywords.

Examples of **valid** variables names:

```
a
strName
_My_Variable
data1
data_1_23
index
bReset
nCount
```

Examples of **invalid** variable names:

```
1
1data
%MyValue%
$strData
for
local
_FirstName+LastName_
User Name
```

Reserved Keywords

The following words are reserved and cannot be used for variable or function names:

and	break	do	else	elseif
end	false	for	function	if
in	local	nil	not	or
repeat	return	table	then	true
until	while			

Types and Values

TrueUpdate's scripting language is dynamically typed. There are no type definitions—instead, each value carries its own type.

What this means is that you don't have to declare a variable to be of a certain type before using it. For example, in C++, if you want to use a number, you have to first declare the variable's type and then assign a value to it:

```
int j;
j = 10;
```

The above C++ example declares `j` as an integer, and then assigns 10 to it.

As we have seen, in TrueUpdate you can just assign a value to a variable without declaring its type. Variables don't really have types; instead, it's the values inside them that are considered to be one type or another. For example:

```
j = 10;
```

...this automatically creates the variable named “j” and assigns the value 10 to it. Although this value has a type (it's a *number*), the variable itself is still typeless. This means that you can turn around and assign a different type of value to j, like so:

```
j = "Hello";
```

This replaces the number 10 that is stored in j with the string “Hello.” The fact that a string is a different type of value doesn't matter; the variable j doesn't care what kind of value it holds, it just stores whatever you put in it.

There are six basic data types in TrueUpdate: number, string, nil, Boolean, function, and table. The sections below will explain each data type in more detail.

Number

A number is exactly that: a numeric value. The number type represents real numbers—specifically, double-precision floating-point values. There is no distinction between integers and floating-point numbers (also known as “fractions”)...all of them are just “numbers.” Here are some examples of valid numbers:

4 4. .4 0.4 4.57e-3 0.3e12

String

A string is simply a sequence of characters. For example, “Joe2” is a string of four characters, starting with a capital “J” and ending with the number “2.” Strings can vary widely in length; a string can contain a single letter, or a single word, or the contents of an entire book.

Strings may contain spaces and even more exotic characters, such as carriage returns and line feeds. In fact, strings may contain any combination of valid 8-bit ASCII characters, including null characters (“\0”). TrueUpdate automatically manages string memory, so you never have to worry about allocating or de-allocating memory for strings.

Strings can be used quite intuitively and naturally. They should be delimited by matching single quotes or double quotes. Here are some examples that use strings:


```
Name = "Joe Blow";
Dialog.Message("Title", "Hello, how are you?");
LastName = 'Blow';
```

Normally double quotes are used for strings, but single quotes can be useful if you have a string that contains double quotes. Whichever type of quotes you use, you can include the other kind inside the string without escaping it. For example:

```
doubles = "How's that again?";
singles = 'She said "Talk to the hand," and I was all like "Dude!"';
```

If we used double quotes for the second line, it would look like this:

```
escaped = "She said \"Talk to the hand,\" and I was all like \"Dude!\"";
```

Normally, the scripting engine sees double quotes as marking the beginning or end of a string. In order to include double quotes inside a double-quoted string, you need to *escape* them with backslashes. This tells the scripting engine that you want to include an actual quote character *in* the string.

The backslash and quote (\") is known as an *escape sequence*. An escape sequence is a special sequence of characters that gets converted or “translated” into something else by the script engine. Escape sequences allow you to include things that can’t be typed directly into a string.

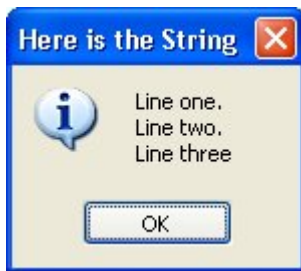
The escape sequences that you can use include:

- \a - bell
- \b - backspace
- \f - form feed
- \n - newline
- \r - carriage return
- \t - horizontal tab
- \v - vertical tab
- \\ - backslash
- \\" - quotation mark
- \' - apostrophe
- \[- left square bracket
- \] - right square bracket

So, for example, if you want to represent three lines of text in a single string, you would use the following:

```
Lines = "Line one.\nLine two.\nLine three";  
Dialog.Message("Here is the String", Lines);
```

This assigns a string to a variable named `Lines`, and uses the newline escape sequence to start a new line after each sentence. The `Dialog.Message` function displays the contents of the `Lines` variable in a message box, like this:



Another common example is when you want to represent a path to a file such as `C:\My Folder\My Data.txt`. You just need to remember to escape the backslashes:

```
MyPath = "C:\\My Folder\\My Data.txt";
```

Each double-backslash represents a single backslash when used inside a string.

If you know your ASCII table, you can use a backslash character followed by a number with up to three digits to represent any character by its ASCII value. For example, the ASCII value for a newline character is 10, so the following two lines do the exact same thing:

```
Lines = "Line one.\nLine two.\nLine three";  
Lines = "Line one.\10Line two.\10Line three";
```

However, you will not need to use this format very often, if ever.

You can also define strings on multiple lines by using double square brackets (`[[` and `]]`). A string between double square brackets does not need any escape characters. The double square brackets let you type special characters like backslashes, quotes and newlines right into the string.

For example:

```
Lines = [[Line one.  
Line two.  
Line three.]];
```

is equivalent to:

```
Lines = "Line one.\nLine two.\nLine three";
```

This can be useful if you have preformatted text that you want to use as a string, and you don't want to have to convert all of the special characters into escape sequences.

The last important thing to know about strings is that the script engine provides automatic conversion between numbers and strings at run time. Whenever a numeric operation is applied to a string, the engine tries to convert the string to a number for the operation. Of course, this will only be successful if the string contains something that can be interpreted as a number.

For example, the following lines are both valid:

```
a = "10" + 1; -- Result is 11  
b = "33" * 2; -- Result is 66
```

However, the following lines would not give you the same conversion result:

```
a = "10+1"; -- Result is the string "10+1"  
b = "hello" + 1; -- ERROR, can't convert "hello" to a number
```

For more information on working with strings, see page 236.

Nil

Nil is a special value type. It basically represents the absence of any other kind of value.

You can assign nil to a variable, just like any other value. Note that this isn't the same as assigning the letters "nil" to a variable, as in a string. Like other keywords, nil must be left unquoted in order to be recognized. It should also be entered in all lowercase letters.

Nil will always evaluate to false when used in a condition:

```
a = nil;
if a then
  -- Any lines in here
  -- will not be executed
end
```

It can also be used to “delete” a variable:

```
y = "Joe Blow";
y = nil;
```

In the example above, “y” will no longer contain a value after the second line.

Boolean

Boolean variable types can have one of two values: true, or false. They can be used in conditions and to perform Boolean logic operations. For example:

```
boolybooly = true;
if boolybooly then
  -- Any script in here will be executed
end
```

This sets a variable named boolybooly to true, and then uses it in an if statement. Similarly:

```
a = true;
b = false;
if (a and b) then
  -- Any script here will not be executed because
  -- true and false is false.
end
```

This time, the if statement needs both “a” and “b” to be true in order for the lines inside it to be executed. In this case, that won’t happen because “b” has been set to false.

Function

The script engine allows you to define your own functions (or “sub-routines”), which are essentially small pieces of script that can be executed on demand. Each function has a name that is used to identify the function. You can actually use that function name as a special kind of value, in order to store a “reference” to that function in a variable, or to pass it to another function. This kind of reference is of the *function* type.

For more information on functions, see page 231.

Table

Tables are a very powerful way to store lists of indexed values under one name. Tables are actually associative arrays—that is, they are arrays that can be indexed not only with numbers, but with any kind of value (including strings).

Here are a few quick examples (we cover tables in more detail on page 223):

Example 1:

```
guys = {"Adam", "Brett", "Darryl"};
Dialog.Message("Second Name in the List", guys[2]);
```

This will display a message box with the word “Brett” in it.

Example 2:

```
t = {};
t.FirstName = "Michael";
t.LastName = "Jackson";
t.Occupation = "Singer";
Dialog.Message(t.FirstName, t.Occupation);
```

This will display the following message box:



You can assign tables to other variables as well. For example:

```
table_one = {};  
table_one.FirstName = "Michael";  
table_one.LastName = "Jackson";  
table_one.Occupation = "Singer";  
table_two = table_one;  
occupation = table_two.Occupation;  
Dialog.Message(b.FirstName, occupation);
```

Tables can be indexed using array notation (`my_table[1]`), or by dot notation if not indexed by numbers (`my_table.LastName`).

Note that when you assign one table to another, as in the following line:

```
table_two = table_one;
```

...this doesn't actually copy `table_two` into `table_one`. Instead, `table_two` and `table_one` both refer to the *same* table.

This is because the name of a table actually refers to an address in memory where the data within the table is stored. So when you assign the contents of the variable `table_one` to the variable `table_two`, you're copying the *address*, and not the actual data. You're essentially making the two variables "point" to the same table of data.

In order to copy the contents of a table, you need to create a new table and then copy all of the data over one item at a time.

For more information on copying tables, see page 228.

Variable Assignment

Variables can have new values assigned to them by using the assignment operator (`=`). This includes copying the value of one variable into another. For example:

```
a = 10;  
b = "I am happy";  
c = b;
```

It is interesting to note that the script engine supports multiple assignment:

```
a, b = 1, 2;
```

After the script above, the variable "a" contains the number 1 and the variable "b" contains the number 2.

Tables and functions are a bit of a special case: when you use the assignment operator on a table or function, you create an alias that points to the same table or function as the variable being “copied.” Programmers call this copying *by reference* as opposed to copying *by value*.

Expressions and Operators

An expression is anything that evaluates to a value. This can include a single value such as “6” or a compound value built with operators such as “1 + 3”. You can use parentheses to “group” expressions and control the order in which they are evaluated. For example, the following lines will all evaluate to the same value:

```
a = 10;  
a = (5 * 1) * 2;  
a = 100 / 10;  
a = 100 / (2 * 5);
```

Arithmetic Operators

Arithmetic operators are used to perform mathematical operations on numbers. The following mathematical operators are supported:

+	(addition)
-	(subtraction)
*	(multiplication)
/	(division)
unary -	(negation)

Here are some examples:

```
a = 5 + 2;  
b = a * 100;  
twentythreepercent = 23 / 100;  
neg = -29;  
pos = -neg;
```

Relational Operators

Relational operators allow you to compare how one value relates to another. The following relational operators are supported:

>	(greater-than)
<	(less-than)
<=	(less-than or equal to)
>=	(greater than or equal to)
~=	(not equal to)
==	(equal)

All of the relational operators can be applied to any two numbers or any two strings. All other values can only use the == operator to see if they are equal.

Relational operators return Boolean values (true or false). For example:

```
10 > 20; -- resolves to false
```

```
a = 10;  
a > 300; -- false
```

```
(3 * 200) > 500; -- true  
"Brett" ~= "Lorne" -- true
```

One important point to mention is that the == and ~= operators test for *complete equality*, which means that any string comparisons done with those operators are case sensitive. For example:

```
"Jojoba" == "Jojoba"; -- true  
"Wildcat" == "wildcat"; -- false  
"I like it a lot" == "I like it a LOT"; -- false  
"happy" ~= "HaPPy"; -- true
```


Logical Operators

Logical operators are used to perform Boolean operations on Boolean values. The following logical operators are supported:

and	(only true if both values are true)
or	(true if either value is true)
not	(returns the opposite of the value)

For example:

```
a = true;
b = false;
c = a and b; -- false
d = a and nil; -- false
e = not b; -- true
```

Note that only nil and false are considered to be false, and all other values are true.

For example:

```
iaminvisible = nil;
if iaminvisible then
    -- any lines in here won't happen
    -- because iaminvisible is considered false
    Dialog.Message("You can't see me!", "I am invisible!!!!");
end

if "Brett" then
    -- any lines in here WILL happen, because only nil and false
    -- are considered false...anything else, including strings,
    -- is considered true
    Dialog.Message("What about strings?", "Strings are true.");
end
```

Concatenation

In TrueUpdate scripting, the concatenation operator is two periods (..). It is used to combine two or more strings together. You don't have to put spaces before and after the periods, but you can if you want to.

For example:

```
name = "Joe" .. " Blow"; -- assigns "Joe Blow" to name
b = name .. " is number " .. 1; -- assigns "Joe Blow is number 1" to b
```

Operator Precedence

Operators are said to have *precedence*, which is a way of describing the rules that determine which operations in a series of expressions get performed first. A simple example would be the expression $1 + 2 * 3$. The multiply (*) operator has higher precedence than the add (+) operator, so this expression is equivalent to $1 + (2 * 3)$. In other words, the expression $2 * 3$ is performed first, and then $1 + 6$ is performed, resulting in the final value 7.

You can override the natural order of precedence by using parentheses. For instance, the expression $(1 + 2) * 3$ resolves to 9. The parentheses make the whole sub-expression “ $1 + 2$ ” the left value of the multiply (*) operator. Essentially, the sub-expression $1 + 2$ is evaluated first, and the result is then used in the expression $3 * 3$.

Operator precedence follows the following order, from lowest to highest priority:

and	or				
<	>	<=	>=	~=	==
..					
+	-				
*	/				
not	- (unary)				
^					

Operators are also said to have *associativity*, which is a way of describing which expressions are performed first when the operators have equal precedence. In the script engine, all binary operators are left associative, which means that whenever two operators have the same precedence, the operation on the left is performed first. The exception is the exponentiation operator (^), which is right-associative.

When in doubt, you can always use explicit parentheses to control precedence. For example:

```
a + 1 < b/2 + 1
```

...is the same as:

```
(a + 1) < ((b/2) + 1)
```

...and you can use parentheses to change the order of the calculations, too:

```
a + 1 < b/(2 + 1)
```

In this last example, instead of 1 being added to half of b, b is divided by 3.

Control Structures

The scripting engine supports the following control structures: if, while, repeat and for.

If

An if statement evaluates its condition and then only executes the “then” part if the condition is true. An if statement is terminated by the “end” keyword. The basic syntax is:

```
if condition then
    do something here
end
```

For example:

```
x = 50;
if x > 10 then
    Dialog.Message("result", "x is greater than 10");
end

y = 3;
if ((35 * y) < 100) then
    Dialog.Message("", "y times 35 is less than 100");
end
```

In the above script, only the first dialog message would be shown, because the second if condition isn't true...35 times 3 is 105, and 105 is not less than 100.

You can also use else and elseif to add more “branches” to the if statement:

```
x = 5;
if x > 10 then
    Dialog.Message("", "x is greater than 10");
else
    Dialog.Message("", "x is less than or equal to 10");
end
```

In the preceding example, the second dialog message would be shown, because 5 is not greater than 10.

```

x = 5;
if x == 10 then
    Dialog.Message("", "x is exactly 10");
elseif x == 11 then
    Dialog.Message("", "x is exactly 11");
elseif x == 12 then
    Dialog.Message("", "x is exactly 12");
else
    Dialog.Message("", "x is not 10, 11 or 12");
end

```

In that example, the last dialog message would be shown, because x is not equal to 10, or 11, or 12.

While

The while statement is used to execute the same “block” of script over and over until a condition is met. Like if statements, while statements are terminated with the “end” keyword. The basic syntax is:

```

while condition do
    do something here
end

```

The condition must be true in order for the actions inside the while statement (the “do something here” part above) to be performed. The while statement will continue to loop as long as this condition is true. Here's how it works:

If the condition is true, all of the actions between the “while” and the corresponding “end” will be performed. When the “end” is reached, the condition will be reevaluated, and if it's still true, the actions between the “while” and the “end” will be performed again. The actions will continue to loop like this until the condition evaluates to false.

For example:

```

a = 1;
while a < 10 do
    a = a + 1;
end

```

In the preceding example, the “a = a + 1;” line would be performed 9 times.

You can break out of a while loop at any time using the “break” keyword. For example:

```
count = 1;
while count < 100 do
    count = count + 1;
    if count == 50 then
        break;
    end
end
```

Although the while statement is willing to count from 1 to 99, the if statement would cause this loop to terminate as soon as count reached 50.

Repeat

The repeat statement is similar to the while statement, except that the condition is checked at the *end* of the structure instead of at the beginning. The basic syntax is:

```
repeat
    do something here
until condition
```

For example:

```
i = 1;
repeat
    i = i + 1;
until i > 10
```

This is similar to one of the while loops above, but this time, the loop is performed 10 times. The “i = i + 1;” part gets executed before the condition determines that i is now larger than 10.

You can break out of a repeat loop at any time using the “break” keyword. For example:

```
count = 1;
repeat
    count = count + 1;
    if count == 50 then
        break;
    end
until count > 100
```

Once again, this would exit from the loop as soon as count was equal to 50.

For

The for statement is used to repeat a block of script a specific number of times. The basic syntax is:

```
for variable = start,end,step do
    do something here
end
```

The *variable* can be named anything you want. It is used to “count” the number of trips through the for loop. It begins at the *start* value you specify, and then changes by the amount in *step* after each trip through the loop. In other words, the *step* gets added to the value in the *variable* after the lines between the for and end are performed. If the result is smaller than or equal to the *end* value, the loop continues from the beginning.

For example:

```
-- This loop counts from 1 to 10:
for x = 1, 10 do
    Dialog.Message("Number", x);
end
```

This displays 10 dialog messages in a row, counting from 1 to 10.

Note that the step is optional; if you don’t provide a value for the step, it defaults to 1.

Here's an example that uses a step of "-1" to make the for loop count backwards:

```
-- This loop counts from 10 down to 1:
for x = 10, 1, -1 do
    Dialog.Message("Number", x);
end
```

That example would display 10 dialog messages in a row, counting back from 10 and going all the way down to 1.

You can break out of a for loop at any time using the "break" keyword. For example:

```
for i = 1, 100 do
    if count == 50 then
        break;
    end
end
```

Once again, this would exit from the loop as soon as count was equal to 50.

There is also a variation on the for loop that operates on tables. For more information on that, see *Using For to Enumerate Tables* on page 226.

Tables (Arrays)

Tables are very useful. They can be used to store any type of value, including functions or even other tables.

Creating Tables

There are generally two ways to create a table from scratch. The first way uses curly braces to specify a list of values:

```
my_table = {"apple", "orange", "peach"};

associative_table = {fruit="apple", vegetable="carrot"}
```

The second way is to create a blank table and then add the values one at a time:

```

my_table = {};
my_table[1] = "apple";
my_table[2] = "orange";
my_table[3] = "peach";

associative_table = {};
associative_table.fruit = "apple";
associative_table.vegetable = "carrot";

```

Accessing Table Elements

Each “record” of information stored in a table is known as an *element*. Each element consists of a key, which serves as the index into the table, and a value that is associated with that key.

There are generally two ways to access an element: you can use array notation, or dot notation. Array notation is typically used with numeric arrays, which are simply tables where all of the keys are numbers. Dot notation is typically used with associative arrays, which are tables where the keys are strings.

Here is an example of array notation:

```

t = {"one", "two", "three"};
Dialog.Message("Element one contains:", t[1]);

```

Here is an example of dot notation:

```

t = {first="one", second="two", third="three"};
Dialog.Message("Element 'first' contains:", t.first);

```

Numeric Arrays

One of the most common uses of tables is as arrays. An array is a collection of values that are indexed by numeric keys. In the scripting engine, numeric arrays are one-based. That is, they start at index 1.

Here are some examples using numeric arrays:

Example 1:

```

myArray = {255,0,255};
Dialog.Message("First Number", myArray[1]);

```

This first example would display a dialog message containing the number “255.”

Example 2:

```
alphabet = {"a","b","c","d","e","f","g","h","i","j","k",  
"l","m","n","o","p","q","r","s","t","u","v","w","x","y","z"};  
Dialog.Message("Seventh Letter", alphabet[7]);
```

This would display a dialog message containing the letter “g.”

Example 3:

```
myArray = {};  
myArray[1] = "Option One";  
myArray[2] = "Option Two";  
myArray[3] = "Option Three";
```

This is exactly the same as the following:

```
myArray = {"Option One", "Option Two", "Option Three"};
```

Associative Arrays

Associative arrays are the same as numeric arrays except that the indexes can be numbers, strings or even functions.

Here is an example of an associative array that uses a last name as an index and a first name as the value:

```
arrNames = {Anderson="Jason",  
Clemens="Roger",  
Contreras="Jose",  
Hammond="Chris",  
Hitchcock="Alfred"};  
  
Dialog.Message("Anderson's First Name", arrNames.Anderson);
```

The resulting dialog message would look like this:



Here is an example of a simple employee database that keeps track of employee names and birth dates indexed by employee numbers:

```
Employees = {}; -- Construct an empty table for the employee numbers

-- store each employee's information in its own table
Employee1 = {Name="Jason Anderson", Birthday="07/02/82"};
Employee2 = {Name="Roger Clemens", Birthday="12/25/79"};

-- store each employee's information table
-- at the appropriate number in the Employees table
Employees[100099] = Employee1;
Employees[137637] = Employee2;

-- now typing "Employees[100099]" is the same as typing "Employee1"
Dialog.Message("Birthday", Employees[100099].Birthday);
```

The resulting dialog message would look like this:



Using For to Enumerate Tables

There is a special version of the for statement that allows you to quickly and easily enumerate the contents of an array. The syntax is:

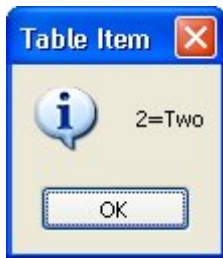
```
for index,value in table do
    operate on index and value
end
```

For example:

```
mytable = {"One","Two","Three"};

-- display a message for every table item
for j,k in mytable do
    Dialog.Message("Table Item", j .. "=" .. k);
end
```

The result would be three dialog messages in a row, one for each of the elements in mytable, like so:



Remember the above for statement, because it is a quick and easy way to inspect the values in a table. If you just want the indexes of a table, you can leave out the *value* part of the for statement:

```

a = {One=1, Two=2, Three=3};

for k in a do
    Dialog.Message("Table Index", k);
end

```

The above script will display three message boxes in a row, with the text “One,” “Three,” and then “Two.”

Whoa there—why aren’t the table elements in order? The reason for this is that internally the scripting engine doesn’t store tables as arrays, but in a super-efficient structure known as a hash table. The important thing to know is that when you define table elements, they are not necessarily stored in the order that you define or add them, unless you use a numeric array (i.e. a table indexed with numbers from 1 to whatever).

Copying Tables:

Copying tables is a bit different from copying other types of values. Unlike variables, you can’t just use the assignment operator to copy the contents of one table into another. This is because the name of a table actually refers to an address in memory where the data within the table is stored. If you try to copy one table to another using the assignment operator, you end up copying the address, and not the actual data.

For example, if you wanted to copy a table, and then modify the copy, you might try something like this:

```

table_one = { mood="Happy", temperature="Warm" };

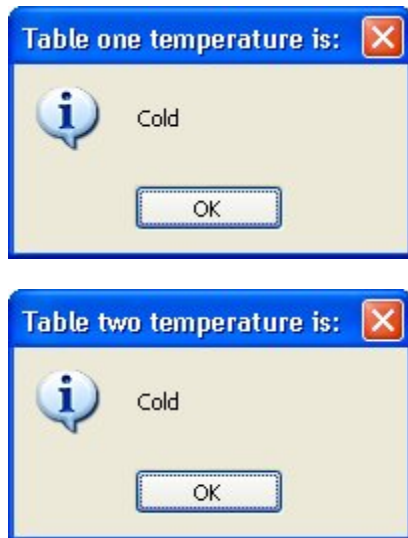
-- create a copy
table_two = table_one;

-- modify the copy
table_two.temperature = "Cold";

Dialog.Message("Table one temperature is:", table_one.temperature);
Dialog.Message("Table two temperature is:", table_two.temperature);

```

If you ran this script, you would see the following two dialogs:



Wait a minute...changing the “temperature” element in `table_two` also changed it in `table_one`. Why would they both change?

The answer is simply because the two are in fact the same table.

Internally, the name of a table just refers to a memory location. When `table_one` is created, a portion of memory is set aside to hold its contents. The location (or “address”) of this memory is what gets assigned to the variable named `table_one`.

Assigning `table_one` to `table_two` just copies that memory address—not the actual memory itself.

It’s like writing down the address of a library on a piece of paper, and then handing that paper to your friend. You aren’t handing the entire library over, shelves of books and all...only the location where it can be found.

If you wanted to actually copy the library, you would have to create a new building, photocopy each book individually, and then store the photocopies in the new location.

That’s pretty much how it is with tables, too. In order to create a full copy of a table, contents and all, you need to create a new table and then copy over all of the elements, one element at a time.

Luckily, the `for` statement makes this really easy to do. For example, here’s a modified version of our earlier example, that creates a “true” copy of `table_one`.

```

table_one = { mood="Happy", temperature="Warm" };

-- create a copy
table_two = {};
for index, value in table_one do
    table_two[index] = value;
end

-- modify the copy
table_two.temperature = "Cold";

Dialog.Message("Table one temperature is:", table_one.temperature);
Dialog.Message("Table two temperature is:", table_two.temperature);

```

This time, the dialogs show that modifying `table_two` doesn't affect `table_one` at all:

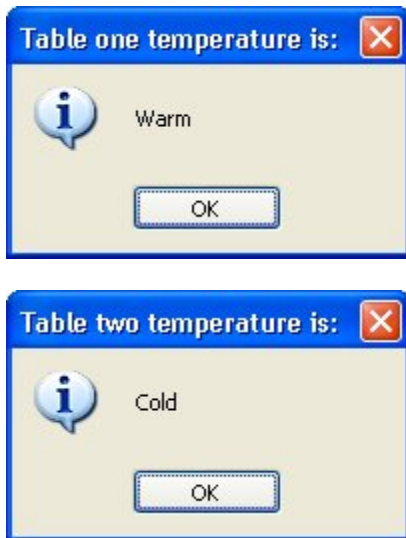


Table Functions

There are a number of built-in table functions at your disposal, which you can use to do such things as inserting elements into a table, removing elements from a table, and counting the number of elements in a table. For more information on these table functions, please see *Program Reference / Actions / Table* in the online help.

Functions

By far the coolest and most powerful feature of the scripting engine is functions. You have already seen a lot of functions used throughout this document, such as “Dialog.Message.” Functions are simply portions of script that you can define, name and then call from anywhere else.

Although there are a lot of built-in TrueUpdate functions, you can also make your own custom functions to suit your specific needs. In general, functions are defined as follows:

```
function function_name (arguments)  
    function script here  
    return return_value;  
end
```

The first part is the keyword “function.” This tells the scripting engine that what follows is a function definition. The *function_name* is simply a unique name for your function. The *arguments* are parameters (or values) that will be passed to the function every time it is called. A function can receive any number of arguments from 0 to infinity (well, not infinity, but don’t get technical on me). The “return” keyword tells the function to return one or more values back to the script that called it.

The easiest way to learn about functions is to look at some examples. In this first example, we will make a simple function that shows a message box. It does not take any arguments and does not return anything.

```
function HelloWorld()  
    Dialog.Message("Welcome", "Hello World");  
end
```

Notice that if you put the above script into an event and build your update, nothing script related happens. Well, that is true and not true. It is true that nothing visible happens but the magic is in what you don’t see. When the event is fired and the function script is executed, the function called “HelloWorld” becomes part of the scripting engine. That means it is now available to the rest of the update in any other script.

This brings up an important point about scripting in TrueUpdate. When making a function, the function does not get “into” the engine until the script is executed. That means that if you define HelloWorld() in a screen’s On Help event, but that event never gets triggered (because the user doesn’t click on the screen’s Help button), the

HelloWorld() function will never exist. That is, you will not be able to call it from anywhere else.

Now back to the good stuff. Let's add a line to actually call the function:

```
function HelloWorld()  
    Dialog.Message("Welcome", "Hello World");  
end  
  
HelloWorld();
```

The "HelloWorld();" line tells the scripting engine to "go perform the function named HelloWorld." When that line gets executed, you would see a welcome message with the text "Hello World" in it.

Function Arguments

Let's take this a bit further and tell the message box which text to display by adding an argument to the function.

```
function HelloWorld(Message)  
    Dialog.Message("Welcome", Message);  
end  
  
HelloWorld("This is an argument");
```

Now the message box shows the text that was "passed" to the function.

In the function definition, "Message" is a variable that will automatically receive whatever argument is passed to the function. In the function call, we pass the string "This is an argument" as the first (and only) argument for the HelloWorld function.

Here is an example of using multiple arguments.

```
function HelloWorld(Title, Message)  
    Dialog.Message(Title, Message);  
end  
  
HelloWorld("This is argument one", "This is argument two");  
HelloWorld("Welcome", "Hi there");
```

This time, the function definition uses two variables, one for each of its two arguments...and each function call passes two strings to the HelloWorld function.

Note that by changing the content of those strings, you can send different arguments to the function, and achieve different results.

Returning Values

The next step is to make the function return values back to the calling script. Here is a function that accepts a number as its single argument, and then returns a string containing all of the numbers from one to that number.

```
function Count(n)

    -- start out with a blank return string
    ReturnString = "";

    for num = 1,n do
        -- add the current number (num) to the end of the return string
        ReturnString = ReturnString..num;

        -- if this isn't the last number, then add a comma and a space
        -- to separate the numbers a bit in the return string
        if (num ~= n) then
            ReturnString = ReturnString..", ";
        end
    end

    -- return the string that we built
    return ReturnString;
end

CountString = Count(10);
Dialog.Message("Count", CountString);
```

The last two lines of the above script uses the Count function to build a string counting from 1 to 10, stores it in a variable named CountString, and then displays the contents of that variable in a dialog message box.

Returning Multiple Values

You can return multiple values from functions as well:

```
function SortNumbers(Number1, Number2)
    if Number1 <= Number2 then
        return Number1, Number2
    else
        return Number2, Number1
    end
end

firstNum, secondNum = SortNumbers(102, 100);
Dialog.Message("Sorted", firstNum .. ", " .. secondNum);
```

The above script creates a function called `SortNumbers` that takes two arguments and then returns two values. The first value returned is the smaller number, and the second value returned is the larger one. Note that we specified two variables to receive the return values from the function call on the second last line. The last line of the script displays the two numbers in the order they were sorted into by the function.

Redefining Functions

Another interesting thing about functions is that you can override a previous function definition simply by re-defining it.

```
function HelloWorld()
    Dialog.Message("Message", "Hello World");
end

function HelloWorld()
    Dialog.Message("Message", "Hello Earth");
end

HelloWorld();
```

The script above shows a message box that says “Hello Earth,” and not “Hello World.” That is because the second version of the `HelloWorld()` function overrides the first one.

Putting Functions in Tables

One really powerful thing about tables is that they can be used to hold functions as well as other values. This is significant because it allows you to make sure that your functions have unique names and are logically grouped. (This is how all of the `TrueUpdate` functions are implemented.) Here is an example:

```
-- Make the functions:
function HelloEarth()
    Dialog.Message("Message", "Hello Earth");
end

function HelloMoon()
    Dialog.Message("Message", "Hello Moon");
end

-- Define an empty table:
Hello = {};

-- Assign the functions to the table:
Hello.Earth = HelloEarth;
Hello.Moon = HelloMoon;

-- Now call the functions:
Hello.Earth();
Hello.Moon();
```

It is also interesting to note that you can define functions right in your table definition:

```
Hello = {
    Earth = function () Dialog.Message("Message", "Hello Earth") end,
    Moon = function () Dialog.Message("Message", "Hello Moon") end
};

-- Now call the functions:
Hello.Earth();
Hello.Moon();
```

String Manipulation

In this section we will briefly cover some of the most common string manipulation techniques, such as string concatenation and comparisons.

(For more information on the string functions available to you in TrueUpdate, see *Program Reference / Actions / String* in the online help.)

Concatenating Strings

We have already covered string concatenation, but it is well worth repeating. The string concatenation operator is two periods in a row (`..`). For example:

```
FullName = "Bo".." Derek"; -- FullName is now "Bo Derek"

-- You can also concatenate numbers into strings
DaysInYear = 365;
YearString = "There are "..DaysInYear.." days in a year.";
```

Note that you can put spaces on either side of the dots, or on one side, or not put any spaces at all. For example, the following four lines will accomplish the same thing:

```
foo = "Hello " .. user_name;
foo = "Hello " .. user_name;
foo = "Hello " ..user_name;
foo = "Hello " ..user_name;
```

Comparing Strings

Next to concatenation, one of the most common things you will want to do with strings is compare one string to another. Depending on what constitutes a “match,” this can either be very simple, or just a bit tricky.

If you want to perform a case-sensitive comparison, then all you have to do is use the equals operator (`==`).

For example:

```
strOne = "Strongbad";
strTwo = "Strongbad";

if strOne == strTwo then
    Dialog.Message("Guess what?", "The two strings are equal!");
else
    Dialog.Message("Hmmm", "The two strings are different.");
end
```

Since the == operator performs a case-sensitive comparison when applied to strings, the above script will display a message box proclaiming that the two strings are equal.

If you want to perform a case-*insensitive* comparison, then you need to take advantage of either the String.Upper or String.Lower function, to ensure that both strings have the same case before you compare them. The String.Upper function returns an all-uppercase version of the string it is given, and the String.Lower function returns an all-lowercase version. Note that it doesn't matter which function you use in your comparison, so long as you use the same function on both sides of the == operator in your if statement.

For example:

```
strOne = "Moohahahaha";
strTwo = "MOOohaHAHAha";

if String.Upper(strOne) == String.Upper(strTwo) then
    Dialog.Message("Guess what?", "The two strings are equal!");
else
    Dialog.Message("Hmmm", "The two strings are different.");
end
```

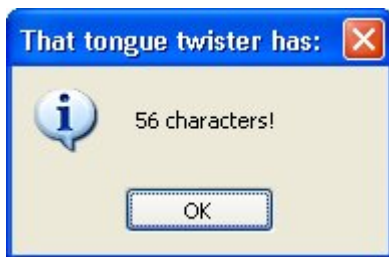
In the example above, the String.Upper function converts strOne to "MOOOHAHAHAHA" and strTwo to "MOOOHAHAHAHA" and then the if statement compares the results. (Note: the two original strings remain unchanged.) That way, it doesn't matter what case the original strings had; all that matters is whether the letters are the same.

Counting Characters

If you ever want to know how long a string is, you can easily count the number of characters it contains. Just use the `String.Length` function, like so:

```
twister = "If a wood chuck could chuck wood, how much would...um...";
num_chars = String.Length(twister);
Dialog.Message("That tongue twister has:", num_chars .. " characters!");
```

...which would produce the following dialog message:



Finding Strings:

Another common thing you'll want to do with strings is to search for one string within another. This is very simple to do using the `String.Find` action.

For example:

```
strSearchIn = "Isn't it a wonderful day outside?";
strSearchFor = "wonder";

-- search for strSearchIn inside strSearchFor
nFoundPos = String.Find(strSearchIn, strSearchFor);

if nFoundPos ~= nil then
    -- found it!
    Dialog.Message("Search Result", strSearchFor ..
        " found at position " .. nFoundPos);
else
    -- no luck
    Dialog.Message("Search Result", strSearchFor.." not found!");
end
```

...would cause the following message to be displayed:



Tip: Try experimenting with different values for `strSearchFor` and `strSearchIn`.

Replacing Strings:

One of the most powerful things you can do with strings is to perform a search and replace operation on them.

The following example shows how you can use the `String.Replace` action to replace every occurrence of a string with another inside a target string:

```
strTarget      = "There can be only one. Only one is allowed!";
strSearchFor   = "one";
strReplaceWith = "a dozen";
strNewString   = String.Replace( strTarget
                                , strSearchFor
                                , strReplaceWith );

Dialog.Message("After searching and replacing:", strNewString);

-- create a copy of the target string with no spaces in it
strNoSpaces = String.Replace(strTarget, " ", "");

Dialog.Message("After removing spaces:", strNoSpaces);
```

The above example would display the following two messages:



Extracting Strings

There are three string functions that allow you to “extract” a portion of a string, rather than copying the entire string itself. These functions are `String.Left`, `String.Right`, and `String.Mid`.

`String.Left` copies a number of characters from the beginning of the string.

`String.Right` does the same, but counting from the right end of the string instead.

`String.Mid` allows you to copy a number of characters starting from any position in the string.

You can use these functions to perform all kinds of advanced operations on strings.

Here’s a basic example showing how they work:

```
strOriginal = "It really is good to see you again.";

-- copy the first 13 characters into strLeft
strLeft = String.Left(strOriginal, 13);

-- copy the last 18 characters into strRight
strRight = String.Right(strOriginal, 18);
```



```
-- create a new string with the two pieces
strNeo = String.Left .. "awesome" .. strRight .. " Whoa.";

-- copy the word "good" into strMiddle
strMiddle = String.Mid(strOriginal, 13, 4);
```

Converting Numeric Strings into Numbers

There may be times when you have a numeric string, and you need to convert it to a number.

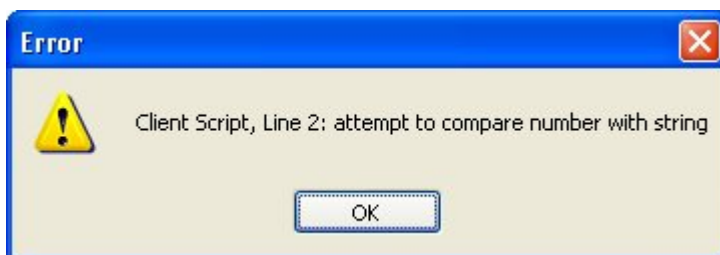
For example, if you have an input field where the user can enter their age, and you read in the text that they typed, you might get a value like “31”. Because they typed it in, though, this value is actually a string consisting of the characters “3” and “1”.

If you tried to compare this value to a number, you would get a syntax error saying that you attempted to compare a number with a string.

For example, the following script:

```
age = "31";
if age > 18 then
    Dialog.Message("", "You're older than 18.");
end
```

...would produce the following error message if run from the Client Script:



The problem in this case is the line that compares the contents of the variable “age” with the number 18:

```
if age > 18 then
```

This generates an error because age contains a string, and not a number. The script engine doesn’t allow you to compare numbers with strings in this way. It has no way of knowing whether you wanted to treat age as a number, or treat 18 as a string.

The solution is simply to convert the value of age to a number before comparing it. There are two ways to do this. One way is to use the `String.ToNumber` function.

The `String.ToNumber` function translates a numeric string into the equivalent number, so it can be used in a numeric comparison.

```
age = "31";  
if String.ToNumber(age) > 18 then  
    Dialog.Message("", "You're older than 18.");  
end
```

The other way takes advantage of the scripting engine’s ability to convert numbers into strings when it knows what your intentions are. For example, if you’re performing an arithmetic operation (such as adding two numbers), the engine will automatically convert any numeric strings to numbers for you:

```
age = "26" + 5; -- result is a numeric value
```

The above example would not generate any errors, because the scripting engine understands that the only way the statement makes sense is if you meant to use the numeric string as a number. As a result, the engine automatically converts the numeric string to a number so it can perform the calculation.

Knowing this, we can convert a numeric string to a number without changing its value by simply adding 0 to it, like so:

```
age = "31";  
if (age + 0) > 18 then  
    Dialog.Message("", "You're older than 18.");  
end
```

In the preceding example, adding zero to the variable gets the engine to convert the value to a number, and the result is then compared with 18. No more error.

Other Built-in Functions

Script Functions

There are three other built-in functions that may prove useful to you: `dofile`, `require`, and `type`.

dofile

Loads and executes a script file. The contents of the file will be executed as though it was typed directly into the script. The syntax is:

```
dofile(file_path);
```

For example, say we typed the following script into a file called `MyScript.lua` (just a text file containing this script, created with notepad or some other text editor):

```
Dialog.Message( "Hello", "World" );
```

Assume that the file was distributed with our update.

Now wherever the following line of script is added:

```
dofile(_TempFolder.."\\MyScript.lua");
```

...that script file will be read in and executed immediately. In this case, you would see a message box with the friendly “hello world” message.

Tip: Use the `dofile` function to save yourself from having to re-type or re-paste a script into your projects over and over again.

require

Loads a script file into the scripting engine and runs it. It is similar to `dofile` except that it will only load a given file once per session, whereas `dofile` will re-load and re-run the file each time it is used. The syntax is:

```
require(file_path);
```

So, for example, even if you do two `requires` in a row:

```
require("%TempFolder%\\foo.lua");  
require("%TempFolder%\\foo.lua"); -- this line won't do anything
```

...only the first one will ever get executed. After that, the scripting engine knows that the file has been loaded and run, and future calls to require that file will have no effect.

Since require will only load a given script file once per session, it is best suited for loading scripts that contain only variables and functions. Since variables and functions are global by default, you only need to “load” them once; repeatedly loading the same function definition would just be a waste of time.

This makes the require function a great way to load external script libraries. Every script that needs a function from an external file can safely require() it, and the file will only actually be loaded the first time it’s needed.

type

This function will tell you the type of value contained in a variable. It returns the string name of the variable type. Valid return values are “nil,” “number,” “string,” “boolean,” “table,” or “function.” For example:

```
a = 989;
strType = type(a); -- sets strType to "number"

a = "Hi there";
strType = type(a); -- sets strType to "string"
```

The type function is especially useful when writing your own functions that need certain data types in order to operate. For example, the following function uses type() to make sure that both of its arguments are numbers:

```
-- find the maximum of two numbers
function Max(Number1, Number2)
    -- make sure both arguments are numeric
    if (type(Number1) ~= "number") or (type(Number2) ~= "number") then
        Dialog.Message("Error", "Please enter numbers");
        return nil; -- we're using nil to indicate an error condition
    else
        if Number1 >= Number2 then
            return Number1;
        else
            return Number2;
        end
    end
end
```

Actions

TrueUpdate comes with a large number of built-in functions. In the program interface, these built-in functions are commonly referred to as *actions*. For scripting purposes, actions and functions are essentially the same; however, the term “actions” is generally reserved for those functions that are built into the program and are included in the alphabetical list of actions in the online help. When referring to functions that have been created by other users or yourself, the term “functions” is preferred.

Debugging Your Scripts

Scripting (or any kind of programming) is relatively easy once you get used to it. However, even the best programmers make mistakes, and need to iron the occasional wrinkle out of their code. Being good at debugging scripts will reduce the time to market for your projects and increase the amount of sleep you get at night. Please read this section for tips on using TrueUpdate as smartly and effectively as possible!

This section will explain TrueUpdate’s error handling methods as well as cover a number of debugging techniques.

Error Handling

All of the built-in TrueUpdate actions use the same basic error handling techniques. However, this is not necessarily true of any third-party functions, modules or scripts—even scripts developed by Indigo Rose Corporation that are not built into the product. Although these externally developed scripts can certainly make use of TrueUpdate’s error handling system, they may not necessarily do so. Therefore, you should always consult a script or module’s author or documentation in order to find out how error handling is, well, handled.

There are two kinds of errors that you can have in your scripts when calling TrueUpdate actions: syntax errors, and functional errors.

Syntax Errors

Syntax errors occur when the syntax (or “grammar”) of a script is incorrect, or a function receives arguments that are not appropriate. Some syntax errors are caught by TrueUpdate when you build your application.

For example, consider the following script:

```
foo =
```

This is incorrect because we have not assigned anything to the variable `foo`—the script is incomplete. This is a pretty obvious syntax error, and would be caught by the scripting engine at build time (when you build your project).

Another type of syntax error is when you do not pass the correct type or number of arguments to a function. For example, if you try and run this script:

```
Dialog.Message( "Hi There" );
```

...the project will build fine, because there are no *obvious* syntax errors in the script. As far as the scripting engine can tell, the function call is well formed. The name is valid, the open and closed parentheses match, the quotes are in the right places, and there's even a terminating semi-colon at the end. Looks good!

However, at run time you would see something like the following:



Looks like it wasn't so good after all. Note that the message says two arguments are required for the `Dialog.Message` action. Ah. Our script only provided one argument.

According to the function prototype for `Dialog.Message`, it looks like the action can actually accept up to *five* arguments:

```
number Dialog.Message ( string Title,  
                        string Text,  
                        number Type = MB_OK,  
                        number Icon = MB_ICONNONE,  
                        number DefaultButton = MB_DEFBUTTON1 )
```

Looking closely at the function prototype, we see that the last three arguments have default values that will be used if those arguments are omitted from the function call. The first two arguments—Title and Text—don't have default values, so they cannot be omitted without generating an error. To make a long story short, it's okay to call the Dialog.Message action with anywhere from 2 to 5 arguments...but 1 argument isn't enough.

Fortunately, syntax errors like these are usually caught at build time or when you test your application. The error messages are usually quite clear, making it easy for you to locate and identify the problem.

Functional Errors

Functional errors are those that occur because the functionality of the action itself fails. They occur when an action is given incorrect information, such as the path to a file that doesn't exist. For example, the following code will produce a functional error:

```
filecontents = TextFile.ReadToString("this_file_don't exist.txt");
```

If you put that script into an event or script tab right now and try it, you will see that nothing appears to happen. This is because TrueUpdate's functional errors are not automatically displayed the way syntax errors are. We leave it up to you to handle (or to not handle) such functional errors yourself.

The reason for this is that there may be times when you don't care if a function fails. In fact, you may expect it to. For example, the following code tries to remove a folder called C:\My Temp Folder:

```
Folder.Delete("C:\\My Temp Folder");
```

However, in this case you don't care if it really gets deleted, or if the folder didn't exist in the first place. You just want to make sure that if that particular folder exists, it will be removed. If the folder isn't there, the Folder.Delete action causes a functional error, because it can't find the folder you told it to delete...but since the end result is exactly what you wanted, you don't need to do anything about it. And you certainly don't want the user to see any error messages.

Conversely, there may be times when it is very important for you to know if an action fails. Say for instance that you want to copy a very important file:

```
File.Copy("C:\\Temp\\My File.dat", "C:\\Temp\\My File.bak");
```

In this case, you really want to know if it fails and may even want to exit the program or inform the user. This is where the Debug actions come in handy. Read on.

Debug Actions

TrueUpdate comes with some very useful functions for debugging your updates. This section will look at a number of them.

Application.GetLastError

This is the most important action to use when trying to find out if a problem has occurred. At run time there is always an internal value that stores the status of the last action that was executed. At the start of an action, this value is set to 0 (the number zero). This means that everything is OK. If a functional error occurs inside the action, the value is changed to some non-zero value instead.

This last error value can be accessed at any time by using the Application.GetLastError action.

The syntax is:

```
last_error_code = Application.GetLastError();
```

Here is an example that uses this action:

```
File.Copy("C:\\Temp\\My File.dat", "C:\\Temp\\My File.bak");

error_code = Application.GetLastError();
if (error_code ~= 0) then
    -- some kind of error has occurred!
    Dialog.Message("Error", "File copy error: "..error_code);
    Application.Exit();
end
```

The above script will inform the user that an error occurred and then exit the application. This is not necessarily how all errors should be handled, but it illustrates the point. You can do anything you want when an error occurs, like calling a different function or anything else you can dream up.

The above script has one possible problem. Imagine the user seeing a message like this:

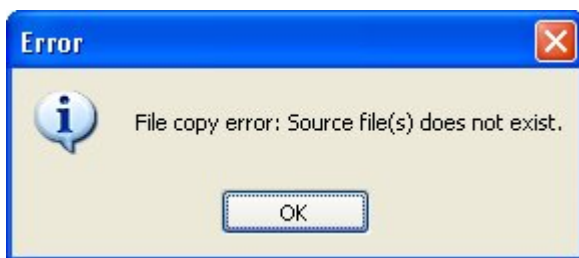


It would be much nicer to actually tell them some information about the exact problem. Well, you are in luck! At run time there is a table called `_tblErrorMessage`s that contains all of the possible error messages, indexed by the error codes. You can easily use the last error number to get an actual error message that will make more sense to the user than a number like “1021.”

For example, here is a modified script to show the actual error string:

```
File.Copy("C:\\Temp\\My File.dat", "C:\\Temp\\My File.bak");  
  
error_code = Application.GetLastError();  
  
if (error_code <= 0) then  
    -- some kind of error has occurred!  
    Dialog.Message( "Error", "File copy error: "  
        .. _tblErrorMessages[error_code] );  
  
    Application.Exit();  
end
```

Now the script will produce the following error message:



Much better information!

Just remember that the value of the last error gets reset every time an action is executed. For example, the following script would not produce an error message:

```
File.Copy("C:\\Temp\\My File.dat", "C:\\Temp\\My File.bak");

-- At this point Application.GetLastError() could be non-zero, but...

Dialog.Message("Hi There", "Hello World");

-- Oops, now the last error number will be for the Dialog.Message action,
-- and not the File.Copy action. The Dialog.Message action will succeed,
-- resetting the last error number to 0, and the following lines will not
-- catch any error that happened in the File.Copy action.

error_code = Application.GetLastError();

if (error_code ~= 0) then

    -- some kind of error has occurred!
    Dialog.Message( "Error", "File copy error: "
        .. _tblErrorMessages[error_code] );

    Application.Exit();

end
```

Debug.ShowWindow

The TrueUpdate runtime has the ability to show a debug window that can be used to display debug messages. This window exists throughout the execution of your update, but is only visible when you tell it to be.

The syntax is:

```
Debug.ShowWindow(show_window);
```

...where *show_window* is a Boolean value. If true, the debug window is displayed, if false, the window is hidden. For example:

```
-- show the debug window
Debug.ShowWindow(true);
```

If you call this script, the debug window will appear on top of your update, but nothing else will really happen. That's where the following Debug actions come in.

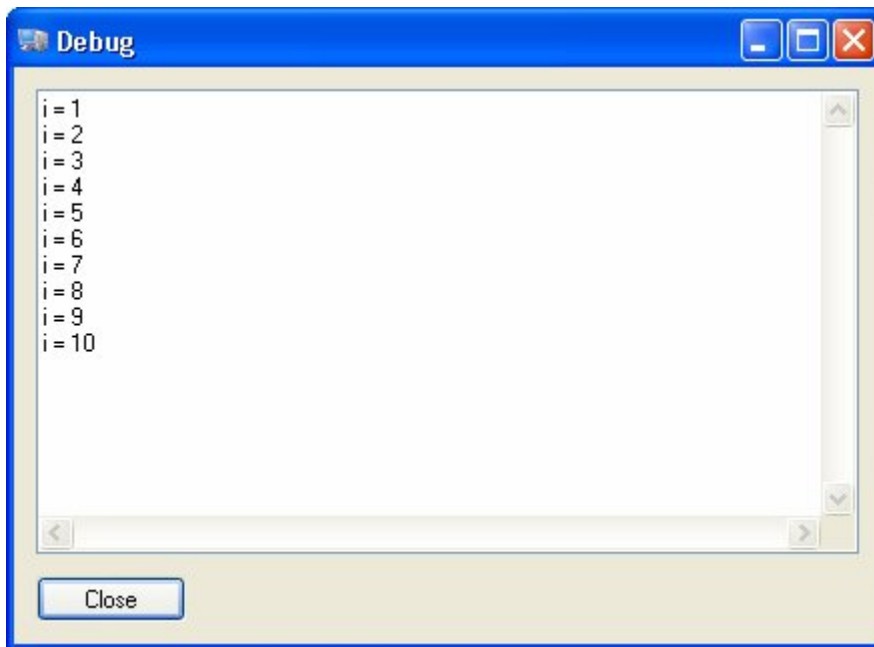
Debug.Print

This action prints the text of your choosing in the debug window. For example, try the following script:

```
Debug.ShowWindow(true);  
  
for i = 1, 10 do  
    Debug.Print("i = " .. i .. "\r\n");  
end
```

The “\r\n” part is actually two escape sequences that are being used to start a new line. (This is technically called a “carriage return/linefeed” pair.) You can use \r\n in the debug window whenever you want to insert a new line.

The above script will produce the following output in the debug window:



You can use this method to print all kinds of information to the debug window. Some typical uses are to print the contents of a variable so you can see what it contains at run time, or to print your own debug messages like “inside outer for loop” or “foo() function started.” Such messages form a trail like bread crumbs that you can trace in

order to understand what's happening behind the scenes in your project. They can be invaluable when trying to debug your scripts or test your latest algorithm.

Debug.SetTraceMode

TrueUpdate can run in a special “trace” mode at run time that will print information about every line of script that gets executed to the debug window, including the value of `Application.GetLastError()` if the line involves calling a built-in action. You can turn this trace mode on or off by using the `Debug.SetTraceMode` action:

```
Debug.SetTraceMode(turn_on);
```

...where *turn_on* is a Boolean value that tells the program whether to turn the trace mode on or off.

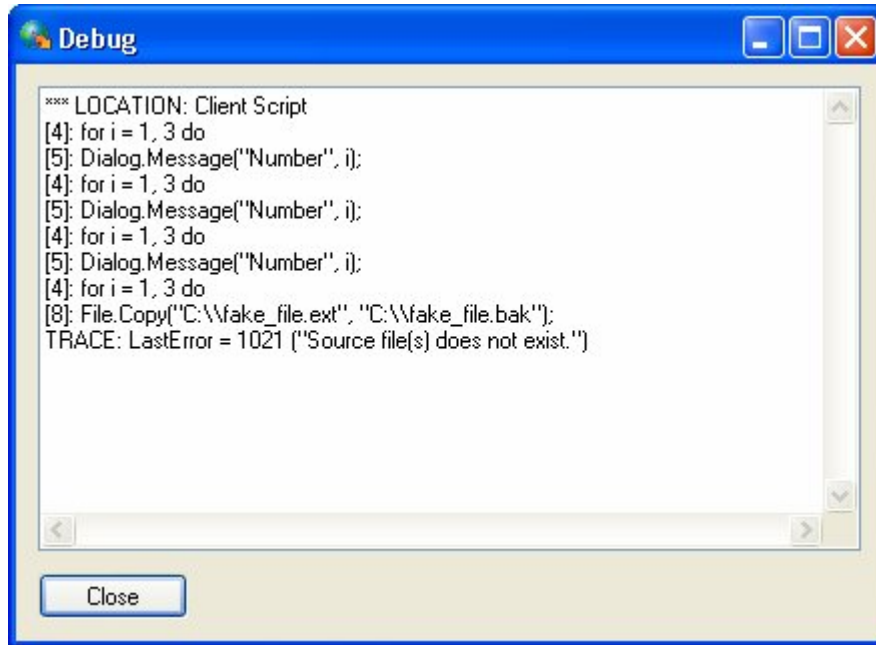
Here is an example:

```
Debug.ShowWindow(true);
Debug.SetTraceMode(true);

for i = 1, 3 do
    Dialog.Message("Number", i);
end

File.Copy("C:\\fake_file.ext", "C:\\fake_file.bak");
```

Running that script will produce the following output in the debug window:



Notice that every line produced by the trace mode starts with “TRACE:” This is so you can tell them apart from any lines you send to the debug window with `Debug.Print`. The number after the “TRACE:” part is the line number that is currently being executed in the script.

Turning trace mode on is something that you will not likely want to do in your final, distributable update, but it can really help find problems during development.

Debug.GetEventContext

The `Debug.GetEventContext` action is used to get a descriptive string about the event that is currently being executed. This can be useful if you define a function in one place but call it somewhere else, and you want to be able to tell where the function is being called from at any given time.

For example, if you execute this script from the On Preload event of an Edit Fields screen:

```
Dialog.Message("Event Context", Debug.GetEventContext());
```

...you will see something like this:



Dialog.Message

This brings us to good ole' `Dialog.Message`. You have seen this action used throughout this document, and for good reason. This is a great action to use throughout your code when you are trying to track down a problem.

For example, you can use it to display the current contents of a variable that you're working with:

```
Dialog.Message("The current value of nCats is: " .. nCats);
```

You can also use it to put up messages at specific points in a script, to break it into arbitrary stages. This can be helpful when you're not sure where in a script an error is occurring:

```
function foobar(arg1, arg2)

    Dialog.Message("Temporary Debug Msg", "In foobar()");

    -- bunch of script

    Dialog.Message("Temporary Debug Msg", "1");

    -- bunch of script

    Dialog.Message("Temporary Debug Msg", "2");

    -- bunch of script

    Dialog.Message("Temporary Debug Msg", "Leaving foobar()");

end
```

Final Thoughts

Hopefully this chapter has helped you to understand scripting in TrueUpdate. Once you get the hang of it, it is a really fun, powerful way to get things done.

Other Resources

Here is a list of other places that you can go for help with scripting in TrueUpdate.

Help File

The TrueUpdate help file is packed with good reference material for all of the actions, events and script tabs supported by TrueUpdate, and for the design environment itself. You can access the help file at any time by choosing Help > TrueUpdate Help from the menu.

Tip: If you are in a screen's action editor, or a script tab and you want to learn more about an action, simply click on the action and press the F1 key on your keyboard.

TrueUpdate Web Site

The TrueUpdate web site is located at <http://www.trueupdate.com>. Be sure to check out the user forums where you can read questions and answers by fellow users and Indigo Rose staff as well as ask questions of your own.

Tip: A quick way to access the online forums is to choose Help > User Forums from the menu.

Indigo Rose Technical Support

If you need help with any scripting concepts or have a mental block to push through, feel free to open a support ticket at <http://support.indigorose.com>. Although we can't write scripts for you or debug your specific scripts, we will be happy to answer any general scripting questions that you have.

The Lua Web Site

TrueUpdate's scripting engine is based on a popular scripting language called *Lua*. Lua is designed and implemented by a team at Tecgraf, the Computer Graphics Technology Group of PUC-Rio (the Pontifical Catholic University of Rio de Janeiro in Brazil). You can learn more about Lua and its history at the official Lua web site:

<http://www.lua.org>

The Lua website is also where you can find the latest documentation on the Lua language, along with tutorials and a really friendly community of Lua developers.

Note that there may be other built-in functions that exist in Lua and in TrueUpdate that are not officially supported in TrueUpdate. These functions, if any, are documented in the Lua 5.0 Reference Manual.

Only the functions listed in the online help are supported by Indigo Rose Software. Any other “undocumented” functions that you may find in the Lua documentation are not supported. Although these functions may work, you must use them entirely on your own.



INDEX

- .
- .ts1, 31, 35, 174, 180, 181
- .ts2, 35, 174, 180, 181
- .ts3, 35, 175, 180, 181
- |
- \r\n, 251
-
- _CommandLineArgs, 166
- _ProgramFilesFolder, 138
- _SourceFolder, 192
- _tblErrorMessages, 249
- _WindowsFolder, 138
- A**
- access rights, 57
- accessing table elements, 224
- action wizard, 87, 89, 95
- actions, 78, 108, 124, 245
 - adding, 88
 - editing, 93
- adding languages, 158
- adding screens, 106
- adding servers, 132
- adding the client files to your software, 192
- alternative interfaces, 126
- Application.Exit, 248, 249, 250
- Application.ExitScript, 113
- Application.GetInstallLanguage, 163, 164, 165
- Application.GetLastError, 248, 252
- Application.GetUpdateLanguage, 167
- Application.SetUpdateLanguage, 165
- arguments, 231, 232
- arithmetic operators, 215

- arrays. *See* tables
 - accessing elements, 224
- assignment, 214
- associative arrays, 225
- attributes, 108, 111
- authentication, 46
- autocomplete, 81, 82, 83
- auto-save preferences, 73

B

- Back button, 103, 108, 110, 111, 113
- banner style, 108, 115, 117
- blowfish, 174
- body, 116
- Boolean variables, 212, 244
- build process, 180, 191
- build settings, 184
- building and distributing, 178
- built-in session variables, 138

C

- calling ShellExecute from C/C++, 195
- calling ShellExecute from Visual Basic, 194
- Cancel button, 111
- case sensitive, 202
- changing the current language, 165
- choosing a theme, 121
- client, 29
- client data file, 172, 174, 185
- client executable, 172, 175, 181, 185
- client screens, 34, 64, 104
- client script, 33, 85
- client-server communication, 175
- command line arguments, 57, 166, 189, 191
- comments, 88, 202
- comparing strings, 236

- concatenating strings, 236
- concatenation operator (..), 217, 236
- constants, 94, 188, 189
- context sensitive help, 85
- control area, 119
- control area offsets, 119
- control structures, 219–23
- controls, 87, 113, 115, 119
- converting numeric strings into numbers, 241
- copying tables, 228
- counting characters, 238
- CRC value, 53
- creating a custom theme, 122
- creating tables, 223
- creating the user interface, 100
- creation date, 52
- Ctrl+Space, 82
- current language, 163, 165
- custom client-server communication, 176
- custom session variables, 142
- custom themes, 122
- customizing error messages and prompts, 162

D

- debug actions, 248
- debug window, 250, 251
- Debug.GetEventContext, 253
- Debug.Print, 251
- Debug.SetTraceMode, 252
- Debug.ShowWindow, 250, 252
- debugging your scripts, 245
- default language, 43, 156, 165
- delimiters, 203
- design environment, 64
- determining the current language, 163
- development environment, the, 60
- dialog actions, 124
- dialog style, 41
- Dialog.Message, 88, 89, 90, 93, 124, 198, 200, 205, 224, 225, 231, 235, 246, 247, 248, 249, 250, 252, 254

- Dialog.TimedMessage, 124
- dialog-based updates, 126
- disabled, 111
- distributing your client, 178
- docking panes, 67
- document preferences, 73
- dofile, 243
- double quotes, 208
- download location, 57
- download method, 44, 55
- download settings, 46, 56
- dynamic control layout, 119

E

- editing actions, 93
- editing screens, 107
- editing servers, 132
- else, 219
- elseif, 219
- enabled, 111
- enumerating tables, 226
- environment preferences, 74
- error handling, 245
- error messages, 162
- escape sequences, 209, 251
- escaping backslashes, 210
- escaping strings, 209
- events, 86, 111, 112
- existing translated messages, 167
- expanding session variables, 147
- exporting screen translations, 162
- expressions, 215
- extracting strings, 240

F

- F1 help, 69, 85, 255
- file copy, 187
- file information, 51
- file version, 52
- File.Copy, 247, 248, 249, 250, 252
- File.Find, 125
- finding strings, 238
- folder preferences, 74

- Folder.Delete, 247
- footer, 116
- for, 222, 226
- forums, 71
- frequently asked questions, 20
- FTP, 45, 49, 131, 187
- function arguments, 232
- function definition, 232
- function prototypes, 95, 246
- functional errors, 247
- functions, 213, 231, 244

G

- getting additional language files, 157
- getting help, 69, 95
- ghosting buttons, 111
- global environment, 201
- global variables, 204

H

- header, 115
- HelloWorld function, 231, 232, 234
- help, 69
- Help button, 108
- help file, 255
- help pane, 70
- HTTP, 45, 131
- HTTP authentication, 46
- HTTP.Download, 125
- HTTP.Submit, 177
- HTTP.SubmitSecure, 177
- HTTPS, 45, 131, 176

I

- if, 219
- importing screen translations, 162
- includes, 96
- ini file, 51
- integrating the client into your software, 192
 - source code, 194
- intellisense, 81, 82
- interface, 100

- interface type, 41
- internationalizing, 154
- introduction to scripting, 76

K

- keyboard shortcuts, 24
- knowledge base, 71

L

- LAN, 45, 131, 187
- language detection, 154
- language files, 156, 167
- language ID, 154, 163, 165
- language identifier. *See* language ID
- language manager, 155
- language selector, 43, 109, 159
- languages, 42, 152
 - adding, 158
 - removing, 159
 - testing, 166
- local variables, 205
- localized strings, 167
- localizing actions, 166
- localizing screens, 160
- log files, 196
- logical operators, 217
- Lua variables, 204
- Lua web site, 255

M

- making your own language file, 157
- MD5, 173
- menu commands, 24
- message IDs, 167
- modified date, 53
- moving panes, 67
- multi-line comments, 203
- multiple arguments, 232
- multiple server scripts, 86

N

- naming variables, 206

- navigation, 110
- navigation actions, 112
- navigation buttons, 111
- navigation events, 112
- Next button, 103, 108, 110, 111, 113
- nil, 211, 244
- null characters, 208
- numbers, 208, 244
- numeric arrays, 224

O

- On Back, 111, 112, 113
- On Cancel, 112
- On Ctrl Message, 112
- On Help, 112
- On Next, 111, 112, 113
- On Preload, 112, 149
- On Startup, 200, 201, 253
- online forums, 71
- online help, 85
- operating systems, 196
- operator precedence, 218
- operators, 215
- order of table elements, 228
- other uses for TrueUpdate, 28
- output folder, 74, 181, 184, 185
- Output tab, 185
- overriding functions, 234
- overriding themes, 123

P

- panes, 64, 68
 - docking and undocking, 66, 67
 - moving, 67
 - pinning and unpinning, 68
 - resizing, 65
- parameters, 90, 93, 231
- passive mode, 50
- pinned panes, 68
- pinning panes, 68
- plugins, 97
- port, 46
- post-build steps, 190

- pre-build steps, 190
- preferences, 72, 191
- prefix, 181, 182, 185
- primary ID. *See* primary language ID
- primary language ID, 154, 164, 165
- product information, 144
- product version, 52
- program menu, 63
- program window, 62
- programming environment, 79
- progress screens, 108, 125
- project files, 74
- project templates, 99
- project wizard, 36, 39, 99
- prompts, 162
- protocol, 44
- publish report, 184
- publish settings, 184
- publish wizard, 92, 181, 191
- publishing, 178
- putting functions in tables, 235

Q

- quick help, 63, 64, 83

R

- redefining functions, 234
- redundancy, 132
- registry key, 51
- relational operators, 216
- removing languages, 159
- removing screens, 107
- removing servers, 132
- removing session variables, 145
- repeat, 221
- replacing strings, 239
- require, 243
- reserved keywords, 207
- resizing panes, 65
- resources, 96
- return, 231, 233
- return code, 57
- return values, 110

- returning multiple values, 234
- returning values, 233
- run after build, 190
- run before build, 190
- running TrueUpdate, 193
- run-time language detection, 154

S

- sample projects, 99
- scalability, 133
- scope, 204
- screen attributes, 108
- screen controls, 113
- screen events, 79, 86, 87
- screen ID, 160
- screen layout, 115
- screen lists, 105
- screen navigation, 110
- screen panes, 63, 104
- screen preview, 63, 64, 66
- screen properties, 108
- screen settings, 108
- screen tabs, 63
- Screen.Back, 111, 113
- Screen.End, 113
- Screen.GetLocalizedString, 149, 168
- Screen.Jump, 83, 113
- Screen.Next, 111, 113
- Screen.Previous, 113
- Screen.SetLocalizedString, 168
- Screen.Show, 107
- screens, 102, 146
 - adding, 106
 - editing, 107
 - removing, 107
- script editor, 63, 64, 79
- script files, 96
- script help toolbar, 69
- script tabs, 33, 63, 79
- scripting, 76, 78, 198
 - important scripting concepts, 201
- scripting resources, 200–256
- scripts, 32, 78

- secondary ID. *See* secondary language ID
- secondary language ID, 154, 164, 165
- secure protocols, 176
- security, 170
- self-updating, 35
- server configuration files, 30
- server data file, 174
- server files prefix, 181, 182, 185
- server redundancy, 132
- server scalability, 133
- server screens, 35, 64, 105
- server script, 85
- server scripts, 34, 86
- servers, 30, 130
- session variables, 110, 136, 138
 - expanding, 146
 - removing, 145
 - setting, 142–44
- session variables tab, 142
- SessionVar.Expand, 110, 146, 147, 149
- SessionVar.Get, 110, 148
- SessionVar.Remove, 110, 145
- SessionVar.Set, 110, 144, 149
- setting session variables, 142–44
- setting the current language, 165
- Setup Factory, 14, 16, 23, 51, 189, 192
- SFTP, 176, 187
- Shell.GetFolder, 138
- showing screens, 107
- side banner, 118
- silent update, 41
- silent updates, 126
- single quotes, 209
- source code, 194
- SSH, 176
- SSL, 176
- standard toolbar, 63
- starting a new project, 38
- status bar, 63
- status dialogs, 125
- storing functions in tables, 235
- String.Find, 238
- String.Left, 240

- String.Length, 238
- String.Lower, 237
- String.Mid, 240
- String.Replace, 239
- String.Right, 240
- String.ToNumber, 242
- String.Upper, 237
- strings, 208, 244
 - comparing, 236
 - concatenating, 236
 - converting to numbers, 241
 - counting characters, 238
 - extracting, 240
 - finding, 238
 - manipulating, 242
 - replacing, 239
- style, 108
- syntax errors, 245
- syntax highlighting, 81
- System.GetDefaultLangID, 163, 164, 167

T

- table functions, 230
- tables, 213, 223–30, 244
 - accessing elements, 224
 - associative arrays, 225
 - copying, 228
 - creating, 223
 - enumerating, 226
 - numeric arrays, 224
 - order of elements, 228
- target version, 53
- taskbar settings, 124
- taskbar visibility, 126
- technical support, 71, 255
- testing, 196
- testing different languages, 166
- TextFile.ReadToString, 247
- the build process, 180
- the development environment, 60
- the TrueUpdate model, 26
- the update process, 31
- theme, 42

- themes, 120, 122, 123
- timeout, 46
- toolbars, 63
- top banner, 117
- translated messages, 167
- translated strings, 167
- translating actions, 166
- translating screens, 160
- triggering TrueUpdate, 193
- TrueUpdate 2.0, 11
 - key features, 12
 - web site, 255
 - what's new in 2.0, 15
- TrueUpdate Client, 29, 62, 130, 175, 181
- TrueUpdate program window, 62
- TrueUpdate Servers, 30, 128, 130, 175, 182
- TrueUpdate.GetLocalizedString, 167
- TrueUpdate.GetServerFile, 133, 175
- TrueUpdate.GetUpdateServerList, 133
- TrueUpdate.SetLocalizedString, 168
- ts1, 31, 35, 174, 180, 181
- ts2, 35, 174, 180, 181
- ts3, 35, 175, 180, 181
- type, 244
- types, 207

U

- unattended builds, 189
- undo/redo preferences, 73
- un-docking panes, 66
- undocumented functions, 256
- unpinned panes, 68
- unpinning panes, 68
- update method, 54
- update process, 31
- updating an existing client, 35
- updating TrueUpdate, 62
- upload, 48
- upload locations, 182, 186
- user forums, 71
- user interface, 100
- using session variables on screens, 146

using the action wizard, 87

V

values, 207

variable assignment, 214

variable scope, 204

variables, 136, 204

 naming, 206

version identification, 51

visibility, 111

Visual Patch, 14, 16, 23

W

Welcome dialog, 39

what TrueUpdate does, 28

while, 220

wizard, 103

 action wizard, 87, 89, 95

 project wizard, 36, 39, 99

 publish wizard, 92, 181, 191

wizard style, 41

