



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

AutoPlay Media Studio 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 © 2007 Indigo Rose Software Design Corporation.

All Rights Reserved.

FMOD sound and music system is copyright © Firelight Technologies, Pty Ltd. 1994-2003.

LUA is copyright © 2003 Tecgraf, PUC-Rio.

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 | 8 |
| What Can I Create With AutoPlay Media Studio?..... | 9 |
| About This Guide | 10 |
| Document Conventions | 11 |
| | |
| LESSON 1: GETTING STARTED | 12 |
| Starting a New Project..... | 14 |
| Making Sure You Have the Latest Version..... | 18 |
| Learning the Interface..... | 18 |
| Getting Help | 24 |
| Setting Preferences..... | 25 |
| Modifying the Application Settings | 29 |
| | |
| LESSON 2: GRAPHICS AND TEXT | 34 |
| Choosing a Page Background | 37 |
| Adding Image Objects | 48 |
| Adding Label Objects | 56 |
| Duplicating Objects | 60 |
| Changing Text..... | 61 |
| Naming Objects..... | 66 |
| Changing Font Settings..... | 67 |
| Changing Text Colors..... | 72 |
| Copying Colors..... | 78 |
| Matching Colors | 82 |
| Adding a Slogan..... | 83 |
| Saving the Project..... | 85 |
| Previewing | 85 |
| | |
| LESSON 3: WORKING WITH MULTIPLE OBJECTS | 88 |
| Selecting Multiple Objects | 91 |

| | |
|--|------------|
| Moving Multiple Objects | 96 |
| Editing Multiple Objects | 98 |
| Aligning Objects | 98 |
| Aligning Objects to the Page | 102 |
| Arranging Objects..... | 104 |
| Getting Rid of Leftovers..... | 113 |
| Grouping Objects | 116 |
| Pinning Objects..... | 119 |
| Distributing Objects | 120 |
| Locking Objects..... | 125 |
| LESSON 4: BUTTONS, ACTIONS AND PAGES..... | 128 |
| Adding Buttons..... | 131 |
| Matching the Width and Height..... | 137 |
| Changing Text Settings | 138 |
| Duplicating Objects | 141 |
| Lining Them Up..... | 144 |
| Adding Simple Actions..... | 145 |
| Adding Pages..... | 155 |
| Adding Navigation Buttons | 158 |
| Copying Objects..... | 164 |
| Trying It Out | 168 |
| Sending Email..... | 169 |
| LESSON 5: STATUS TEXT..... | 174 |
| Adding a Paragraph Object | 177 |
| Making the Text Dynamic | 182 |
| Adding Page Actions..... | 187 |
| LESSON 6: SCROLLING TEXT..... | 192 |
| Adding a Panel Image | 195 |

| | |
|---|------------|
| Adding a Scrollable Paragraph Object | 199 |
| LESSON 7: VIDEO | 208 |
| Adding a Panel Image | 211 |
| Adding a Text Banner..... | 213 |
| Adding a Video Object..... | 216 |
| Adding Custom Video Controls..... | 222 |
| Taking Control of the Video with Actions..... | 227 |
| LESSON 8: AUDIO | 236 |
| Changing the Default Object Sounds..... | 239 |
| Setting Object-Specific Sound Effects..... | 242 |
| Adding Background Music..... | 244 |
| Pausing the Background Audio..... | 247 |
| Loading and Playing an Audio File..... | 251 |
| LESSON 9: PUBLISHING | 260 |
| Building to a Folder | 263 |
| Building a Compressed Executable | 267 |
| Burning a CD or DVD | 271 |
| LESSON 10: SCRIPTING BASICS | 276 |
| Displaying a Message | 279 |
| Using a Variable..... | 280 |
| Adding an If Statement..... | 283 |
| Testing a Numeric Value | 288 |
| Using a For Loop..... | 291 |
| Creating Functions | 294 |
| Where to Go from Here | 300 |

| | |
|-------------------------------------|------------|
| SCRIPTING GUIDE | 302 |
| Script is Global | 306 |
| Script is Case-Sensitive..... | 306 |
| Comments..... | 307 |
| Delimiting Statements..... | 307 |
| Variable Scope..... | 309 |
| Variable Naming..... | 310 |
| Reserved Keywords | 311 |
| Types and Values..... | 311 |
| Arithmetic Operators | 319 |
| Relational Operators | 320 |
| Logical Operators..... | 321 |
| Concatenation | 321 |
| Operator Precedence | 322 |
| If | 323 |
| While..... | 324 |
| Repeat | 325 |
| For | 326 |
| Creating Tables..... | 327 |
| Accessing Table Elements | 328 |
| Numeric Arrays | 328 |
| Associative Arrays..... | 329 |
| Using For to Enumerate Tables | 330 |
| Copying Tables | 332 |
| Table Actions | 334 |
| Function Arguments | 336 |
| Returning Values..... | 337 |
| Returning Multiple Values..... | 338 |
| Redefining Functions..... | 338 |
| Putting Functions in Tables | 339 |

| | |
|---|-----|
| Concatenating Strings | 340 |
| Comparing Strings..... | 340 |
| Counting Characters | 342 |
| Finding Strings: | 342 |
| Replacing Strings: | 343 |
| Extracting Strings | 344 |
| Converting Numeric Strings into Numbers | 345 |
| Actions..... | 349 |
| Error Handling..... | 349 |
| Syntax Errors | 349 |
| Functional Errors..... | 351 |
| Debug Actions..... | 352 |
| Other Resources | 358 |



Welcome!

Introduction

AutoPlay Media Studio 7.0 is the state of the art in CD-autoplay multimedia tools. With its intuitive workflow and drag-and-drop objects, even absolute beginners can quickly achieve impressive results. But despite its world-renown ease of use, AutoPlay Media Studio is a serious development tool. In fact, it's used by thousands of people to create everything from AutoRun/AutoPlay menus and CD business cards, to fully interactive training applications. With AutoPlay, your imagination is your only limit!



What Can I Create With AutoPlay Media Studio?

Quite simply, AutoPlay Media Studio helps you make professional multimedia software. While the product excels at certain tasks like making front-end browsers for CD-ROMs (i.e. AutoRun/AutoPlay menus), that's just the beginning.

Integrating diverse media types such as images, sounds, videos, text, and flash into a single cohesive presentation is what AutoPlay Media Studio does best. In fact, it has been trusted to deliver rock-solid multimedia experiences to millions of people around the world. Users just like you have chosen AutoPlay Media Studio for:

- Multimedia Authoring & Application Development
- Computer Based Training (CBT) Applications
- CD-ROM AutoPlay/AutoRun Menu Systems
- Interactive Marketing Presentations
- CD Business Cards
- Much more

Intuitive Drag and Drop Design

AutoPlay Media Studio has always been known for its easy-to-use visual design environment. Simply drop high-level, interactive objects (images, video, text etc.) onto pages to create instant functionality. Move them into position using your mouse or with the aid of an extensive assortment of alignment tools and grids. Once you've got your interface down, it's a snap to attach powerful actions to various events such as mouse clicks and key presses. Whether it's opening a PDF file, playing a video or showing a web site, the Action wizard makes it easy to choose from the over 640 built-in functions.

About This Guide

This user's guide is intended to walk you through building a sample project: a CD business card for a fictional real estate agent named Ted Sellers. You'll learn the ins and outs of the program interface, and how to perform many common tasks.

The guide is organized into 10 lessons:

- Lesson 1:** Getting Started
- Lesson 2:** Graphics and Text
- Lesson 3:** Working with Multiple Objects
- Lesson 4:** Buttons, Actions and Pages
- Lesson 5:** Status Text
- Lesson 6:** Scrolling Text
- Lesson 7:** Video
- Lesson 8:** Audio
- Lesson 9:** Publishing
- Lesson 10:** Scripting Basics

Each lesson begins with a brief overview and a list of the things you will learn in that lesson. The lessons are divided into a number of exercises, which are broken down into individual steps. Each step appears in bolded text, with a number beside it so you don't lose your place. Additional information or explanation for each step is included in the non-bolded text that follows it.

Document Conventions

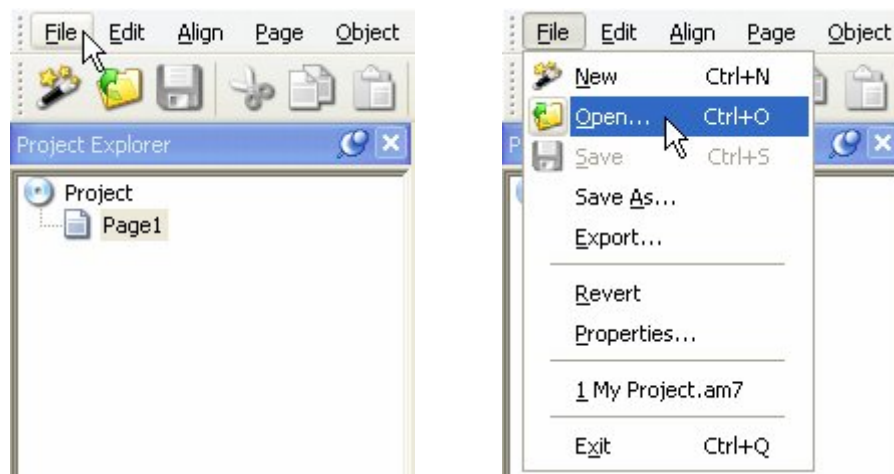
This user's guide follows some simple rules for presenting information such as keyboard shortcuts and menu commands.

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 AutoPlay program window, and then click on the Open command in the list that appears.



"File > Open" means click on the File menu, then 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 *"AutoPlay makes me happy"* into the Message setting. This means to type in "AutoPlay makes me happy", including the quotes.

AutoPlay = AutoPlay Media Studio

Throughout this user's guide, the name "AutoPlay" has been used as a short form of "AutoPlay Media Studio." Whenever I refer to "AutoPlay," I'm referring to the actual product.



Lesson 1:

Getting Started

Every journey begins with a first step. In this lesson, I'll walk you through the creation of a new project, and introduce you to the AutoPlay program interface.

1

What You'll Learn

In this lesson, you'll learn how to:

- Create a new project
- Make sure you have the latest version of AutoPlay Media Studio
- Recognize the different parts of the program interface
- Customize the workspace
- Load a pre-configured workspace layout
- Take advantage of self-help resources
- Change the project's window title
- Set the page size for the project

How Long Will It Take?

This lesson takes approximately 20 minutes to do.

Starting the Lesson

Since this is the first lesson, there isn't anything you need to do before you get started. So let's dive right in and start learning AutoPlay!

Starting a New Project

Everything has to start somewhere, and in AutoPlay, the design process starts with the creation of a new project.

A project is just the collection of files and settings and everything else that makes up an AutoPlay application. A typical project will consist of several pages, with different objects on each page. Each page, and each object on it, has individual settings that you can edit and adjust to make the application do what you want it to. These settings are all stored in a single file, called the *project file*.

This file, along with any images, videos, or other files that you add to your project, are collected inside a special folder called the *project folder*. The project folder contains everything that belongs to the project, including the project file. Each project has its own project folder, which serves as a container for the project.

Throughout this tutorial, you'll be working on a single project. This project will start out simple and then gradually become a fully working example as you learn more of AutoPlay's advanced features.

But before you can start adding cool things to your application, you need a project to work on. So let's open the AutoPlay Media Studio program and start a new blank project.

1) Open AutoPlay.

Use the Start menu to launch the AutoPlay Media Studio program.

Start > All Programs > Indigo Rose Corporation > AutoPlay Media Studio 7.0

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

The Welcome dialog appears whenever you run AutoPlay Media Studio.

It easily lets you create a new project, open an existing one, or restore the last project that you worked on. (Restoring the last project automatically opens the project you had open the last time you ran AutoPlay.)



The Welcome dialog

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

3) Explore the New Project dialog.

The New Project dialog lets you name your project and choose what kind of project you’d like to start.



The New Project dialog

In the middle of the New Project dialog, there is a scrollable list with a bunch of thumbnail images on it. Each thumbnail image represents a different project template. A project template is a ready-made “starter project” that you can use to get your own project off the ground. As you can see, the project templates come in many different styles and themes.

4) Change the project name to *Tutorial*.

You want to replace the default text “My Project” with the correct name for this project, so highlight all of the text in the Enter Project Name field and type *Tutorial*.

This will be the name of the *project folder*, which is the folder that will be used to keep all of the project’s files together, and of the *project file*, which is the file that contains all of the settings and options that make up your project. (This is the file that you will “open” whenever you want to load this project into AutoPlay.)

Every project needs a name, and that name has to be unique. You can’t give two projects the same name (and in fact AutoPlay won’t let you).

6) Click on the thumbnail for the Blank Project template.

You'll be starting the tutorial project from scratch, so select the Blank Project template.

Note: To select a template, just click on its thumbnail.

7) Click on "Create Project Now."

When you click on Create Project Now, the New Project Options dialog closes, AutoPlay sets up the project folder and project file with the name you chose, and the project is loaded into the design environment.

Note: The AutoPlay Media Studio program interface is also known as the *design environment*.

Since you selected the Blank Project template, your project consists of a single blank page, with no objects on it or anything. A clean slate for you to build on!

Tip: Once you're in the design environment, you can start a new project by choosing File > New.

8) Maximize the AutoPlay program window.

The easiest way to work with AutoPlay is with the program window *maximized* so it covers the whole screen. This way, you have the whole desktop area to work with, and you won't have any other programs or windows in the background to distract you.

To maximize the window, click on the little Maximize button, which is the second button from the right on the AutoPlay title bar (right next to the Close button).



If the maximize button looks like this:



...then you already have the program window maximized. (That button is actually the Restore button, which takes the place of the Maximize button while the window is

maximized. If you click the Restore button, the window will return to the size and position it had before you maximized it.)

Making Sure You Have the Latest Version

AutoPlay has the built-in ability to check the Internet to see if there is an update available. 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 wizard will open. TrueUpdate is an Internet update technology developed by Indigo Rose Software that makes it easy to add automatic updating capabilities to any piece of software.

2) Click Next.

When you click Next, the TrueUpdate wizard will connect to the Indigo Rose website and determine whether a newer version of AutoPlay is available for you to download.

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. Otherwise, click Finish to exit the TrueUpdate wizard.

Learning the Interface

Now that you have AutoPlay started and you've made sure that you're using the latest version, it's time to get comfortable with the program interface itself.

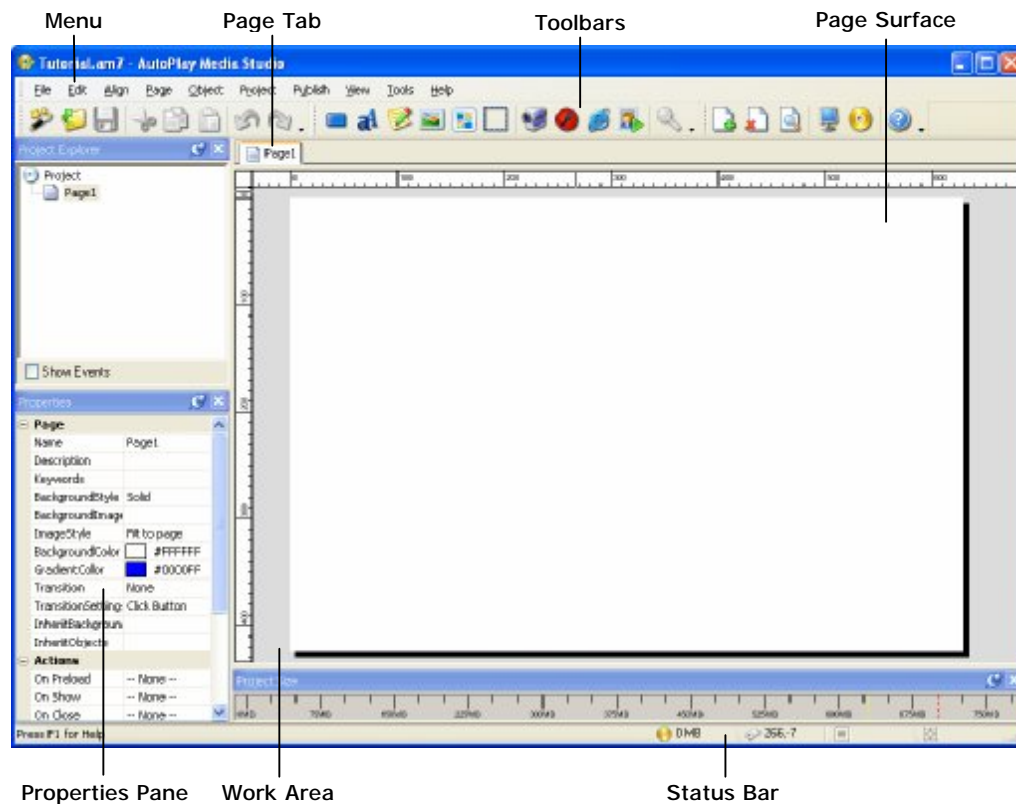
1) Explore the AutoPlay program window.

The AutoPlay 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.



In the middle of the program window, the surface of the current page is visible in the work area, measured in pixels by a pair of rulers.

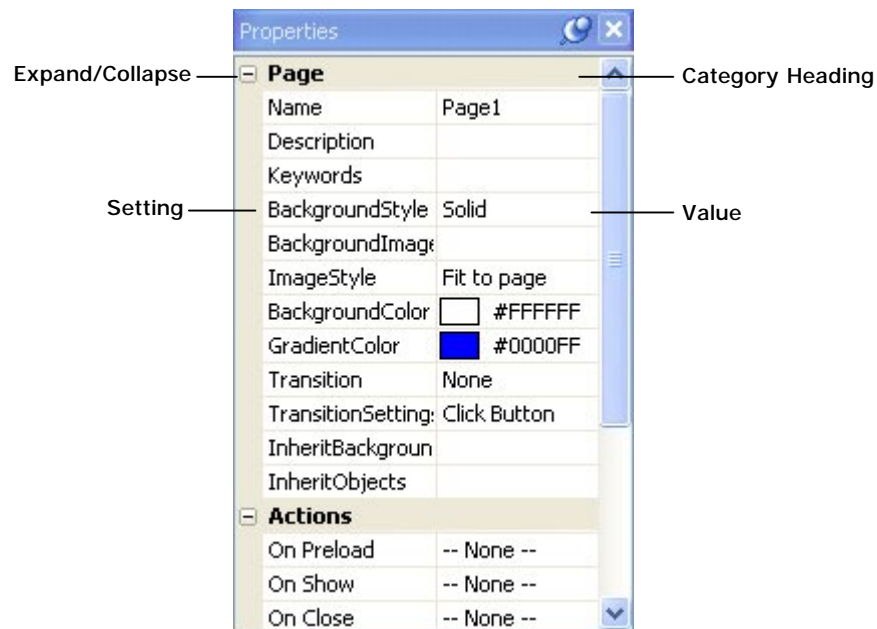
At the very bottom of the window, a status bar reflects your interaction with the program and offers a number of informative readouts.

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.

Tip: To show/hide panes, use the View menu (View > Panes).

One very useful pane is the properties pane. By default, it's located to the left of the work area, along the left side of the screen. This is where you can see and edit the settings for the currently selected object or page.

The settings on the properties pane are organized into categories. You can expand or collapse these categories by double-clicking on the category heading, or by clicking on the little + or - symbols that appear in the column to the left.



Properties pane

2) Close the properties pane.

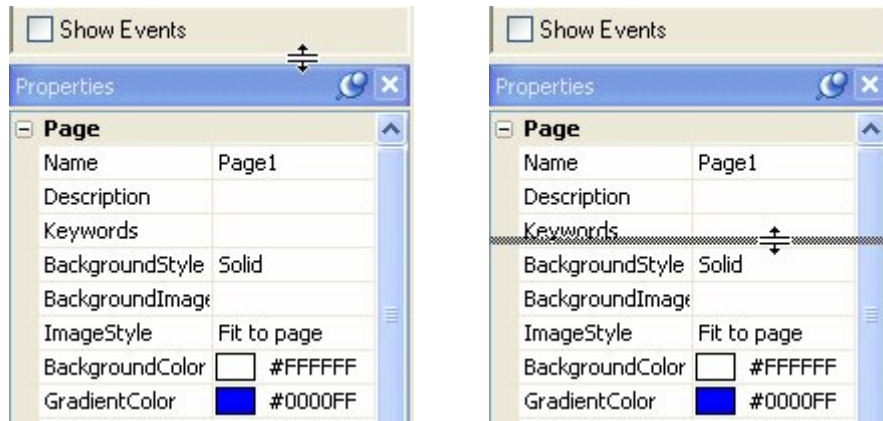
You can close a pane by clicking on the little x on its title bar.

3) Choose View > Panes > Properties to open the properties pane again.

All of the panes can be toggled on or off in the View menu. When you choose View > Panes > Properties, the properties pane is restored to the same position it occupied before you closed it.

4) Make the properties pane smaller by dragging its top edge down.

You can resize the panes by dragging their edges. In this case, you want to drag the part “between” the two panes...the little bit of pane above the properties pane and below the project explorer 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.

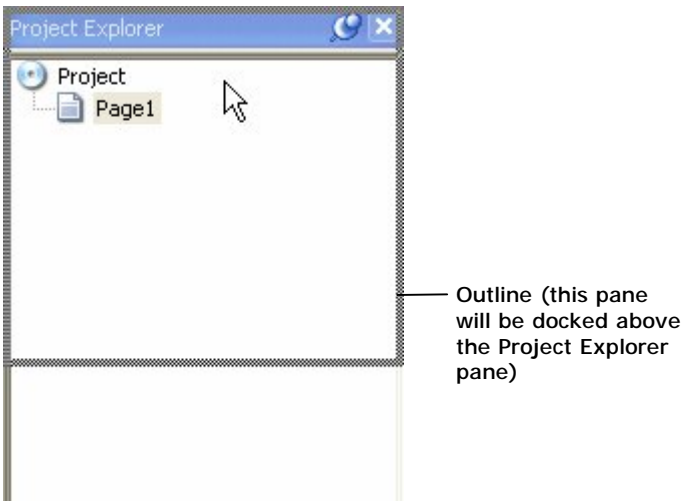


Resizing the properties pane

When panes are docked alongside each other, dragging an edge will resize both panes at the same time—making one taller and the other shorter, for instance. As a matter of fact, when you make the properties pane smaller, you will be making the project pane taller at the same time.

5) Move some of the panes around. Try moving panes to the edge of the screen to dock them, or drag them on top of other panes to combine them in different ways.

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 pane will end up. If you drag the pane near the edge of the screen, or near another pane, the outline will “snap” to show you how the pane can be docked, tabbed, or otherwise combined with the target area.

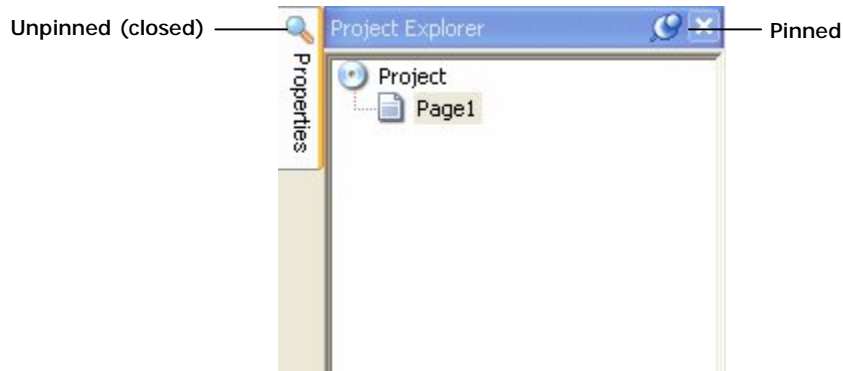


Docking a pane

Note that when panes are tabbed, dragging the title bar moves all of the tabbed panes at once. You can “tear” a tab away by dragging it away from the others. You can also reorder the tabs by dragging them left and right.

Tip: When you’re dragging panes, it’s the position of the mouse cursor that determines how the pane’s outline snaps into place. For example, to dock a pane below another one, drag the pane so the cursor is near the bottom edge of that pane. To “tab” one pane with another, drag the pane so the cursor is on top of the other pane’s title bar.

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.



Pinned and unpinned panes

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.

Note: Tabbed panes are pinned or unpinned together. If you unpin a pane that is tabbed, all of the other panes that are tabbed with it are unpinned too.

6) Choose View > Layouts > AutoPlay Menu Studio 3.0.

AutoPlay comes with a number of pre-configured workspace layouts that you can access from the View menu. The “AutoPlay Menu Studio 3.0” models the design environment after one of AutoPlay's predecessors.

Tip: You can create your own workspace layouts by arranging the panes to your liking and then choosing View > Panes > Save Layout. Any layouts you save are added to the list in the View menu, and can be selected just like any of the pre-configured layouts.

7) Choose View > Layouts > Restore Default.

This returns the design environment to the default workspace layout, which is the one you will use throughout this tutorial.

Tip: Feel free to make the properties pane a bit wider by dragging their inside edges towards the page surface a little.

8) Choose View > Grid to turn the grid on.

A layout grid appears on the page surface. This grid can be helpful when you want to line up objects visually on the page.

Tip: You can make objects “snap” to the grid as you move them by choosing View > Snap to Grid.

9) Choose View > Grid to turn the grid off.

Choosing View > Grid again toggles the grid back off.

Getting Help

If you still have any questions after you complete all the lessons in 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.

The online help is only a key press away! AutoPlay Media Studio comes with an extensive online program reference with information on every object, action, and feature in the program.

In fact, whenever possible, pressing F1 will actually 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 object.

Note: You can also access the online help system by choosing Help > AutoPlay Media Studio Help.

There are three ways to navigate the online help system: you can find the appropriate topic using the table of contents, or with the help of the keyword index, or by searching through the entire help system for a specific word or phrase.

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

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

3) Choose Help > User Forums.

AutoPlay Media Studio is used by thousands of users worldwide, many of whom 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.

4) Close your web browser and return to the AutoPlay design environment.

Exit from your web browser and switch back to the AutoPlay Media Studio program.

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

5) Choose Help > Technical Support > Support Options.

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

This is also where you can find information about ordering one of our premium support packages and submitting a support request.

6) Close your web browser and return to the AutoPlay design environment.

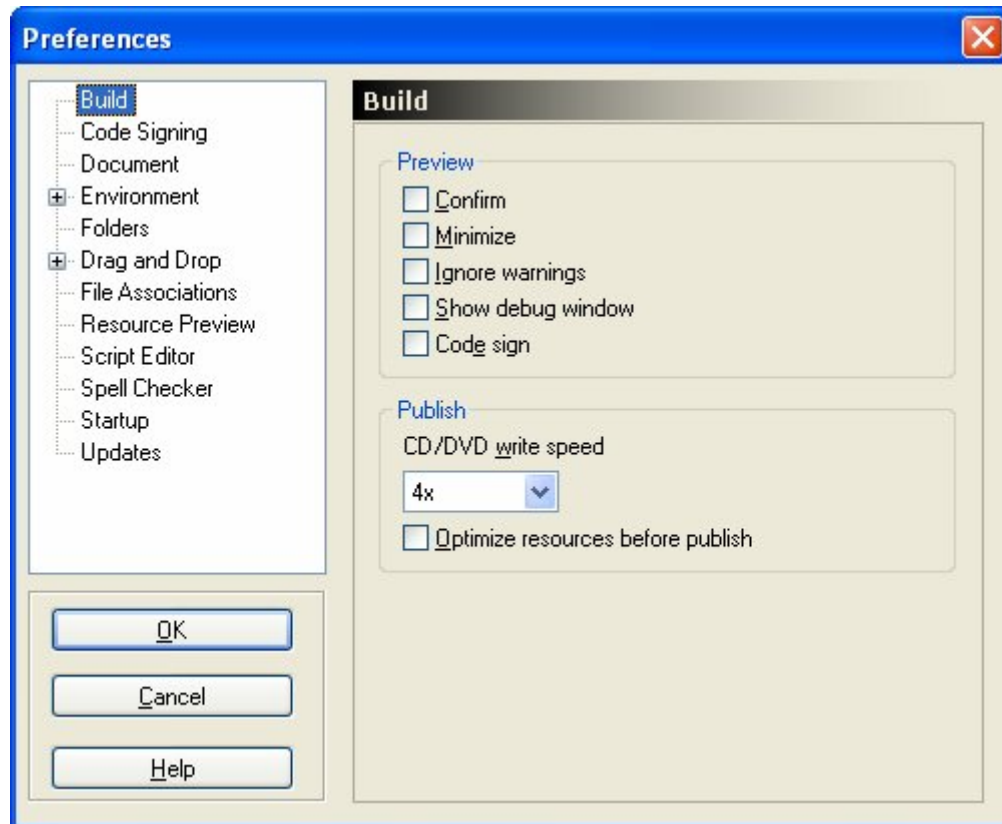
When you're done browsing the technical support information, return to the AutoPlay design environment to continue with the lesson.

Setting Preferences

There are a number of preferences that you can configure to adjust the AutoPlay 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 AutoPlay's preferences can be found.



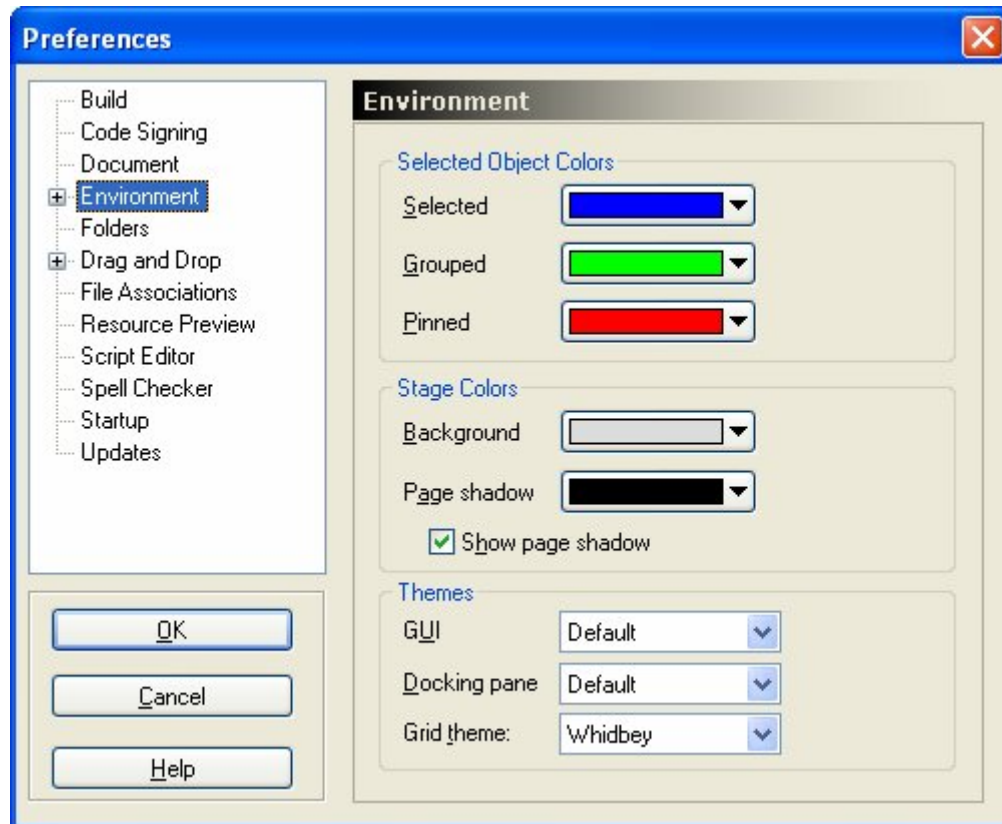
Build preferences

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.

Tip: If this is the first time you've opened the Preferences dialog, the Build category will already be selected. One of the Preview options in this category is worth noting. The Show debug window option allows you to display debug information in a separate window whenever you preview the project. Turning this option on can help you see what's going on "under the hood" during a preview, which can be helpful if you run into any problems while you're testing your project.

2) Click on the Environment category.

The Environment preferences allow you to change the way the design environment looks. For example, you can change the colors that are used when objects are selected, or change the background color of the entire work area. You can even change the color of the “shadow” that appears behind the page surface (or turn it off completely).

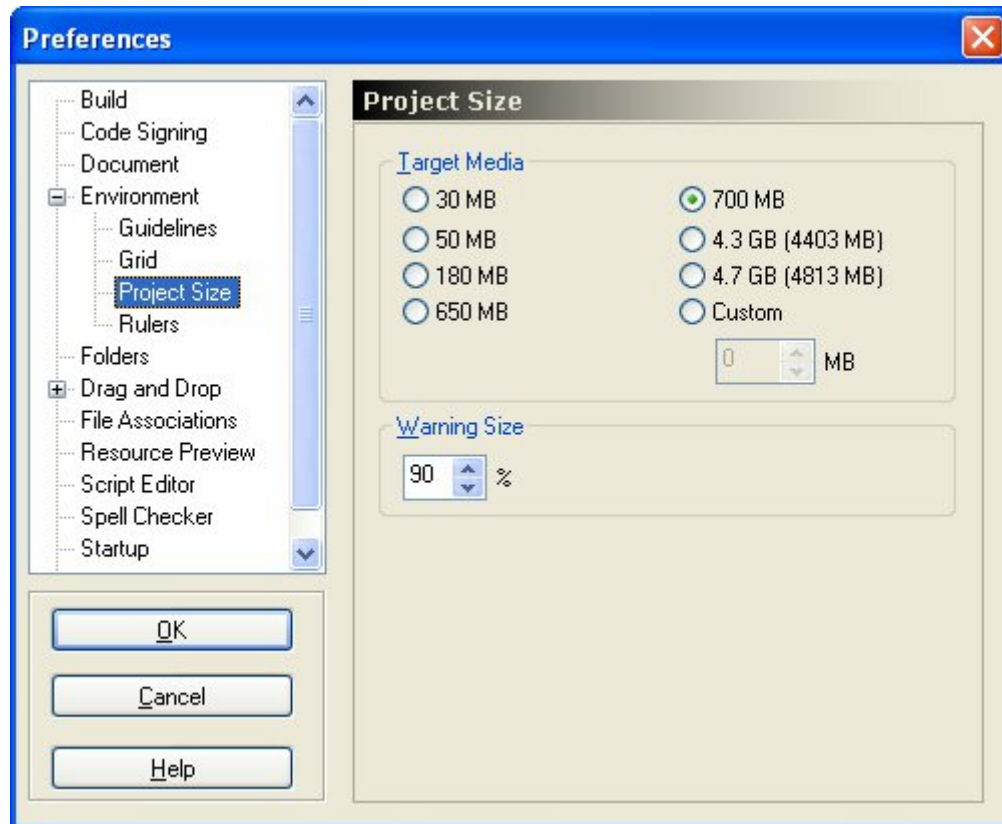


Environment preferences

Tip: Turning off the page shadow can make it easier to visually line objects up with the bottom and right edges of the page.

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

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



Project Size preferences

The Project Size preferences are where you can set the target media size, which determines the scale of the meter on the Project Size pane.



Project Size pane

If you set this target media size to match the size of the media you're using, you'll have a visual indicator of how much space is left in your project as the project grows in size.

Note that this preference doesn't affect the project in any way; the target media size only affects the size meter on the Project Size pane. It also doesn't matter how big the disc in your burner is...if you want the size meter to match the size of the media you're using, you'll need to change the target media size manually whenever you switch from one media size to another.

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 the number of undo levels for the program (in the Document category) and what happens when you drag a file onto the page (in the Drag and Drop 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.

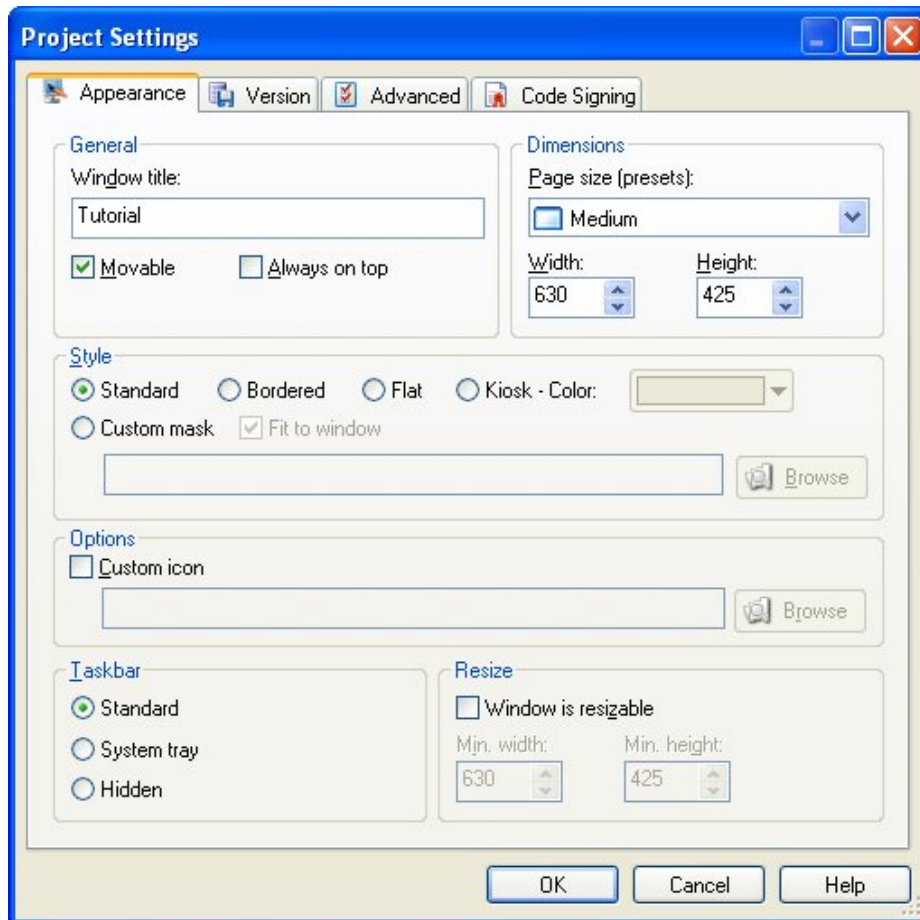
Modifying the Project Settings

The project settings affect the appearance and behavior of your application at run time.

Note: Each project has a number of settings that can be configured on a per-project basis. These project-wide settings can be accessed from the Project menu.

1) Choose Project > Settings.

This opens the Project Settings dialog.



Project Settings dialog

2) Change the Window Title to *Ted Sellers - An Agent You Can Trust*.

By default, the window title is the same as the name of the project. This project is going to be a CD business card for a (fake) real estate agent named Ted Sellers, so let's make the title something more appropriate.

To change the window title, replace the text in the Window Title field with *Ted Sellers - An Agent You Can Trust*.

Note: This text will appear in the application's title bar when the user runs your application.

3) Make sure the page size is set to “Medium.”

Setting the page size to “Medium” sets the Width and Height settings to 630 and 425, respectively. This is a good “safe” size for an application, since it will fit on every common Windows screen resolution from 640x480 and up.

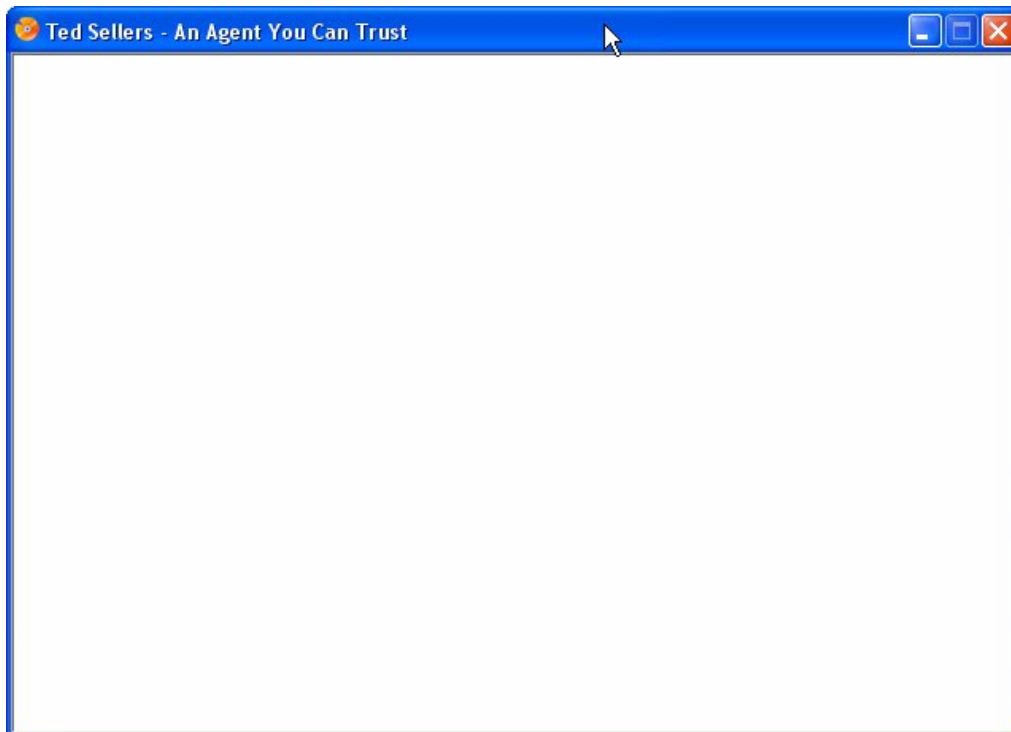
Note: This sets the width and height for all of the pages in your project.

4) Click OK to close the Application Settings dialog.

The rest of the settings are fine, so click OK to close the Application Settings dialog and return to the main program window.

5) Choose Publish > Preview.

AutoPlay’s preview feature lets you take a “sneak peek” at your application, without actually building or burning the project. At this point, there isn’t much to see, since the project still consists of a single blank page. But the text you entered for the window title does appear in the title bar, and if you could measure the page, you’d find that it is indeed 630 pixels wide by 425 pixels tall.



6) Click the Close button on the title bar to exit the preview.

When you exit the preview, you're returned to the AutoPlay program window.

7) Choose File > Save.

When you save the project, all of the changes that you've made to it are stored in the project file. You can open that project file at any time to continue working on the project.

Lesson 1 Summary

In this lesson, you learned how to:

- Create a new project
- Make sure you have the latest version of AutoPlay Media Studio
- Recognize the different parts of the program interface
- Customize the workspace
- Load a pre-configured workspace layout
- Take advantage of self-help resources
- Change the project's window title
- Set the page size for the project

Lesson 2:

Graphics and Text

This lesson will introduce you to *objects*, the basic building blocks that make every AutoPlay application unique. The two easiest objects to use are images and labels. Together, they let you populate your project with graphics and text. Although they're simple to use, they allow you to do some very advanced things, and by the end of this lesson, you'll be well on your way to manipulating these objects like a pro.

Most of the skills you'll learn in this lesson can be applied to all of the objects you'll use in AutoPlay, so pay attention—this is very useful information. For instance, all of the objects in AutoPlay can be resized and repositioned freely, and in this lesson you'll learn how to do that. You'll also learn how to rename objects, and how to quickly create a clone of an existing object by duplicating it...a real time saver that you'll find yourself using time and time again.

2

What You'll Learn

In this lesson, you'll learn how to:

- Change the page background
- Add an image object
- Resize objects
- Add label objects
- Duplicate objects
- Change the text in a label object
- Rename objects
- Use custom font settings
- Use different text colors
- Copy a color from one object to another
- Match an object's normal, highlight and click colors
- Save the project
- Preview the project

How Long Will It Take?

This lesson takes approximately 45 minutes to do.

Starting the Lesson

If you're continuing from Lesson 1, you should still have AutoPlay running with the Tutorial project open. If so, you're ready to move on to the next exercise: Choosing a Page Background.

Otherwise, you'll need to open the project file that you saved at the end of Lesson 1.

1) Open the Tutorial.am7 file that you saved in Lesson 1.

When you save a project, AutoPlay automatically creates a project folder for it inside your "My Documents\AutoPlay Media Studio 7.0\Projects" folder. This project folder is where everything that belongs to the project is stored—including the project file, which contains all of the settings used in the project.

The project folder and the project file always have the same name that you gave to the project when it was created. The project file's name ends with a ".am7" file extension.

Since you named this project "Tutorial" in Lesson 1, the name of the project folder will be Tutorial, and the name of the project file will be Tutorial.am7. So, the path to the project file should be something like:

```
..\My Documents\AutoPlay Media Studio 7.0\Projects\Tutorial\Tutorial.am7
```

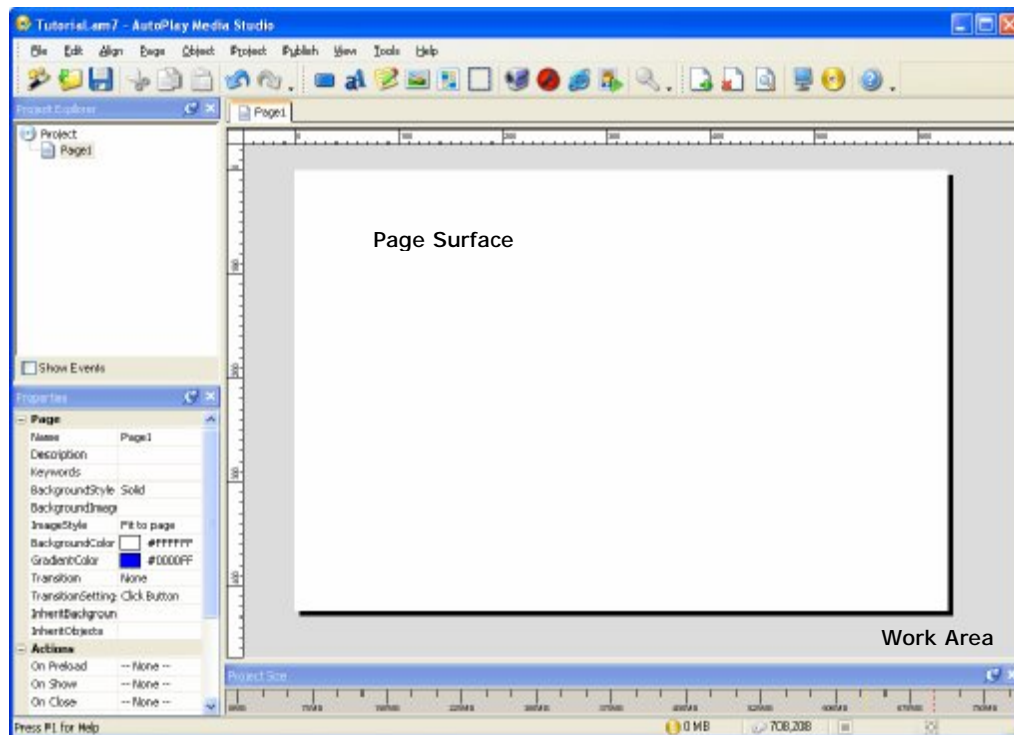
To open the project, you just need to open that project file.

Choosing a Page Background

Each project is made up of one or more *pages*. A page in AutoPlay is very much like a page in a book. Just like in a book, each page can have different items on it, such as different photographs and text.

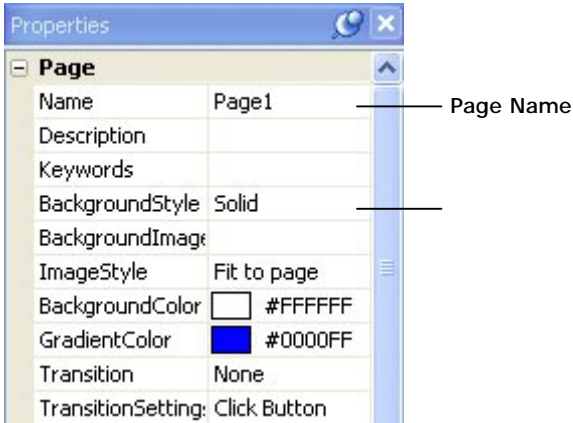
Of course, in AutoPlay, there's a lot more that you can put on a page, from text and graphics to interactive buttons and full-motion video. (You'll see the different kinds of objects you can put on a page as you go through the lessons in this user's guide.) But for now, let's just focus on the page surface itself.

1) Click on the page surface.



The page surface is shown in the middle of the work area. (It's the big white rectangle.)

When you click on the page surface, the page's settings are displayed in the properties pane.



The properties pane

In the properties pane, you can see that this page is named Page1, and that its background style is set to Solid.

Tip: You can click on the page surface at any time to “select” the page and see its settings in the properties pane.

Since a big white rectangle is a bit boring for this project, let’s make the page more interesting to look at.

2) In the properties pane, set the Background Color to Pale Blue (#99CCFF).

The surface of the page is also called the page *background*. When you start a blank project, it has a single, blank page, and the background color is solid white.

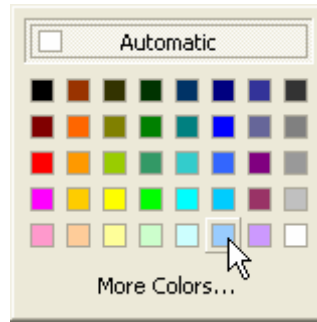
To set the background color to Pale Blue, click on the BackgroundColor setting in the properties pane, and then click the select button to bring up a color chooser.



Click the Select button to choose a background color

On the color chooser, hold your mouse over one of the color squares until the name of the color appears. (If you hold your mouse over a color square for a second or two, the name of the color will appear in a tooltip.) Now move your mouse around and look at the names of the other colors. Try to find the color named Pale Blue. (It should be the

third color from the right on the bottom row.) Click on the color named Pale Blue to make it the new background color.



You can also set the color by editing the hexadecimal value directly. A quick way to do this is to double-click on the name of the text field (in this case, the “BackgroundColor” part itself), type in `#99ccff` and press Enter.

Note: Double-clicking on the name of a text field automatically highlights all of the text in the field for you. Whenever text is highlighted, anything you type instantly replaces the highlighted text.

Tip: When typing in hexadecimal color values, you don’t have to use capital letters. AutoPlay will automatically capitalize the letters for you.

Hexadecimal...?

Hexadecimal color values aren't as complicated as they look. They're just a compact way of describing colors.

Each color is made up of three values: one for the amount of red in the color, one for the amount of green, and one for the amount of blue. (Computers make different colors by mixing red, green and blue light together on the screen.)

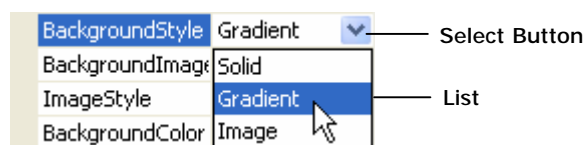
If you've used any graphics tools like PhotoShop (or even MS Paint, for that matter), you're probably used to setting the RGB value for a color using numbers between 0 and 255. The hexadecimal color #99CCFF is just a compact way of saying: "set Red to 153, Green to 204, and Blue to 255."

Each pair of digits in the color represents either red, green or blue, in this pattern: #RRGGBB. (Two digits for red, then two digits for green, and then two digits for blue.) So what do the letters like FF and CC mean? Well, that's just hexadecimal, which is a compact way of writing numbers. Hexadecimal (or "hex" for short) extends the regular decimal numbers (0-9) with six extra digits (A-F). So, if you were counting to 18 in hexadecimal, you would go: "0...9, A, B, C, D, E, F, 10,11,12." Understanding hexadecimal is a bit technical, but all you need to know to make hexadecimal color values is that letters are bigger than numbers. ("A0" is bigger than "99", "DB" is bigger than "D7", etc.) In #RRGGBB notation, bigger values mean "more of that color." FF is the biggest value you can have, so #FF0000 means "100% red, 0% green, 0% blue." #FFFFFF is white (100% red, green and blue), and #000000 is black (no red, green or blue light at all).

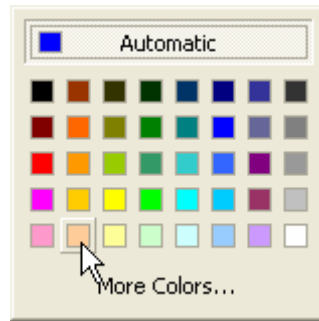
As for the number sign (#) in front, it simply indicates that hexadecimal numbers are being used—kind of like how a dollar sign (\$) lets you know that \$5.24 means five dollars and 24 cents.

3) Change the background style from "Solid Color" to "Gradient," and set the gradient color to Tan (#FFCC99).

To change the page's background style, click on the BackgroundStyle setting in the properties pane, then click the select button and choose "Gradient" from the list.



To change the gradient color, click on the GradientColor setting, and then click the select button to bring up the color chooser. Find the color square named Tan—it should be the second color from the left on the bottom row—and click on it.



The page background should now be a *gradient* between the background color at the top (Pale Blue), and the gradient color at the bottom (Tan). A gradient is just a smooth blend from one color to another. So, the page surface blends smoothly from Pale Blue at the top, to Tan at the bottom. (I like to call this “*Desert Sunrise*.”)

Using a gradient is an easy way to create a simple, attractive background for the page.

4) Change the background style from “Gradient” to “Image.”

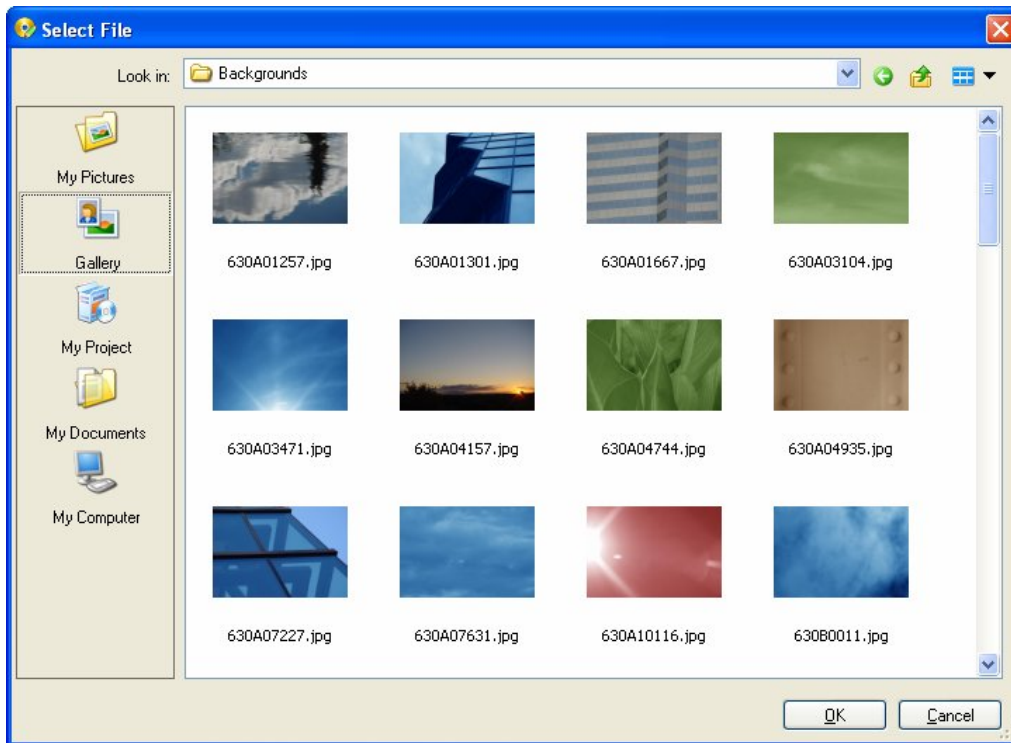
Setting the background style to “Image” lets you specify an image for the page background. This does exactly what it sounds like: the image is stretched over the page surface, and any objects that you put on the page will appear on top of the image. (We’ll select a background image in a moment.)

Note that the background color changes from a gradient to a single color, specifically the “BackgroundColor” setting. This is the color that any uncovered parts of the page will have if the image style is set to “Actual size” and the image is smaller than page.

5) Click on the BackgroundImage setting, and then click the browse button. When the Select File dialog opens, click the Gallery button.

Selecting an image is done using the Select File dialog. To open the Select File dialog, click on the BackgroundImage setting, and then click the browse button.

When you click this button, a Select File dialog opens so you can browse for a file. Clicking the Gallery button on the left side of this dialog gives you access to the complete library of files that came with AutoPlay.



Click the Gallery button

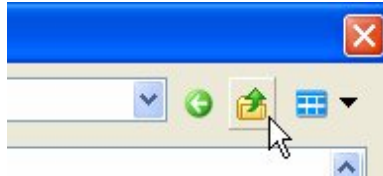
Note: There are five buttons on the left side of the Select File dialog: My Pictures, Gallery, My Project, My Documents, and My Computer. The My Project button is for files that you've already added to your project; the Gallery button is where you'll find the files that came with AutoPlay; and the other three buttons let you bring in files from other locations on your computer.

6) Try navigating in the Select File dialog. When you find an image that you like, click OK to select it as the background image.

Because AutoPlay knows that you're looking for a page background, when you click the Gallery button on the Select File dialog, it automatically takes you inside the Backgrounds folder.

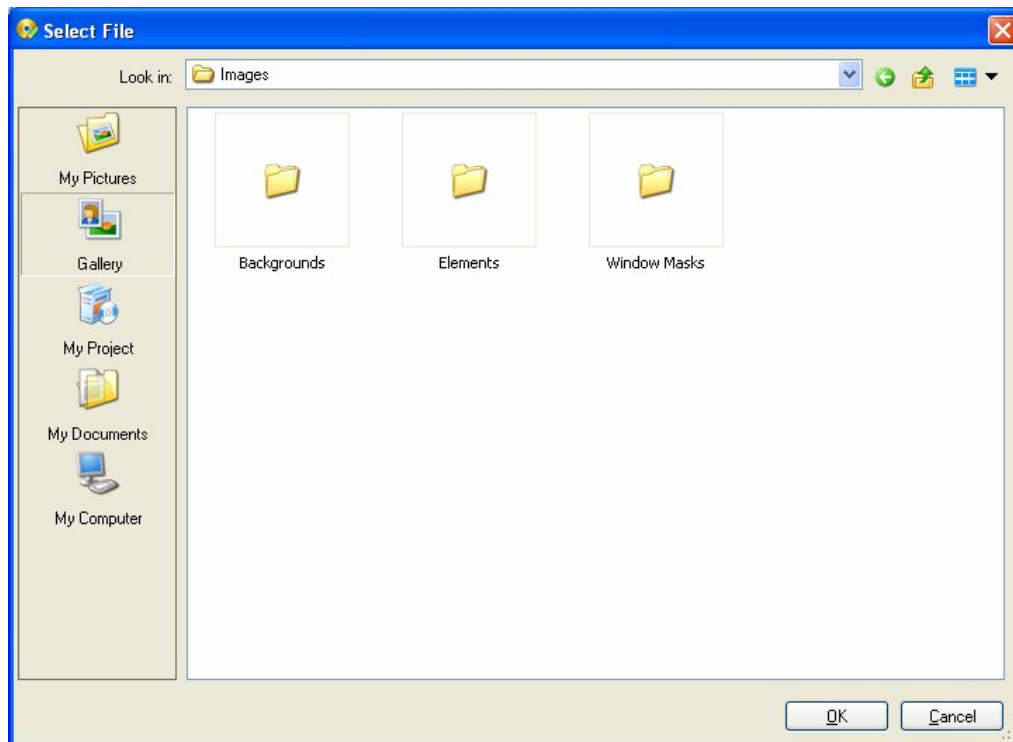
Although you will find many nice backgrounds in that folder, you can navigate to another folder in the Select File dialog if you want.

Navigating in the Select File dialog is just like browsing for files in Windows. To move back “up” or “out” of a folder, click the *move up* button at the top of the dialog.



The Move Up button

If you moved up to the parent “Images” folder, you would see the Backgrounds folder, along with two other folders in the AutoPlay image gallery.



The Backgrounds folder and its siblings as seen from their parent folder

To reach the files in the Backgrounds folder again, just double-click on it. When you double-click on a folder, you essentially go “into” that folder, revealing the folder’s contents (which can include files or even other folders).

As you can see, the gallery is full of useful images, including several high-quality abstract images that make excellent backgrounds. (Which is great for those of us whose artistic talent goes about as far as “*Desert Sunrise*.”)

Once you find an image that you like, click OK to select it as the background image. The Select File dialog will close, and the image will appear on the page as the background.

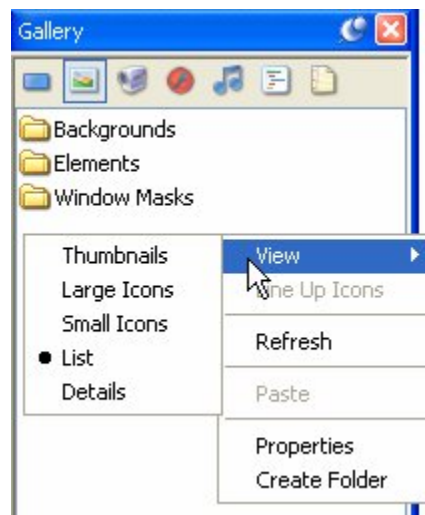
Note: The image you select automatically gets copied to your project’s Images folder.

7) Choose View > Panes > Gallery Browser, and then choose View > Panes > Resource Preview.

In the default window layout, the Gallery and Preview panes are not displayed. To show the Gallery pane, choose View > Panes > Gallery Browser. (There will be a check mark next to Gallery Browser in the View > Panes submenu when the Gallery pane is visible.) Next, make the Preview pane visible by choosing View > Panes > Resource Preview.

8) Switch the Gallery pane to use the List view.

As in Windows folders, the gallery pane has a number of different “views” you can use to look at the files it contains. You can switch views by right-clicking on an empty spot within the pane, and choosing a different item from the View menu.



Different view options

Most of the screenshots throughout this user's guide were taken using the "List" view. AutoPlay Media Studio uses the "Thumbnail" view by default.

Don't worry if your panes don't look like the ones in this user's guide—you can switch to the "List" view everywhere if you'd like, but the other views provide the same basic functionality. Which view setting you use is entirely up to you.

9) On the Gallery pane, click on the Images button, and navigate into the Backgrounds folder.

The Gallery pane lets you browse the library of files that came with AutoPlay. This library is organized into different sections, each one accessible with a button at the top of the pane. Make sure you're viewing the Images section of the Gallery pane; if not, click on the Images button at the top to switch to it.



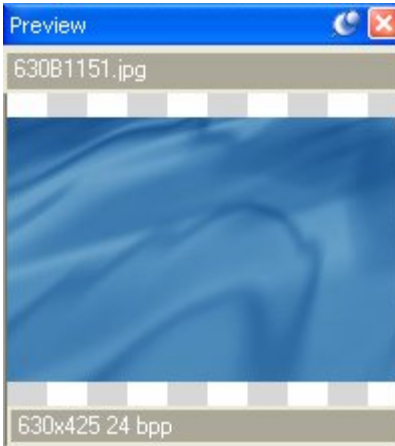
Click on the Images button at the top of the Gallery pane to browse for images

Navigating in the Gallery pane is just like navigating in the Select File dialog, or in Windows. To reach the files inside the Backgrounds folder, just double-click on it.

The Gallery pane gives you another convenient way to change the page background. Let's use it to change the background to a different image.

10) Locate the file named 630B1151.jpg, and drag it onto the page surface. When asked if you want to set it as your page background, click Yes.

If the Gallery pane and the Preview pane are both open at the same time, anything you select in the Gallery pane will automatically be previewed in the Preview pane. So, when you select 630B1151.jpg, the Preview pane gives you a sneak peak at the file.

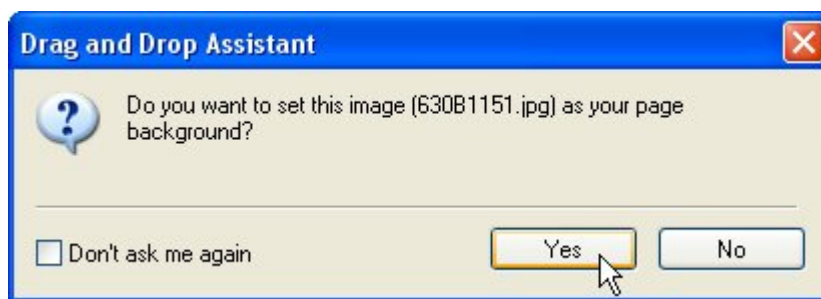


630B1151.jpg in the Preview pane

To make the 630B1151.jpg file the background image for the page, you can drag it from the Gallery pane onto the page surface.

Note: To drag the file, click on it with the mouse, and, *keeping the left mouse button held down*, move the mouse so the mouse pointer is over the page surface. (A little + symbol will appear next to the mouse pointer to show that you're dragging something.) When you let go of the left mouse button, the file will be “dropped” onto whatever is under the mouse pointer...in this case, the page surface.

Whenever you drag and drop an image that is larger than 75% of the page size, AutoPlay asks you if you want the image to be used as the page background. (An image that big would cover most of the page anyway.)



Click Yes to add the image as a page background

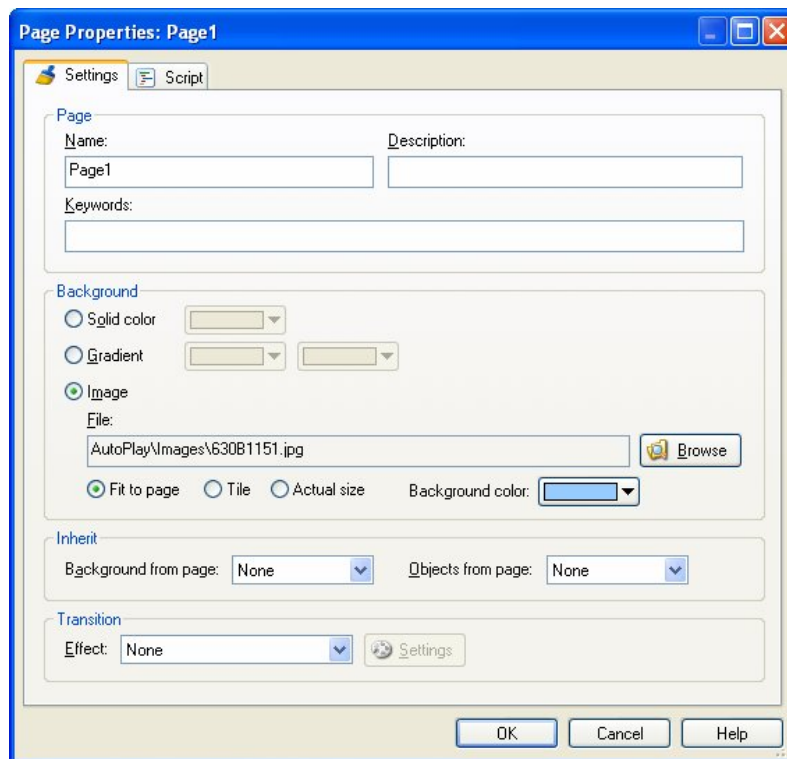
If you answer Yes, the page's Image setting will automatically be set to use that image as the background. (If you answer No, the image will be dropped onto the page as an image object.)

Tip: You can change the % size threshold and turn this feature on or off by choosing Edit > Preferences and adjusting the settings in the Drag and Drop category.

11) Double-click on the page surface to open the Page Properties dialog.

Another way to access page settings is by double-clicking on the page surface. Double-clicking on a page (or object) in AutoPlay opens a dialog with all of the settings on it.

The Page Properties dialog contains all of the settings that are available in the properties pane. As you can see, the changes that you made in the properties pane are reflected on the Page Properties dialog.



Page Properties dialog

Some people prefer double-clicking to make their changes, while others prefer to use the properties pane; the double-click dialogs are a bit easier to use in some cases, whereas the properties pane can be quicker at other times—such as when editing multiple objects at once. Feel free to use whichever method you prefer. (Most of us here at Indigo Rose use both.)

In this user's guide, I've focused more on the properties pane, since as a relatively new interface it requires a bit more explanation...the double-click dialogs are more familiar to most users and should be self-explanatory.

12) Click Cancel to close the Page Properties dialog.

Clicking the Cancel button closes the dialog and cancels any changes you've made while it was open. (In this case you didn't make any changes, so it doesn't really matter whether you click Cancel or OK, but it doesn't hurt to be careful.)

Adding Image Objects

As nice as that background is, our project will be pretty boring if that's all there is to look at. Since a picture's worth a thousand words, let's start off by adding an image object to the page.

Image "Object?"

In AutoPlay, anything that you can place on the page is called an *object*. So, when you add an image to the page, we say that you're adding an *image object*. As you'll see in a moment, each object in AutoPlay has its own settings and properties, such as the object's width, height, and position on the page.

Take a photograph for example. On its own, it's just an image of something. But in AutoPlay, the image takes on specific settings—for example, it might be the image at exactly 123 pixels wide by 300 pixels tall, positioned 200 pixels from the left edge of the page, and 100 pixels from the top. If you think of the *image* as the actual photograph itself, the *object* is the photograph along with all of its settings.

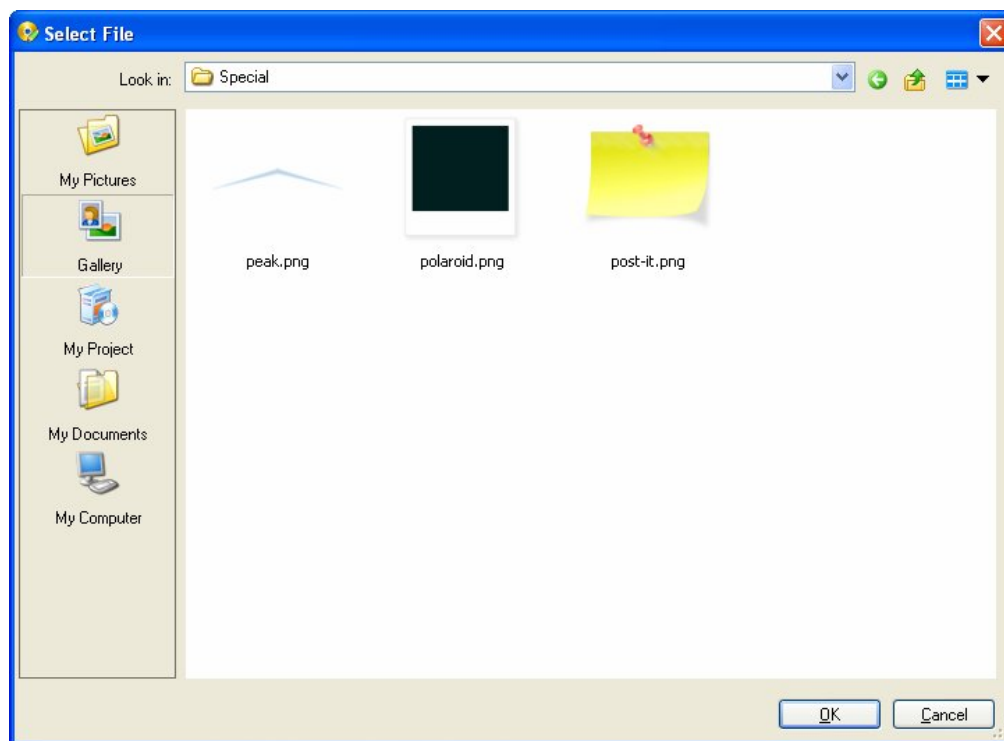
1) Choose Object > Image. In the Select File dialog, click the Gallery button, and double-click on the Special folder. Select the peak.png file and click OK.

Choosing Object > Image automatically brings up the Select File dialog so you can choose an image to display in the image object.

Tip: The Select File dialog automatically opens to the same folder you were in the last time you browsed for a particular type of file (such as an image, or a video).

Clicking the Gallery button takes you directly to the gallery folder that is most convenient for the type of file you are adding. In this case, because you're adding an image object, the Gallery button takes you to the Gallery\Images\Elements folder.

Double-clicking on the Special folder takes you to Gallery\Images\Elements\Special, which is where the file you want to add is located.



The Gallery\Images\Elements\Special folder

When you select the peak.png file and click OK, that image file is copied into your project folder's Images subfolder. The path to this newly copied file is what will appear in the File setting on the Image Properties dialog.

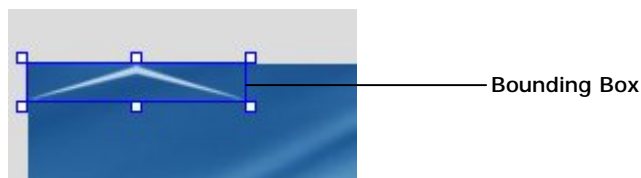
Note that the original file isn't referenced directly. When you select a file, AutoPlay makes a copy of it and puts the copy in the project folder, leaving the original untouched.

3) Click OK on the Select File dialog to finish adding the image object.

Once you click OK, the little pointed peak image is added to the page.

Note: When you add an object using the Object menu, the object is automatically positioned in the upper left corner of the page.

If you look closely, you'll see that the image is surrounded by a blue rectangle with a bunch of little white squares on it. This blue rectangle is the object's *bounding box*.

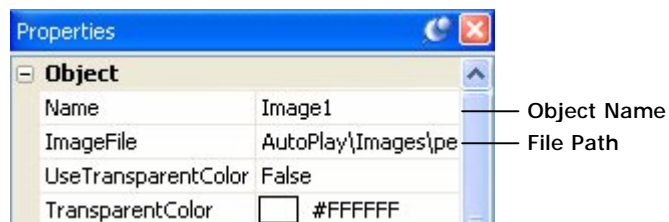


The bounding box appears whenever an object is selected. It shows you the actual size and position of the object. (As you can see, when you add a new image object, it gets selected for you automatically.)

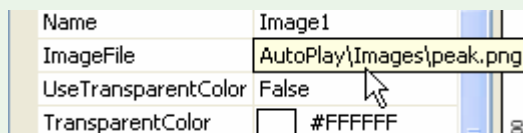
Note: The bounding box is only visible at design time, when you're working on the project.

Another thing you'll notice when an object is selected is that its settings appear in the properties pane.

If you look in the properties pane, you can see that this object is named Image1, and the path to the image file is "AutoPlay\Images\peak.png."



Tip: If you can't see all of the text in a setting, hover the mouse over the setting for a moment and a tooltip will appear to reveal the entire contents.



Does the File path seem a bit odd to you?

You're probably used to seeing paths in Windows that start with a drive letter, like "C:\Temp" or "D:\Program Files\Blah blah blah." So why doesn't this image object have that kind of path? Well, it's actually quite simple. Instead of showing you the "full" path, like you normally see in Windows, AutoPlay shows you only the part of the path that is relevant to the project. In this case, it's the part of the path inside the project's CD_ROOT folder. This makes the path shorter and easier to read. This kind of "partial path" is what's known as a *relative path*.

Feel free to take a look at some of the other properties this image object has.

4) Click on the page surface to deselect the image object.

Clicking anywhere outside the selected object's bounding box deselects the object. Basically, it just selects something else—in this case, the page.

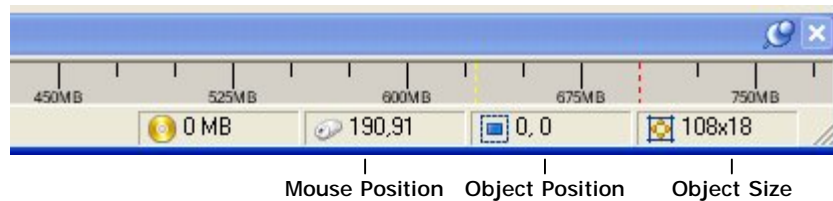
(You can tell that the page is selected because the page settings appear in the properties pane.)

5) Click on the image object to select it again.

When you select the image object, its bounding box reappears, and its settings appear in the properties pane again.

Note: To select an object, just click on it.

In addition, the object's current position and size are displayed on the status bar.



The numbers on the status bar indicate the current X,Y coordinates of the top left corner of the object's bounding box, and the current width and height of the object's bounding box, in pixels.

6) Drag the image object to the middle of the page.

Moving an object is easy: just use the mouse to drag it where you want.

Note: To drag an object, first position the mouse pointer over the object. Then, “grab onto” the object by pressing (and holding) the left mouse button. Keep holding the left mouse button down while you “drag” the object by moving the mouse.

As you start dragging the object, a dotted rectangle will appear...this rectangle represents the object, and allows you to see where the object will be placed when you release the mouse button. (The object doesn’t actually move until the *end* of the drag operation.)

Once you’ve positioned the dotted rectangle where you want the object to go, “let go” of the object by releasing the mouse button. As soon as you release the mouse button, the object moves to its new position.

As you move the object, the position readout changes on the status bar.



The numbers indicate where the top left corner of the object’s bounding box will be when you let go of the mouse button.

Tip: Here’s a neat trick: try holding the ctrl and shift keys down (together) before you start moving the image object. When the ctrl and shift keys are held down, an object can only move vertically or horizontally—whichever direction the object is moved in first. So, if you hold Ctrl+Shift and then start moving an object to the right, the object will *only* move left or right until you let go of those keys. (The same goes for moving it up or down.)

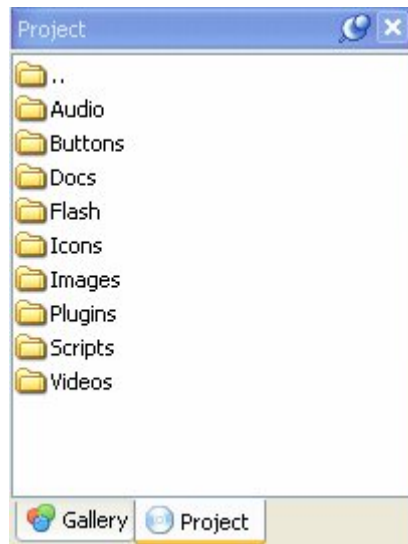
Tip: You can also move objects around using the cursor keys. (Graphic artists commonly refer to this as “nudging” the object.) This can be helpful when you want to “fine-tune” an object’s position.

7) Make sure the object is selected, and then press the Delete key to remove it.

Don’t worry, we’ll add the image object back in the next step. But I wanted to show you how to remove an object from the page.

Note that this removes the object from the page—along with all its settings—but doesn’t actually delete the image file. As you’ll see in the next step, a copy of the original image is still in your project’s Images folder.

8) Choose View > Panes > Project Browser to open the Project pane. Using the Project pane, navigate into the Images folder, locate the peak.png file, and drag it onto the page surface.



The Project pane

Once an image has been added to your project, it becomes even easier to add it a second time. This is because adding an image to your project creates a local copy of that file inside your project's Images folder.

Note: Every time you start a new project, AutoPlay automatically creates a project folder for you. Immediately inside the project folder are two things: the project file, where all the project's settings are stored; and a folder called CD_ROOT. The CD_ROOT folder is where all of your project files are organized. When you burn your project onto a compact disc, it's the contents of the CD_ROOT folder that are placed on the CD. (In other words, everything below the project's CD_ROOT folder will end up on the CD when you publish your application.)

In order to keep things neat and tidy, the files that you add to your project are stored under CD_ROOT in a folder called "AutoPlay." This folder is further divided into a number of subfolders. Images are kept in a subfolder called Images, Videos are kept in a subfolder called Videos, and so on.

Every project has a complete set of these subfolders, which you can access by using the Project pane. Most of the time, these subfolders are all you will ever see; while

you're working on the project, you're working "in" the CD_ROOT folder. (You won't see the CD_ROOT folder itself unless you look for it in Windows.)

Here's how it works: whenever you add an image to your project, it's copied into your project's Images folder first. Then the project uses that copy instead of the original. (Which is why the File setting is always a path like "AutoPlay\Images\redshoes.jpg" even if you added the file from "C:\My Documents\Graphics\Pictures of clown feet.")

This is done so you're always working with a local copy of the file—one that won't be affected by any changes you (or someone else) might later make to the original. It also means that your project won't mind in the least if the original file is ever moved or misplaced. This makes your project much more robust. (So, if someone ever comes along and decides that a nice person like you doesn't *need* all those pictures of clown feet in the My Documents folder, AutoPlay's got you covered.)

Now, most of the time you don't have to worry about this—not even one iota. You just browse for the file you want, and let AutoPlay take care of things like copying files and making your project reliable. But, it's good information to know, because it helps you understand why it's so easy to add the same image to your project a second time around.

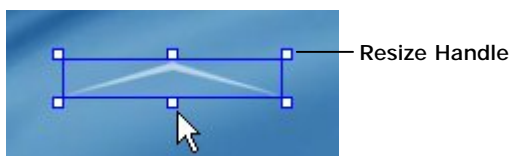
Note: If you aren't comfortable with files, folders and subfolders, just think of the Project pane as the place where you can see all the files that you've added to your project. The folders just divide the pane into different sections to keep your project files organized.

Since there are three image files that you've added to the project so far, you should see three files in the Images folder: the two images you used for the page background (including 630B1151.jpg), and peak.png.

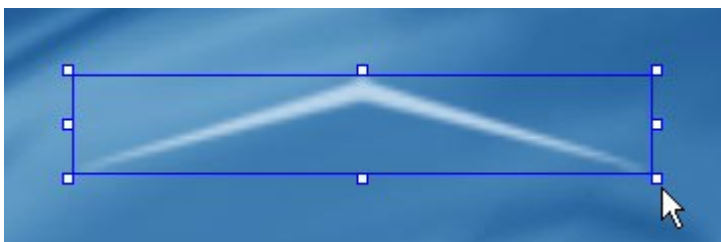
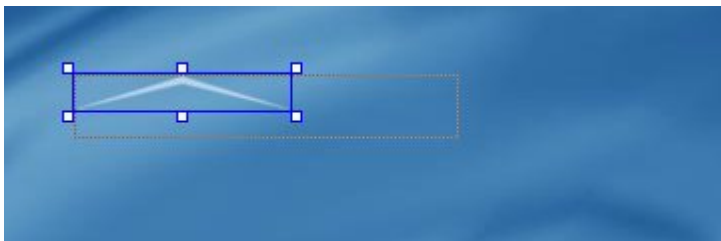
When you drag peak.png onto the page, an image object is created automatically.

9) Resize the object by dragging one of the little white boxes on the bounding box. Try holding the ctrl and shift keys down while you resize the object to maintain its current aspect ratio.

The little white boxes on an object's bounding box are called *resize handles*.



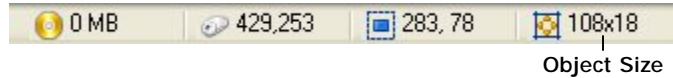
The resize handles appear on the bounding box whenever an object can be resized with the mouse. To resize the object, just drag one of the resize handles and make the bounding box bigger or smaller.



If you hold the ctrl and shift keys down (together) while you resize the object, the current aspect ratio will be maintained. What this means is that the width and height will both increase or decrease proportionally in order to keep the shape of the object the same. You can see this by watching the dotted rectangle that appears when you drag the resize handle. Normally, the dotted rectangle follows the mouse pointer exactly. When you hold the ctrl and shift keys down, however, the dotted rectangle maintains its overall shape as it increases or decreases in size.

Tip: You can also right-click on an object and choose Keep Aspect to make it always maintain its aspect ratio, even if you resize it without holding the ctrl and shift keys down.

As you resize the object, the size readout changes on the status bar.



The numbers indicate what the width and height of the object's bounding box will be when you let go of the mouse button.

Tip: You can abort a resize operation (in the middle of dragging) by pressing the Esc key.

10) Make sure the image object is selected. In the Position category of the properties pane, set the Left and Top settings to 372 and 112. Set the Width and Height settings to 108 and 18.

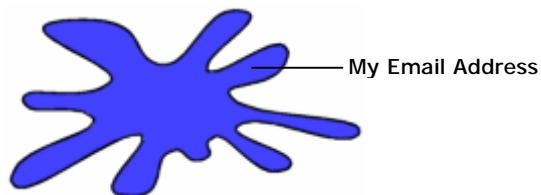
| Position | |
|----------|-----|
| Left | 372 |
| Top | 112 |
| Width | 108 |
| Height | 18 |

You can set the position and size of an object directly by typing values right into the appropriate fields on the properties pane.

Tip: You can also reset an object to its original size by right-clicking on it and choosing Restore Size.

Adding Label Objects

Some information is easier to present with words than with pictures. (For instance, it would be pretty hard to “draw” your name or email address.) Luckily, label objects make it easy to put text on the page.



1) Choose Object > Label to add a new label object. Double-click the object to display the Label Properties dialog, type in *Ted Sellers* and click OK.

Choosing Object > Label automatically adds a label object to the top-left corner of the page. Double click the new label object to bring up the Label Properties dialog so you can configure the new label object.



Label Properties

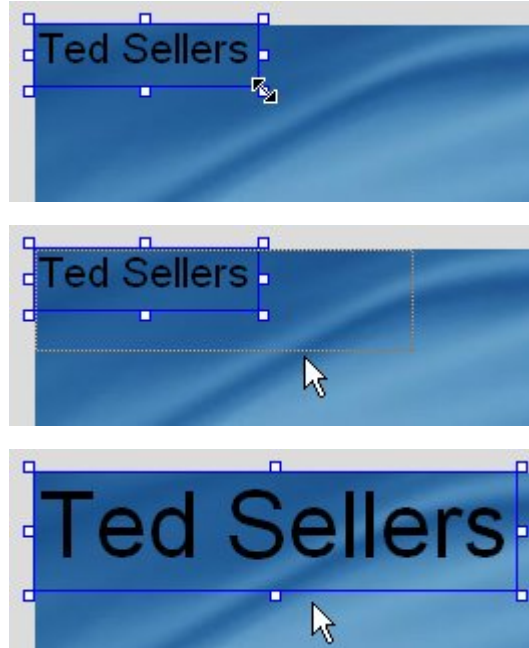
Note that the contents of the Text setting are automatically selected for you. This makes it easy for you to change the object's text right away. When you type in *Ted Sellers*, it instantly replaces the default text.



When you click OK, the label object's properties are updated.

2) Make the label object bigger by dragging one of its resize handles.

When you resize a label object, the text inside it gets bigger or smaller. In fact, resizing a label object's bounding box is just another way of setting the object's font size.



There are two things to remember about resizing label objects. First, label objects always maintain their aspect ratio as you resize them. (You probably noticed this as you were resizing the object.) This is because changing the font size changes the width and height of the text at the same time.

Second, the size of the label object may not always match the size you set it to. The bounding box might end up a tiny bit larger or smaller as the text “snaps” to the nearest font size. You won’t always be able to get a label object to fit perfectly to a specific width and height in pixels...but AutoPlay will get the text as close as it possibly can.

Tip: When resizing label objects, try not to think in terms of pixels; instead, think in terms of points, as in “a 12 point font.” Or better yet, just resize the object until it looks the way you want it to, and forget about pixels and points altogether.

3) Shrink the label object back down to its original size.

To make the label object smaller, just drag the same resize handle back to where it was before, more or less. (Don't worry about getting it exactly the same size; the point of this step is just to avoid having a big text object getting in your way during the next couple of exercises.)

4) Click the New Label Object button to add another label object. Double-click on the new object, type *REALTY LTD.* into the Text field, and click OK.

You can also add an object by clicking the New Label Object button on the toolbar.



Clicking the New Label Object button is just like choosing Object > Label from the menu. It automatically adds a new label object to the top-left corner of the page. Since you already had a label object there, the result might look a bit jumbled, with one label object on top of the other. Don't worry about that for now; we'll move the new object in the next step.

Double-click the new object to bring up its Label Properties dialog. For now, we just want to change the default text to "REALTY LTD." (without the quotes). Note that we've put a space between each letter in order to spread the letters out a bit, and we've put two spaces between the words REALTY and LTD. to make them look like separate words.

Once you click OK, the text on the object is updated.

5) Drag the new label object down so you can see it by itself.

Moving the new object over makes it easier to see what's going on.

6) Right-click on the page surface and choose Label from the right-click menu.

You can also add an object by right-clicking on the page surface and choosing the object type from the right-click menu.

Note: The right-click menu is said to be *context sensitive* in that it displays different items according to what you right-clicked on. For example, right-clicking on the page surface will present a different menu than if you right-clicked on an object.

This time, the object isn't added in the upper left corner...instead, it appears at the location where you right-clicked on the page.

Note: When you add an object with the right-click menu, it gets positioned where you right-clicked on the page.

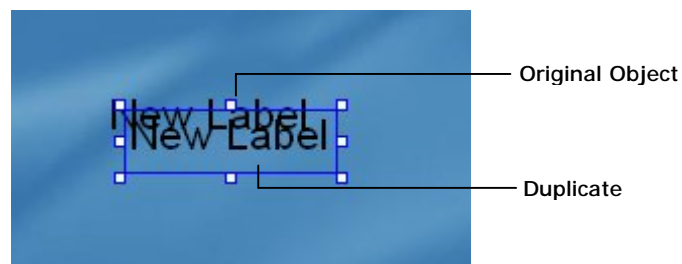
We're going to use this object to display Ted's email address, but for now the default text is okay.

Duplicating Objects

A really quick way to add an object is to make a duplicate of an existing object. Duplicating is just like copying and pasting in Windows, only it's combined into a single step.

1) Make sure the Label3 object is selected, and choose Edit > Duplicate.

When you duplicate an object, all of the original's settings are copied into the new object. The only things that change are the new object's name and position. In order to make it easier to keep track of everything, the new object is positioned a little bit down and to the right of the original. That way, you can see that the duplication was successful.



The new object is also automatically selected for you.

2) Press Ctrl+D to duplicate the new object (Label4), this time using the keyboard instead of the menu.

Duplicating objects is something you'll do pretty often, so it pays to learn the hotkey for the menu command.

Note: Most of the menu commands you can perform in AutoPlay have hotkeys assigned to them. If you look at the Edit > Duplicate command in the menu, you can see that the hotkey (Ctrl+D) is listed on the right. This makes it easy to remember what the hotkey is for duplicating an object—if you ever forget, just look in the menu.

Because the new object is automatically selected for you, you can create a “chain” of label objects very quickly by selecting an object and pressing Ctrl+D a few times.

Since the new objects start off with the same settings as the original, this is a great way to add a bunch of similar objects. Just set up the first object the way you want it, and duplicate it a few times.

Of course, you can do the same by repeatedly choosing Edit > Duplicate from the menu, but I find it much faster to press the Ctrl+D hotkey.

Changing Text

Now it’s time to put some real text in those three new label objects. First, though, let’s spread them apart a bit so they’re easier to work with.

1) Spread the three new label objects apart by dragging Label5 down, and then dragging Label4 down so it’s between Label3 and Label5.

Basically, just drag the bottom one down, and then position the middle one between them, so the objects don’t overlap any more. More or less like this:



Separating the label objects will make it easier to select them in the next few steps, and to see what’s going on. Plus, it might make them happy. (Label objects probably don’t like being crammed together any more than you or I would.)

2) Click on the Label3 object to select it. In the properties pane, change the Text setting to *ted@sellersrealty.com*.

We'll use this label object to display Ted's email address.

Note: Don't worry if you don't click on the right label object at first. You might have to click on more than one label object to find the one named Label3. (Of the three objects with "New Label" on them, it should be the one closest to the top of the page.)

To change the text that is displayed in the label object, highlight the contents of the Text setting in the properties pane, and type in the text that you want it to display.

Tip: A quick way to highlight all of the text in a setting is to double-click on the left-hand column of the properties pane.

In this case, highlight the words "New Label" and type in *ted@sellersrealty.com* instead.

Note: Whenever text is highlighted, anything you type instantly replaces the highlighted text.

As you type in the new text, the object's bounding box automatically resizes to fit.

3) Double-click on the Label4 object and type *(540) 555-3524* when the Properties dialog appears.

We'll use the Label4 object to display Ted's phone number.

4) Select the Label5 object, click on the Text setting, and then click the edit button. When the Edit Text dialog opens, delete the existing text ("New Label") and type in the following three lines:

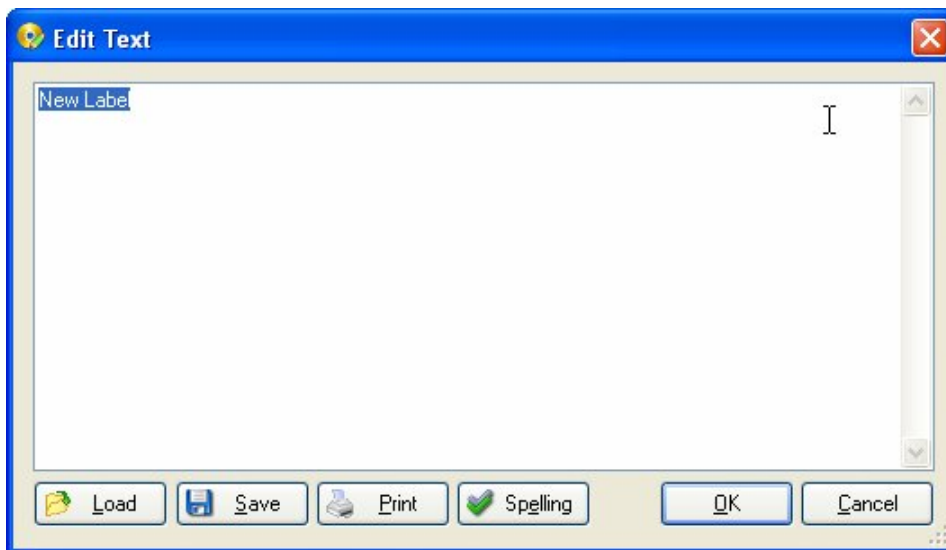
***123 Fakereal Avenue
Realfake, VA
United States 12341***

When you're done, click OK to close the Edit Text dialog.

Label objects are actually able to hold multiple lines of text. You can't just press Enter to insert a new line when you're typing stuff into the properties pane, though—there, pressing Enter means "I'm done typing, keep the changes I just made." To get around this, AutoPlay gives you access to the Edit Text dialog, where pressing Enter starts a new line, just like it would in a text editor like Notepad.



Once you click on the edit button, the Edit Text dialog will open, showing you the single line of text that the label object started out with.

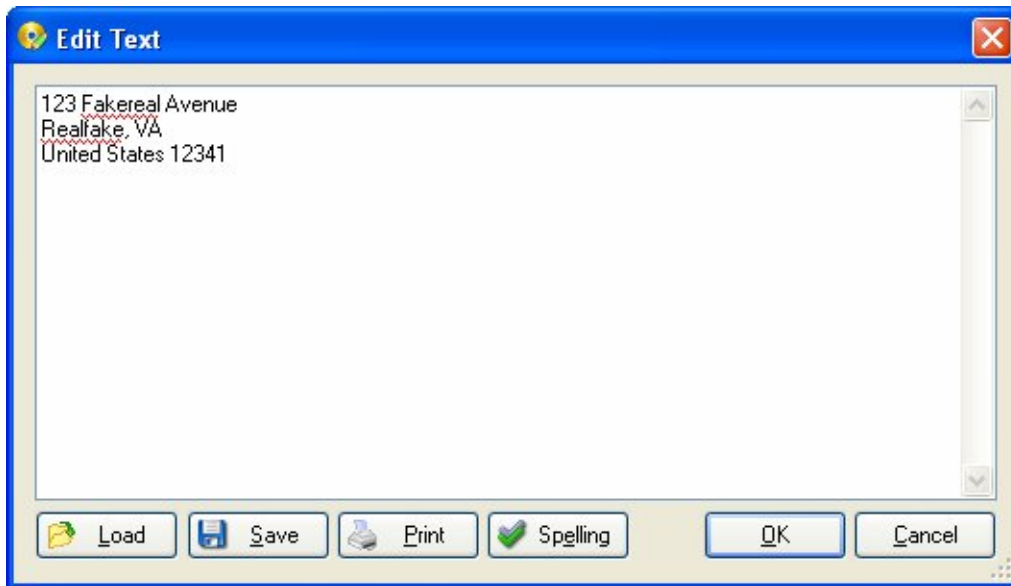


You want to replace that text with Ted's fictional mailing address, so highlight all of the text and press the Delete key to remove it.

Tip: You can also click anywhere on the text and use the keyboard to edit it, just like you would edit something in Notepad: using the cursor keys to move around, and the Backspace and Delete keys to delete one character at a time.

As you type in Ted's mailing address, make sure you press Enter at the end of each line.

Note: Pressing Enter actually inserts a *newline* into the text. A newline is a special character that basically marks the position where a new line of text should begin. The newline itself is invisible, but you can see its effects on the Edit Text dialog.

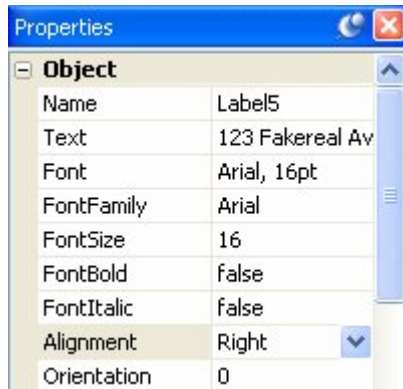


You may notice some red squiggles under the words *Fakereal* and *Realfake*. These red squiggles are the spell checker's way of alerting you that those words aren't in its dictionary. We don't care if they aren't, since the whole address is fake, but if you wanted to, you could click the spell check button to step through all such "flagged" words and choose from a number of suggested spellings.

When you're done typing the fake mailing address in, click the OK button on the Edit Text dialog. The text will be inserted into the label object, which will automatically resize to fit all of the text.

Note: If you look in the properties pane, you might notice that the Text setting actually contains the text that you typed into the Edit Text dialog. If you're wondering how AutoPlay manages to stuff 3 lines of text onto a single line in the properties pane, try clicking on the text field and then scrolling to the right with the cursor keys. As you scroll through the text, look for a backwards slash followed by the letter n, like this: `\n`. Those two letters are a special code that represents the newline character in places where you can't just press Enter to start a new line. You can actually edit the text in a label object right in the properties pane, by typing "`\n`" wherever you want a new line to appear. Of course, it's a lot easier to just edit the text with the Edit Text dialog.

5) Change the Label5 object's Alignment setting from "Left" to "Right."



When you change the Alignment setting, the object's text lines up along the right side of the bounding box.



Note: Although you can set the Alignment for any label object, the effects of changing a label object's Alignment setting will only be visible if there is more than one line of text in the object, and the lines aren't all the same length.

Tip: You can center the text in a label object, too. Just for fun, try setting the Alignment to "Center" to see how it looks.

6) Change the Label5 object's Alignment back to "Left" again.

For this project, we want the mailing address to be left-aligned.

Tip: Two other objects that support multi-line text are the paragraph object and the rich text object. In fact, paragraph and rich text objects are often a better choice when you need to display multiple lines of text on the page. Although I've chosen to use label objects in this lesson for demonstrative purposes, you could easily substitute paragraph objects or rich text objects to achieve these effects in your own projects.

Naming Objects

As you added the objects in this lesson, they were automatically given names based on the object type, along with a number to make the name unique—names like “Image1” and “Label4.”

You’ve probably already noticed the pattern: as you added more objects of the same type, the numbers got bigger. When you added a second label object, it was automatically named Label2. When you duplicated that object to create a third label object, it was named Label3. And so on.

In fact, as you add more objects of the same type, the numbers just keep increasing. The more objects you add to a page, the bigger the numbers get. (Until one day you end up with an object named Label91238912381923 and you realize you have no life.)

This is done so that every object starts out with a unique name. AutoPlay uses the object names to distinguish one object from another, so they *have* to be unique.

Actually, consider that object naming rule #1: in AutoPlay, *every object on the page has to have a unique name.*

Okay.

But that doesn’t mean you have to stick with the names that AutoPlay gives your objects. In fact, after a while, remembering which object is which can get a bit difficult if they all have names like “Image9,” “Label6” and “Label22.”

Luckily, it’s easy to change the name of an object to anything you want. To illustrate the point, let’s rename three of our label objects to make them a bit more memorable.

1) Click on the object named Label3. In the properties pane, change the Name to *Email Address*.

Since this label object contains Ted Sellers’ fake email address, it makes sense to give the object a name that reflects that.

To change the object’s name, just edit the text in the Name field so it contains “Email Address” instead of “Label3.”

Note that it’s perfectly fine to include spaces in object names. Of course, you don’t *have* to use spaces...we could have named this object “EmailAddress” or “label_email” or even “JimBob” for that matter.

Tip: For some good advice on choosing names for objects, search for “Naming Objects” in the AutoPlay help file.

2) Rename the Label4 object to *Phone* and the Label5 object to *Mailing Address*.

You can rename these other objects the same way: just click on the object, and change the text in the Name field.

Or, you can change the name by double-clicking on the object (to open the Properties dialog), clicking on the Attributes tab, and then changing the Name setting there.

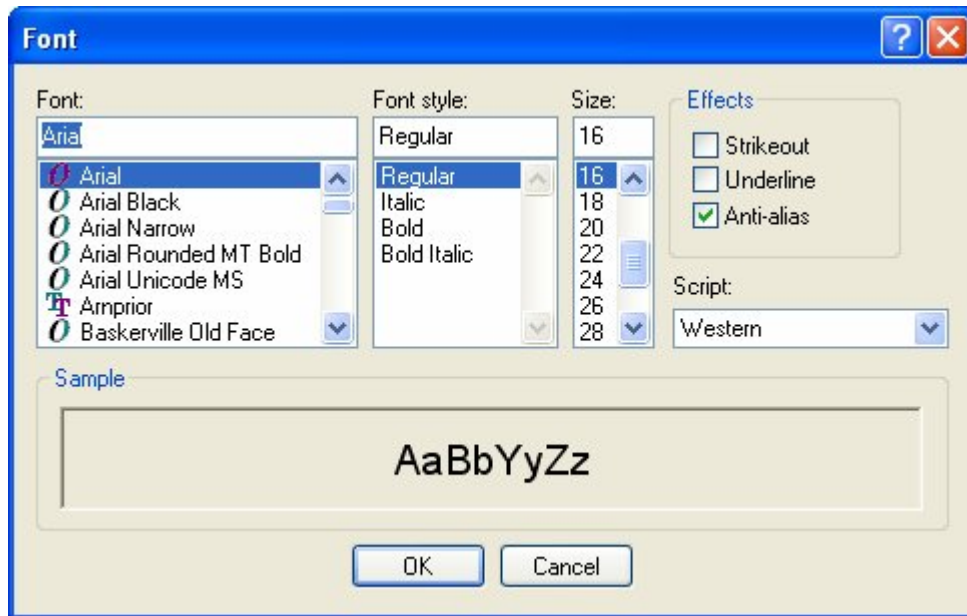
Note: Just because you *can* rename objects, doesn't mean you *have* to. For example, our page only has one image object on it, so we don't really need to give it a special name. (It shouldn't be too hard to remember which object “Image1” refers to.)

Changing Font Settings

Every label object contains text, but that doesn't mean they all have to look the same. AutoPlay gives you full control over several font settings.

1) Click on the Label1 object (the one with “Ted Sellers” on it). In the properties pane, click on the Font setting, and then click the edit button to bring up the Font dialog.

The Font dialog is where you can adjust all the font-related settings for the currently selected object.



2) In the Font dialog, change the font to Tahoma, the font style to Bold, and the size to 36. Leave the other settings the same. Click OK when you're done.

The first setting on the left side of the Font dialog lets you specify the font family you want to use. To switch to a different font, you can either type the name of the font into the text field, or select it from the list below.

To find “Tahoma” quickly in the list, select a name in the list (it doesn’t matter which one), and press the T key. The list will jump to the first font that begins with the letter T. “Tahoma” shouldn’t be too far away. When you find it, just click on it in the list to select it.

Similarly, to set the font style to Bold, just click on the word Bold in the list of font styles. To set the size to 36, either type in “36” in the Size field, or click on the number 36 in the list.

When you click OK, the label object will automatically use the new font settings.

Tip: You can also set some of the font settings in the properties pane.

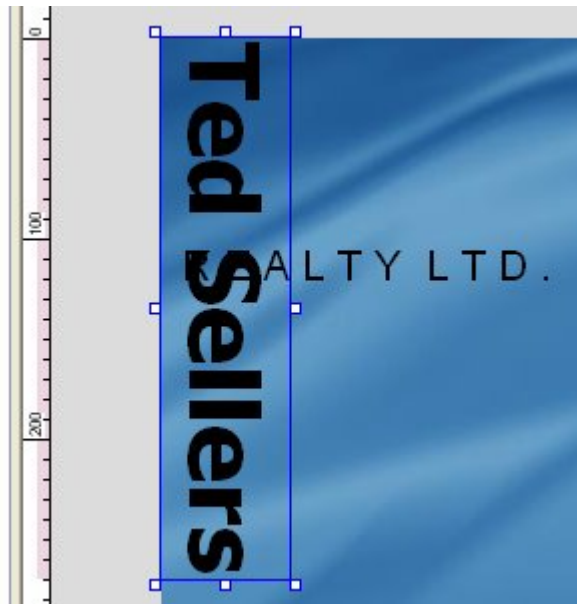
3) Change the Orientation from 0 to 270.

Changing the orientation changes the “direction” that the text is written in.

To illustrate this, we’ll turn the “Ted Sellers” label on its side and make its text read from the top down, so it would read normally (from left to right) if you tilted your head 90 degrees to the right.



To do this, click on the Orientation setting in the Properties Pane, click the select button, and then choose “270” from the list. This will orient the text 270 degrees counter-clockwise, so that it is written vertically from the top down.



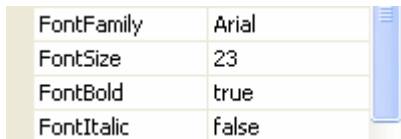
Pretty neat, eh?

4) Change the orientation back to 0.

Setting the label object’s orientation back to 0 will return it to normal, so it reads from left to right without any head tilting. (This assumes you’re not lying on your side or hanging upside down, but you get the picture.)

5) In the properties pane, set FontFamily back to “Arial,” and change FontSize to 23.

You can also change some font settings without opening the Font dialog, by using the four individual font settings in the properties pane.



| | |
|------------|-------|
| FontFamily | Arial |
| FontSize | 23 |
| FontBold | true |
| FontItalic | false |

These settings provide a quick way to change the most common font settings right from the properties pane. (As you’ll see in lesson 3, these settings are especially helpful when working with multiple objects.)

Changing the FontFamily setting is equivalent to choosing the font. In fact, what we normally consider the “name” of a font is technically referred to as the font’s *family*. Arial, Times New Roman, Garamond...each of these refer to a different font family. The reason they’re called “families” is because they encompass all of the different variations within that font...such as Bold, Italic, etc. Each font is actually made up of a collection of different “sub-fonts,” one for each of the various styles within the family. The “Arial” family comprises Arial Bold, Arial Italic, Arial Bold Italic, and so on.

To change the FontFamily setting, just click on it, then click the select button and choose “Arial” from the list. (The same trick applies to this list—you can press the “a” key to jump to the first font name that begins with the letter A.)

To change the FontSize setting, either edit the text in the field to replace 36 with 23, or click the little up/down arrows to change the size one point at a time.

6) Select the Email Address label object. In the properties pane, set the FontBold setting to “true.”

To change the FontBold setting, you could click on the setting, click the select button, and then choose “true” from the list. But there’s an even easier way to change a true-or-false setting like this one: just double-click on the word “false.”

Tip: If a setting offers you a list of values to choose from, you can double-click on the current value to advance to the next item in the list.

If the current value is the last item in the list, it just goes back to the beginning. Since the FontBold setting only gives you two options to choose from—namely, true and false—double-clicking on the value goes from false back to true.

Tip: You can toggle a setting between true and false by double-clicking on it.

7) Double-click on the Label2 object (the one with “R E A L T Y L T D .” on it). On the Label Properties dialog, click Font to open the Font dialog. Set the font style to Bold and the size to 12, and click OK.

You can also change the font settings by double-clicking on the object to open the Label Properties dialog, and then clicking the Font button to open the Font dialog.

To change the font style, just select “Bold” from the list. To set the font size, either highlight the original size (16) and type in *12*, or just select “12” from the list.

Clicking OK on the Font dialog will accept these changes and close the Font dialog.

Note: The Font dialog appears on top of the Label Properties dialog, and it must be closed before you can exit from the Label Properties dialog. Because the Font dialog is smaller, part of the Label Properties dialog will be visible behind the Font dialog. A common mistake is to click on the wrong dialog’s OK button. Luckily, the Font dialog won’t let you sneak around it like that. If you click the wrong OK button, the Font dialog’s title bar will flash (and you might hear your computer’s “alert” sound). If this happens, just click the other OK button.

8) Click OK to close the Label Properties dialog.

Clicking OK on the Label Properties dialog will accept the changes that you made to the object (i.e. the changes that you made to the Font settings) and close the Label Properties dialog.

Why are there different ways to change the font settings?

AutoPlay is designed to be as intuitive and flexible as possible. Having more than one way to perform a task allows you to pick the method that *you* like best, or that is most convenient at the time.

Some people only use the properties pane, while others only use the double-click dialogs. And some just use whatever they feel like using at the time.

Feel free to use whichever method you prefer when you’re working with AutoPlay. It’s your program...use it the way that makes sense for *you*.

Changing Text Colors

Label objects don't have to be black and white. Far from it! AutoPlay lets you set four different colors for each label object—one for each of the three different states that a label object can be in, as well as a 'disabled' color.

What's this about three different states?

Well, each object is able to display three different colors in response to what the user does with the mouse. Normally—when the mouse pointer is *not* over the label object—the “Normal” color is used for the text. If the user positions the mouse pointer *over* the label object, the “Highlight” color is used instead. And if the user *clicks* on the object, the “Down” color is used.

These three different situations—normal, highlighting and clicking—are the three *states* that the label object can be in.

So why three states? Well, it turns out that label objects can be made to do something when the users clicks on them, or moves the mouse onto (or off of) them. By using different colors for the three states, you can provide visual cues for the user to help them see that those label objects are interactive.

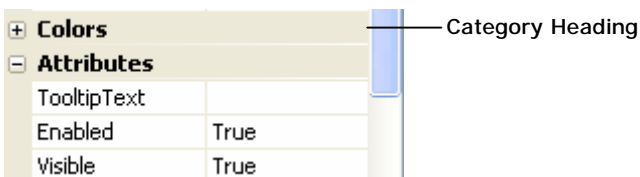
In other words, the three colors let you make label objects that look and act like hyperlinks, similar to the links you're used to seeing on web pages.

This is probably a lot easier to understand if you can see it in action, so let's turn one of our label objects into that kind of interactive label.

Note: There is actually a fourth state, too, for when the object has been purposefully disabled. This allows you to “grey out” an object to make it look “unclickable” when it has ceased to be interactive. You usually don't need to use this state, though, unless you're building a more complex application.

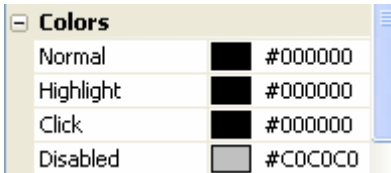
1) Click on the Email Address label object to select it. Make sure the Colors category is open in the properties pane.

If the Colors category is closed, it will look like this:



To open it, either double-click on the category heading, or click once on the little + symbol to the left of it.

Once the category is open, it will look like this:



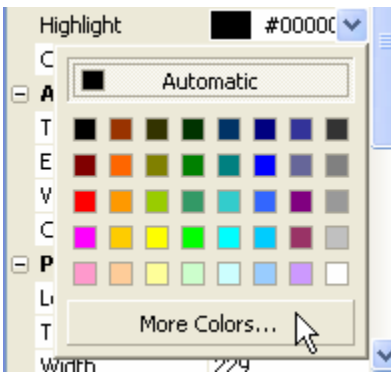
2) Double-click on the Normal color and type #bad5eb.

Double-clicking on a color setting highlights all of the text in it, so that what you type replaces the existing value. In this case, you want to change the default value of #000000, which is the hexadecimal color value for black, to #BAD5EB, which is the hexadecimal color value for a light blue-grey that looks good against the page's background image.

Note: When typing in hexadecimal color values, you don't have to use capital letters. AutoPlay will automatically capitalize the letters for you.

When you set the Normal color, you should see the label object's text change from black to light blue-grey.

3) To change the Highlight color, first click the select button to open the color chooser, then click on "More Colors..." at the bottom.

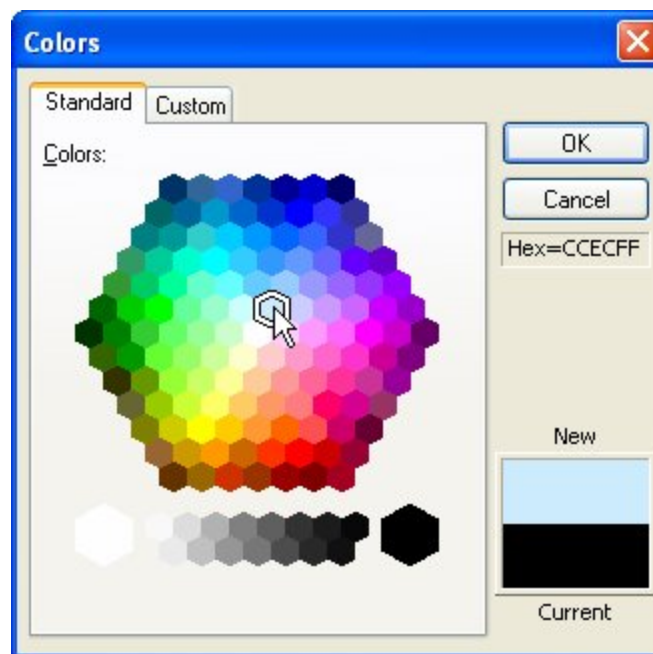


In order to make our label object look like a hyperlink, we want the highlight color to be a lighter version of the normal color. That way, the user will see the text “highlight” when they move the mouse over it. The color chooser doesn’t have the color we want, though, so we need to go one step further. That’s where the “More Colors...” link comes in.

Clicking on “More Colors...” at the bottom of the color chooser opens the Colors dialog.

4) On the Colors dialog, select the light blue color one position up and to the right from the white color in the center (its hexadecimal value is CCECFF). Click OK when you’re done.

The Colors dialog has two tabs: Standard, and Custom. The Standard tab, which is selected by default, has a broader selection of the standard Windows colors for you to pick from. In this case, you want to select the color that is immediately up and to the right from the color in the center.



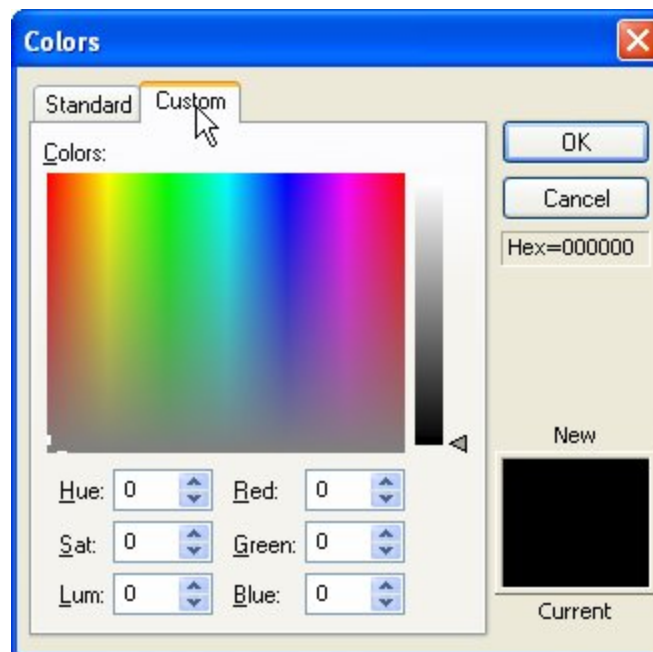
When you’re done changing the color, click OK to close the Colors dialog. The color you selected will show up in the properties pane as the object’s new Highlight color.

You won't be able to see the real effect of changing the Highlight color, though, until you actually see the project running.

Note: The text color will change to the Highlight color when the user moves the mouse *onto* the object. Since the object's size and shape are determined by its bounding box, this means the color will change when the mouse pointer crosses over the edge of the bounding box (like crossing the border into the object's territory).

5) To change the Click color, open the Colors dialog (by clicking on "More Colors..." at the bottom of the color chooser), and click on the Custom tab.

If you can't find the color you want on the Standard tab, you can switch to the Custom tab to create a custom color from scratch.



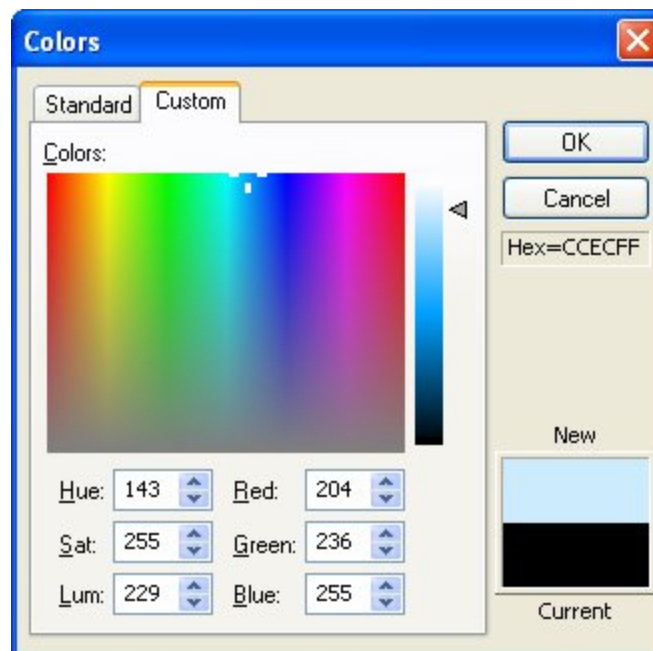
The Custom tab lets you select a color from a "limitless" palette, either by clicking on the color map, or by setting the Hue/Saturation/Luminosity or Red/Green/Blue values directly. If you've used any photo editing tools, you're probably already familiar with this interface, or have used one like it.

For the Click color, we want a color that is similar to the Highlight color, but a little bit lighter. There's a really easy way to do this using the custom tab, but first we have to "prime" it with the Highlight color. (Right now, the Custom tab is showing the settings for the current Click color, which is black. Starting from "scratch" like this would make it harder to pick a color that is similar to the Highlight color.)

6) Click on the Standard tab, and click on the same color that you selected in step 4. Then click on the Custom tab again.

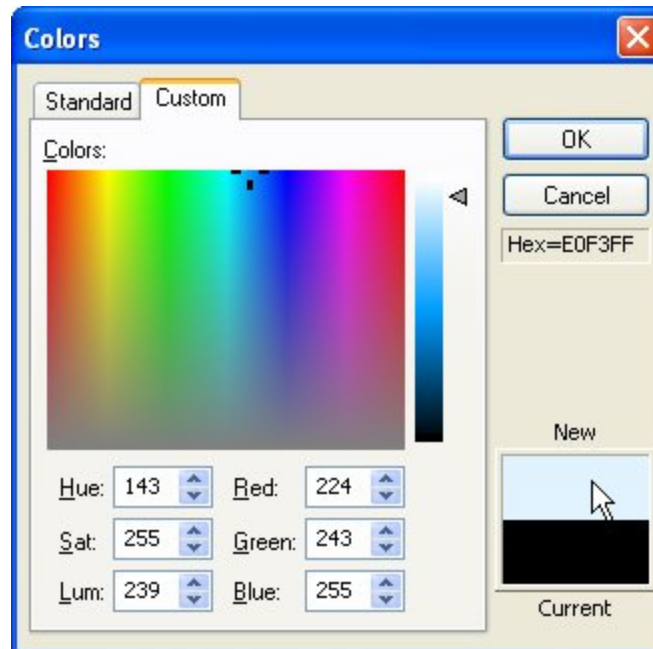
Going back to the Standard tab and selecting the same light blue-grey color that you selected in step 4 will "set" the Click color to that color. This doesn't actually change the object's Down setting...at least, not until you click OK and accept the changes. But it gives us a better starting point to create a similar color for the label's Click state.

When you return to the Custom tab, you'll see the color settings for that light blue-grey color.



7) Drag the luminosity slider up a bit to make the color brighter.

The luminosity slider is the vertical bar with the little triangle next to it, just to the right of the big color map. You can drag the triangle up and down to change the color's luminosity, without changing its saturation or hue.



Pick a color that is noticeably brighter than the Highlight color. (I set the luminosity to about 239, which resulted in a color whose red, green and blue values were 224, 243, and 255, respectively.)

8) Click OK to finish setting the Click color.

When you click OK, the Colors dialog closes, and the color you created shows up in the properties pane as the object's new Click color. Once again, you won't be able to see any other effects of this change to the object until you see the project running.

Note: The text color will change to the Click color when the user clicks inside the object's bounding box.

Watch out for those crooked fonts...

Some italic or “cursive” fonts can pose a bit of a problem for the label object. For various technical reasons, a font that leans to the right a lot can sometimes end up with the last letter extending past the right edge of the label’s bounding box. When this happens, the part that falls outside the bounding box won’t change colors along with the rest of the text—only the text inside the label’s bounding box responds to mouse-overs and clicks.

This is very rare, but if it ever happens, there’s a simple solution: just add a few spaces to the end of the text, so the bounding box is big enough for the last letter to fit.

Copying Colors

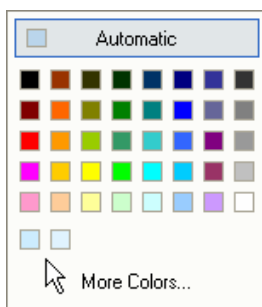
Copying a color from one object to another is something you’ll do fairly often. Luckily, AutoPlay gives you two really fast ways to duplicate a color.

To demonstrate this, let’s make our Phone and Mailing Address label objects use the same color as the Email Address object’s Normal color.

1) Click on the Email Address label object. In the properties pane, click on the Normal color setting, and click the select button to bring up the color chooser.

The first way to give two color settings the same value only works if you used the Colors dialog to set the first color. This is because the color chooser actually keeps track of the last eight colors you added with the Colors dialog and makes it easy for you to select them again.

In the previous exercise, you used the Colors dialog to set the Highlight and Click colors for the Email Address label object. Whenever you choose a color using the Colors dialog, a new color square is added to the bottom row of the color chooser.



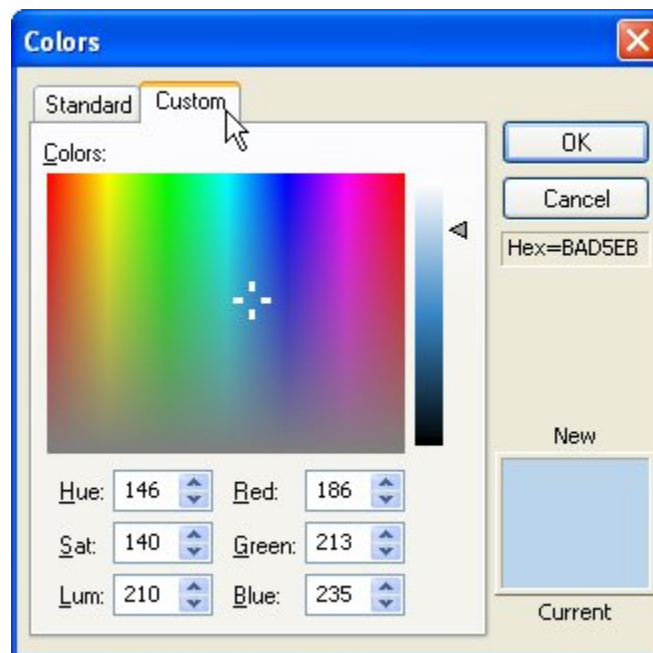
As you can see, there are two such color squares on the bottom row of the color chooser: one from when you chose the Highlight color, and one from when you chose the Click color.

There is no color square for the Normal color, though, because it wasn't set using the Colors dialog. That color was changed by typing the hexadecimal color value directly into the Normal setting. Only colors that are set using the Colors dialog will show up on the color chooser.

In order to create a color square for the Normal color, we need to set it again using the Colors dialog.

2) Click on “More Colors...” at the bottom. On the Colors dialog, click on the Custom tab, and click OK.

When you switch to the Custom tab, the color settings contain the values that describe the current color.

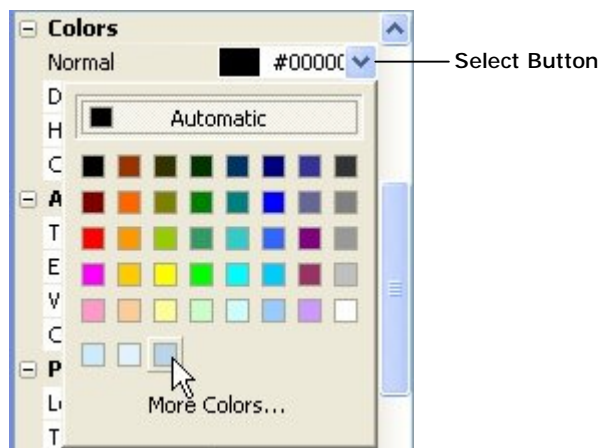


Since the Email Address label object's Normal color is already set to the color we want, all you have to do is click OK on the Colors dialog, and you'll effectively

“re-set” the Normal setting to the same color—only this time, in a way that will result in a new color square being added to the color chooser.

3) Click on the Phone label object. In the properties pane, click on the Normal setting, and click the select button. A third color square has been added to the bottom row of the color chooser. Click on that square to select the same color you chose in step 2.

When you re-selected the Normal color using the Colors dialog in step 2, a new color square was added to the bottom of the color chooser.



This new color square will be available in all color choosers until you exit AutoPlay. So, if you want to use the same color somewhere else—in another label object, for instance—you just have to click on that new color square.

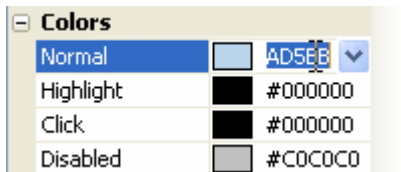
Clicking on the color square sets the Normal color for the Phone label object to the same color you chose for the Email Address label object.

Note: As you continue to select colors using the Colors dialog, additional color squares will be added to the bottom row of the color chooser. The color chooser will only remember as many colors as it can fit on the bottom row, however. If you select enough colors, the color chooser will “forget” the older ones in favor of the new ones. In other words, the color chooser can only show the last eight custom colors that you picked.

The other way to duplicate a color is to copy and paste the hexadecimal value from one setting to another.

4) Double-click on the Normal color setting and press Ctrl+C.

When you double-click on a color setting, all of the text in its text field is highlighted for you. In this case, the text is the hexadecimal value that describes the Normal color.

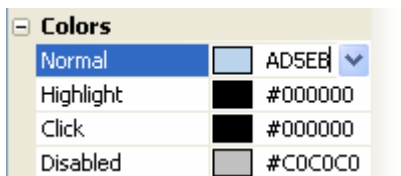


Pressing Ctrl+C copies this text into the clipboard so you can paste it somewhere else.

5) Click on the Mailing Address label object. In the properties pane, double-click on the Normal color setting and press Ctrl+V.

Once again, when you double-click on the Normal color setting, its hexadecimal value is highlighted. Pressing Ctrl+V *pastes* the text from the clipboard into the text field, replacing the highlighted text.

Note: This time, instead of copying the hexadecimal text into the clipboard, you're replacing it with the value that you copied from the other label object.



Tip: Unlike the new color square on the color chooser, which only works for colors that you've selected with the Colors dialog, you can always copy and paste the hexadecimal value between two colors.

6) Press Enter, or just click somewhere else.

To accept the change that you made to the Normal setting, you can either press the Enter key, or click “away” from the Normal setting. (In other words, click on another setting, or another object, or on the page surface.)

Tip: If you wanted to cancel the change, you could press Esc instead.

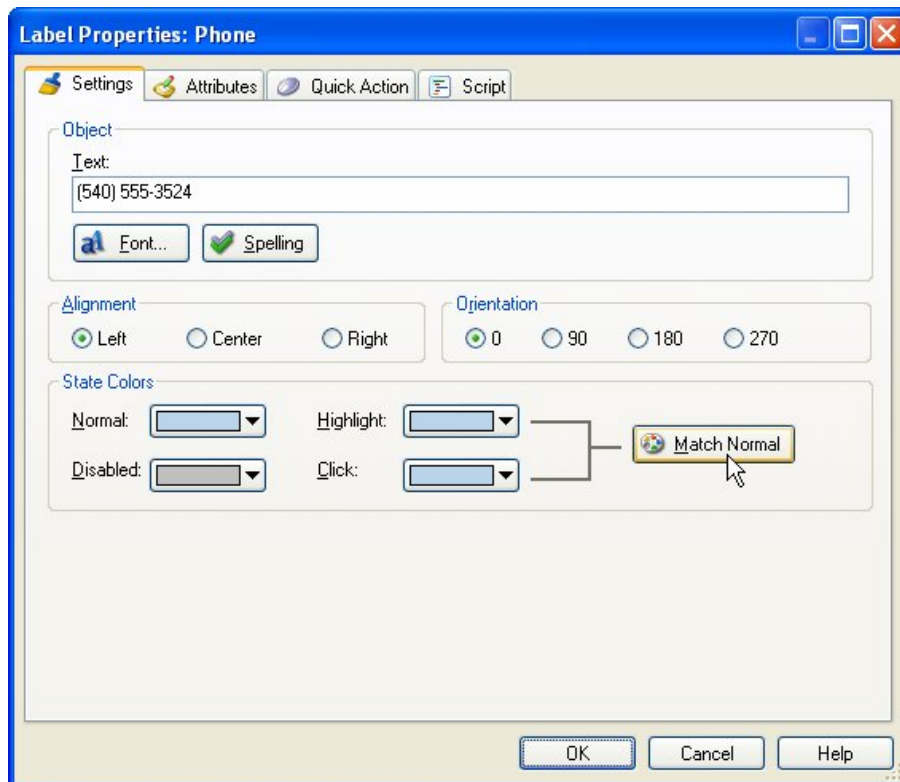
Matching Colors

Unlike the Email Address object, we don't want these labels to appear interactive, so we need to make all three state colors the same. Otherwise, the text colors will change when the user moves the mouse over the labels or clicks on them, which will give the wrong impression.

To make all three colors the same, you could just copy the hexadecimal color value from one of the states and then paste it into the other two—just like copying a color from one object to another. But there's an even easier way to make the colors match: just use the Match Normal button on the object's double-click dialog.

1) Double-click on the Phone label object.

Double-clicking on the label object opens the Label Properties dialog. There are four color choosers on the Settings tab which allow you to set each of the four state colors, just like you can in the properties pane.



Next to the Highlight and Click colors is the Match Normal button.

2) Click the Match Normal button.

Clicking the Match Normal button makes the Highlight and Click colors both match the Normal color.

3) Click OK to accept the changes.

Clicking OK closes the dialog and “locks in” the new Highlight and Click colors.

4) Do the same for the Mailing Address label objects.

Double-click on each Mailing Address label object and make its Highlight and Click colors match its Normal color, too.

While we’re at it, let’s change the color of our first two label objects as well.

5) Double-click on the Label1 object to open the Label Properties dialog. Set the Normal color to white, and then click the Match Normal button to make the Highlight and Click colors match.

The Label1 object is the one with “Ted Sellers” on it. In the next lesson, we’re going to use it with Label2 and the image object to create a little logo. Since it’s going to be used as part of a logo, we don’t want it to appear interactive, so the Normal, Highlight and Click colors should all be the same.

6) Do the same for the Label2 object.

The Label2 object is the one with “R E A L T Y L T D .” on it.

Adding a Slogan

Remember when I mentioned that duplicating an object was a good way to add a similar object to the page? Essentially, you use an existing object as a starting point to create a new one. Let’s take advantage of that trick to add another label object with the same color settings as our Mailing Address label object.

We’ll use this label object to display Ted’s slogan: “Building communities one home at a time.”

1) Select the Mailing Address label object and press Ctrl+D.

Pressing Ctrl+D creates a duplicate of the Mailing Address object, and automatically selects the new object.

2) Drag the new object over to the left corner of the page.

Moving it away from the Mailing Address label object will make it easier to see what's going on.

3) In the properties pane, change the new object's name to *Slogan*.

We're starting to accumulate a fair number of objects. We might as well give this object a name so the list in the object browser doesn't get too confusing.

4) Click on the Text setting, and click the edit button. When the Edit Text dialog appears, replace the mailing address with the following text:

***"Building communities
one home at a time"***

Clicking the edit button in the Text setting opens the Edit Text dialog. When the dialog appears, all of the text in it is already highlighted for you. To replace the text, you just have to type in the new text.

5) Click OK to close the Edit Text dialog.

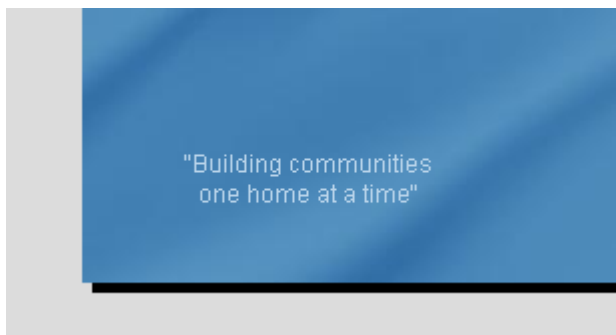
When you're done changing the text, click OK to close the Edit Text dialog. The new text should appear in the label object immediately.

6) Set the font Size to 9, and the Alignment to "Center."

We want the text in the label object to be centered, and we don't want it to be very big.

7) In the Position category, set Left to 46 and Top to 358.

This positions the object down near the bottom left corner of the page.



Note: This will place the slogan below the four button objects we'll be adding in Lesson 4.

That's it...we don't have to bother changing any of the colors, because all of those settings were already made for us simply by duplicating the object.

Saving the Project

If you've been using computers for a while, you know that it's important to save your work from time to time. If you haven't, take my word for it: save your project, and save it often!

(In a perfect world, computers would never crash, and kids would never think the power button was really fun to play with.)

So, before we finish off this lesson, let's take a moment to save the project.

1) Choose File > Save.

Note: When you save the project, all of the changes that you've made to the project are stored on your hard drive, in a project-specific file known as the project file.

Congratulations! You've just protected yourself from having to redo everything in the event that a computer glitch (or four-year-old) causes your computer to spontaneously shut down.

As you can see, there isn't much to it. Feel free to save the project as often as you want. Generally, the more often you save the project, the better.

Tip: You can also save the project by pressing the Ctrl+S hotkey.

Previewing

Now that we've added some text and graphics, let's see how our project would look from the user's perspective. AutoPlay's preview feature makes it easy to take a quick look at how your application will look and operate when it's done, without actually building or burning the project.

1) Choose Publish > Preview.

When you choose Publish > Preview, your application starts up, just like it would if you built the project and ran it.

Previewing doesn't make any changes your project, so feel free to preview your project at any time.

Tip: You can also preview your project by pressing the F5 key, or by clicking the Preview button on the toolbar.



2) Move your mouse over the Email Address label object, and click on it. Observe how the text color changes as you interact with the object.

The Email Address label object is the one with “ted@sellersrealty.com” on it. As you move the mouse over this object, the text color changes from the Normal color (light blue-grey) to the Highlight color (a lighter shade of blue). When you click on the object, the text color changes from the Highlight color to the Click color (an even lighter shade of blue).

As you can see, all you need to do to make a label object appear “interactive” is set the Normal, Highlight and Click colors to different colors.

Note: In Lesson 4, we'll show you how to make this label object actually send an email when it's clicked.

3) Move your mouse over the “Ted Sellers” label object and click on it.

Since you made all three of the “Ted Sellers” label object's colors the same, it doesn't visibly respond to mouse-overs or clicks at all.

4) Click the close button on the title bar to exit the preview.



When you exit the preview, you're returned to the AutoPlay program window.

Tip: You can also press Alt-F4 to exit from a preview.

Lesson 2 Summary

In this lesson, you learned how to:

- Change the page background
- Add an image object
- Resize objects
- Add label objects
- Duplicate objects
- Change the text in a label object
- Rename objects
- Use custom font settings
- Use different text colors
- Copy a color from one object to another
- Match an object's normal, highlight and click colors
- Save the project
- Preview the project



Lesson 3:

Working with Multiple Objects

There are all sorts of things you can do with multiple objects, like move them all at once, line them up, group them together, or even change some of their properties in one fell swoop.

This is something you'll do a lot, and as you'll see in this lesson, it's really easy once you know how.

3

What You'll Learn

In this lesson, you'll learn how to:

- Select multiple objects
- Move more than one object at a time
- Change the settings of several objects at once
- Align objects to each other
- Align objects to the page
- Arrange objects in the z-order
- Remove unused files from the project
- Group objects together
- Pin objects (so they can't be moved or resized)
- Distribute objects evenly
- Lock objects (so they stay out of your way)

How Long Will It Take?

This lesson takes approximately 35 minutes to do.

Starting the Lesson

If you're continuing from Lesson 2, you should still have AutoPlay running with the Tutorial project open. If so, you're ready to move on to the next exercise: Selecting Multiple Objects.

Otherwise, you'll need to open the project file that you saved at the end of Lesson 2.

1) Open the Tutorial.am7 file that you saved in Lesson 2.

When you save a project, AutoPlay automatically creates a project folder for it inside your "My Documents\AutoPlay Media Studio 7.0\Projects" folder. This project folder is where everything that belongs to the project is stored—including the project file, which contains all of the settings used in the project.

The project folder and the project file always have the same name that you gave to the project when it was created. The project file's name ends with a ".am7" file extension.

Since you named this project "Tutorial" in Lesson 1, the name of the project folder will be Tutorial, and the name of the project file will be Tutorial.am7. So, the path to the project file should be something like:

```
...\My Documents\AutoPlay Media Studio 7.0\Projects\Tutorial\Tutorial.am7
```

To open the project, you just need to open that project file.

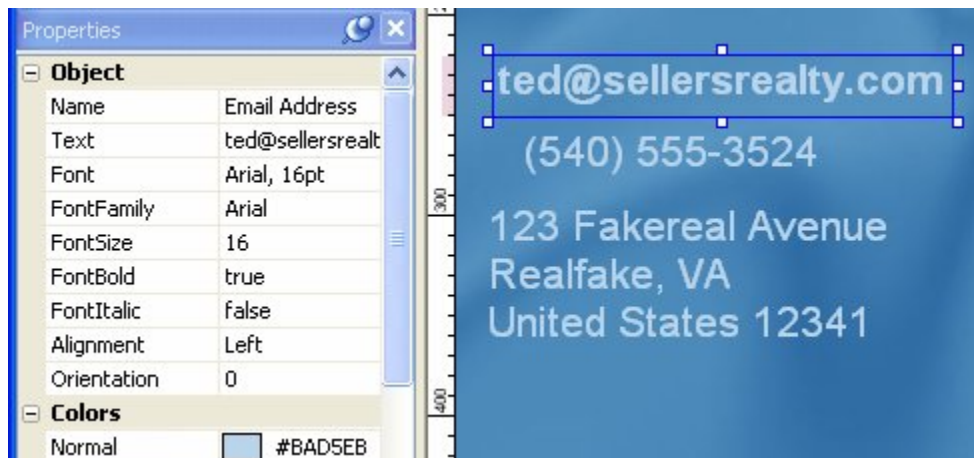
Selecting Multiple Objects

The first step to doing anything with multiple objects is to select the objects you want to work with.

Note: If you're used to selecting multiple files in Windows, you probably already know how to select multiple objects in AutoPlay. Just like in Windows, you either ctrl-click, shift-click, or drag-select. The only difference is that when you're working with objects in AutoPlay, ctrl-clicking and shift-clicking both do the same thing.

1) Click on the Email Address label object to select it.

When you click on the label object, the bounding box appears, and its settings show up in the properties pane.



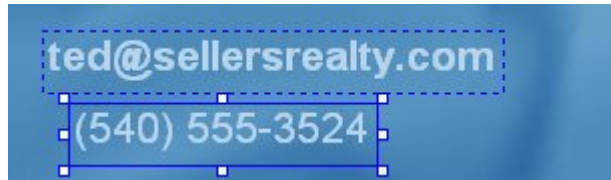
Before you move on to the next step, notice the way the object's bounding box looks, and take a quick peak at the properties pane to see the number of settings that are visible. (Both of those things will change when you have more than one object selected.)

2) Hold the Ctrl key down and click on the Phone label object. (This is known as *ctrl-clicking*.)

Ctrl-clicking (pronounced "control-clicking") *adds* an object to the current selection. In this case, the current selection consists of a single label object named Email Address.

Note: This is just like ctrl-clicking in Windows to select multiple files or folders.

When you ctrl-click on the Phone label object, it becomes selected in addition to the Email Address label object. The end result is that both objects are selected.

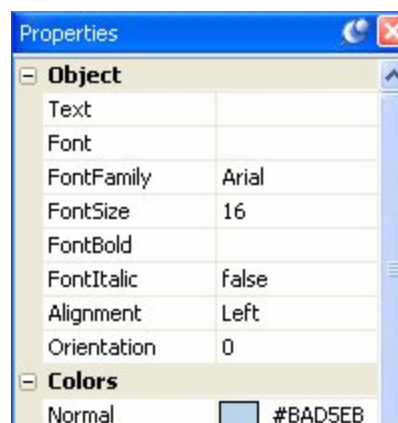


One of the objects has a normal bounding box, with solid edges and little white resize handles. But the other object has a bounding box made up of blue dashes, with no resize handles on it at all.

The object with the normal bounding box is called the *dominant object*. The dominant object is used as a point of reference when you align and resize objects using the tools in the Align menu. (We'll use those tools a bit later in this lesson.)

The other object is just part of the current selection. The dashed-line bounding box means that the object is selected but not dominant. As for the missing resize handles, you can only ever resize one object at a time with the mouse. When you have more than one object selected, the dominant object is the only one you're allowed to resize. So, the resize handles only appear on the dominant object.

Most of the settings are still listed on the properties pane, but some of them are blank, with no values in them at all. And the Name setting, which used to be at the top of the properties pane, isn't even there any more.



If a setting is blank, it means that the setting has different values between the selected objects. A setting will only show a value if *all* of the selected objects have that same value for the setting.

For example, because our two label objects have different text in them, the Text setting is blank. But, since they both have the same Alignment setting, the value of the Alignment setting is visible.

The Name setting is missing because you can't give the same name to more than one object at once. (As you'll see in this lesson, you can actually use the properties pane to change the settings for multiple objects.)

Tip: Feel free to shift-click instead if you prefer. When you're working with objects in AutoPlay, shift-clicking and ctrl-clicking are exactly the same.

3) Ctrl-click on the Mailing Address label objects.

The Mailing Address label objects are added to the selection. Notice that now the last object clicked has the solid bounding box, and the object that was dominant before—the Phone label object—has a dashed-line bounding box instead.



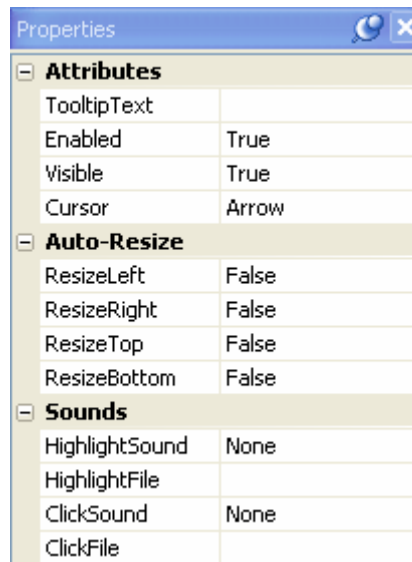
Note: When you add an object to the selection, it becomes the dominant object. (The dominant object is always the last object you clicked on.)

4) Ctrl-click on the image object to add it to the selection.

When you add the image object to the selection, most of the settings will disappear from the properties pane. This is because there are two different kinds of objects selected—namely, label objects and image objects.

Whenever there are multiple objects selected, the properties pane only shows the settings that all of those objects have in common. In this case, you only see the settings that exist in both label objects *and* image objects.

In fact, all that's left are a handful of settings in the various categories which exist across all of the selected objects.



5) Ctrl-click on the image object again to remove it from the selection.

If you ctrl-click on an object that is already selected, it is removed from the current selection. This is also known as “deselecting” the object.

6) Click on the page surface to deselect all of the objects.

Clicking on the page surface deselects any objects that are currently selected, and selects the page instead. (You can tell the page is selected because the page settings appear in the properties pane.)

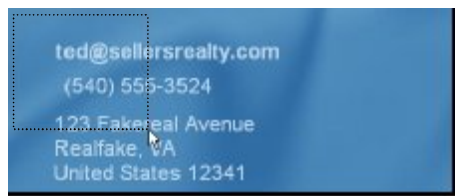
7) Click on the page surface and, while holding down the mouse button, drag a rectangle around the Email Address, Phone and Mailing Address label objects.

This is known as *drag selecting*. It basically involves drawing a temporary “box” around the objects that you want to select, by dragging the mouse from one point to another while the left mouse button is held down.

Note: To drag select a bunch of objects, position the mouse pointer somewhere on the page surface near the objects. (Don't position it on an object, or you'll just end up moving that object instead.) Click the left mouse button and, while holding it down, begin to move the mouse. A temporary drag-select rectangle will appear to outline the selection area. "Stretch" this rectangle by dragging the mouse until the selection area includes all of the items you want to select, then let go of the mouse button. When you release the mouse button, any items within (or touching) the selection area will be selected, and the drag-select rectangle will disappear.



Click and hold...



...drag...



...and release.

When you drag-select objects, the object that is closest to the mouse pointer when you release the mouse button becomes the dominant object. So, if you started dragging from above the Email Address label object, and ended up with the mouse pointer just below the Mailing Address label object, the Mailing Address label object would be the dominant object.

Tip: If you already have a bunch of objects selected, you can add more objects to the selection by drag-selecting them with the ctrl key (or shift key) held down. Holding the ctrl or shift keys down while drag-selecting *adds* objects to the current selection.

Moving Multiple Objects

When you have multiple objects selected, you can move them easily by dragging them with the mouse.

1) Drag the label objects to somewhere else on the page.



Moving multiple objects is just like moving a single object. When you drag one, all the other selected objects come with it.

So, just click on one of the selected objects, and, while holding down the mouse button, drag the objects where you want them to go.

Note: When multiple objects are selected, they all move together.

2) Choose Edit > Undo.

When you choose Edit > Undo, the last change that was made to the project is “undone.” In this case, the three label objects jump back to where they were before you moved them.

Tip: An even faster way to undo the last change is by pressing Ctrl+Z.

3) Choose Edit > Redo.

When you choose Edit > Redo, the last change that you undid is redone. It’s kind of like “undoing the undo.”

Tip: You can also redo the last thing that you undid by pressing Ctrl+Y.

4) Choose Edit > Undo again.

If you’re wondering what all this undoing and redoing has to do with moving objects, well, there *is* a point to it.

Moving objects and then undoing the move can be a very useful technique. You often won’t know exactly where you want your objects to go until you’ve moved them around a bit. Knowing that you can just undo the move if you don’t like where the objects end up frees you to try different ideas. (Sure, you could just move the objects back by hand, but using undo gets them back to that same exact spot a lot faster.)

It also doesn’t hurt to know how to go back a step if you happen to make a mistake, like moving the wrong object by accident.

As for undoing and redoing, this can be helpful when you’re trying to decide between two different positions. Just move the objects from one place to another, and then use the undo and redo commands to go back and forth between the two options. “Do the objects look better over here, or over there? Hmmm...here...no, definitely there.”

This kind of “undo/redo dance” lets you do a quick before-and-after comparison for any change that you make to the project.

Editing Multiple Objects

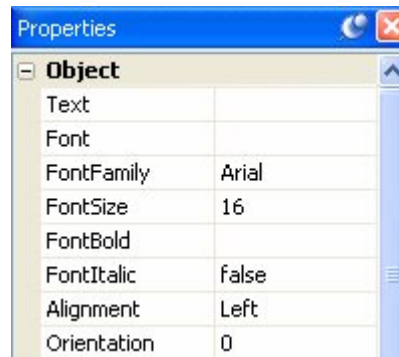
When you have multiple objects selected, you can change a setting for all of them at once just by editing it in the properties pane.

1) Make sure the three label objects are still selected.

If this isn't the case, just click on the page surface, then click on one object, and ctrl-click on the other two.

2) In the properties pane, change the FontSize to 11.

Since all three of the label objects are set to use a 16-point font size, the value 16 is visible in the FontSize setting.



When you change this value from 16 to 11, all three label objects will switch to using an 11-point font size. (You will actually see the text in the label objects get smaller.)

Note: Any changes that you make in the properties pane are applied to all of the objects that are selected.

Aligning Objects

When multiple objects are selected, a number of alignment tools become available. You can use these tools to line the objects up horizontally, vertically, or both.

1) With the three label objects still selected, click on the first Mailing Address object to make it dominant.

When you have multiple objects selected, clicking on one of them makes it the dominant object. There can only be one dominant object at a time, so if the object you

click on isn't already the dominant object, it becomes the dominant object instead of the one that was dominant before.

Note: The dominant object is the one with the normal bounding box.

2) Choose Align > Center Horizontal.

When you choose Align > Center Horizontal, all of the selected objects are centered horizontally with the dominant object.

In other words, the objects line up so that the horizontal midpoint of each object is in line with the horizontal midpoint of the dominant object.



Before aligning...



...after aligning.

Notice that the dominant object didn't move, but the other ones did. The dominant object serves as the "anchor" for any alignment operations that you perform.

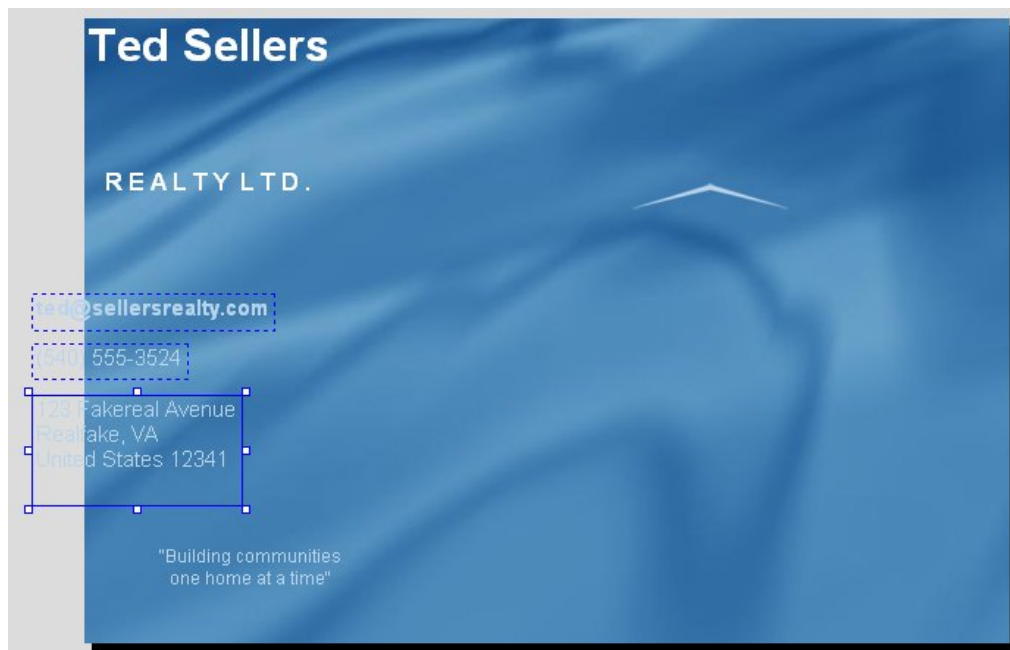
3) Click on the Email Address label object, then choose Align > Left.

Clicking on the Email Address label object makes it the dominant object, so when you choose Align > Left, the Email Address label object stays put, and the other objects are aligned to *it*.

Note: The key to performing object alignments is knowing what the objects will line up with. Luckily, it's quite simple: the objects are always aligned to the dominant object.

4) Drag these three label objects over to the left side of the page.

Don't worry about positioning them carefully; we just want to get them out of the way for a while.



Note: You can drag objects onto the work area, but only the objects (or parts) that are located on the page will be visible in the finished application. If any objects extend onto the work area, a warning will occur when you preview or build the project.

5) Click on Label1 (the one with “Ted Sellers” on it).

When you click on Label1, it becomes the only selected object. The other objects aren't selected any more, because you didn't ctrl-click on Label1 to *add* it to the selection—you only clicked on it to *select* it.

Clicking on an object that isn't selected deselects any objects that currently are. It's essentially like saying “I want to select *this* object now, and only this one.”

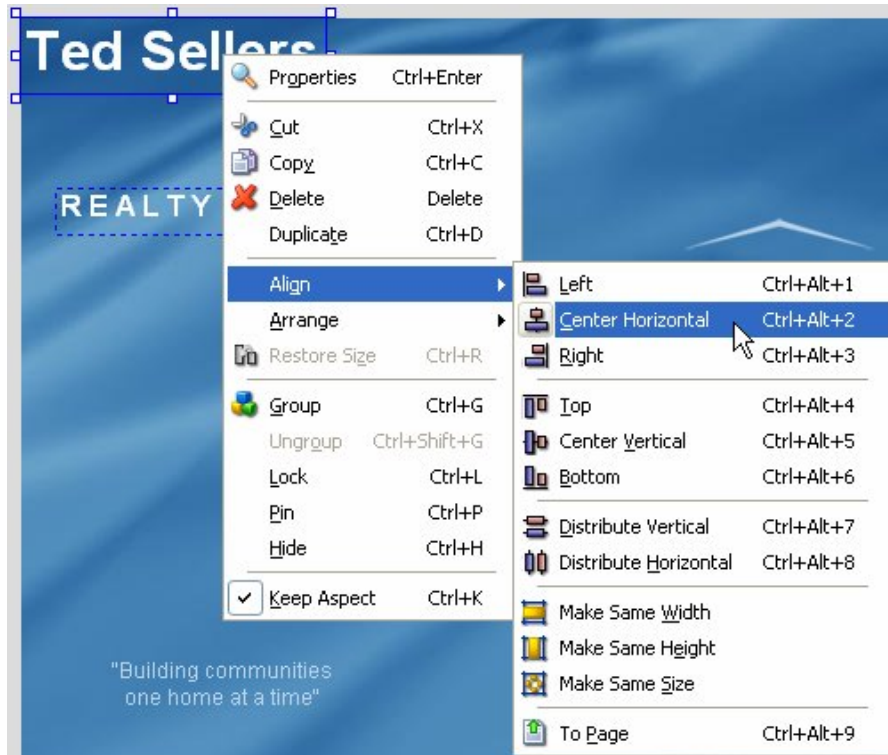
6) Shift-click on Label2 (the one with “R E A L T Y L T D .” on it).

Remember: when you’re working with objects in AutoPlay, shift-clicking is exactly the same as ctrl-clicking. (You can use whichever key you prefer.)

Tip: When selecting multiple objects, use the “last-clicked object is dominant” rule to your advantage. Plan the order you ctrl-click on stuff so that the object you want to align to is the last one you click on.

7) Right-click on the “Ted Sellers” label object and choose Align > Center Horizontal.

When you have multiple objects selected, right-clicking on one of the objects makes it the dominant object. It also opens the right-click menu.



As you can see, right-clicking on an object gives you easy access to the same alignment tools that are in the Align menu.

Tip: The right-click menu is a very fast way to access the alignment tools.

Now that you have these two label objects selected, let's use them with the image object to create a simple logo.

8) Choose View > Toolbars > Align to make the Align toolbar visible.

All of the alignment tools are available on a toolbar, which we have appropriately named "the Align toolbar." (Don't look at me, I wanted to name it George.)



You can use the Align toolbar to access the same alignment features that are available in the Align menu.

Tip: When I'm working with the alignment toolbar, I like to drag it down so it floats over the top part of the work area. That way, it's closer to the objects, and doesn't take up as much vertical space in the design environment.

8) Drag the two label objects just below the image object. Position the objects so they look like the image below. Use the alignment tools to center the label objects horizontally with the image object.



You'll probably need to move the two label objects individually in order to bring them closer together. Try to position the objects without moving the image object at all.

Tip: Feel free to play around with the alignment tools before moving on to the next exercise. They're a lot of fun to use, and being familiar with them makes it a lot easier to design attractive layouts.

Aligning Objects to the Page

In addition to lining objects up with each other, you can also align them to the page surface itself.

Our goal for this exercise will be to try moving our newly created logo to the exact center of the page.

1) Select the three objects that make up the logo.

In other words, select the image object along with Label1 and Label2.

Tip: You can use the drag-select method to select all three objects at once.

2) Choose Align > To Page, or click the Align to Page button.

Choosing Align > To Page makes the objects align to the page surface, instead of to the dominant object. It essentially makes the *page surface* act like the dominant object, so you can use the alignment tools to align objects to the page.

Note: The To Page menu item acts as a toggle. If you choose Align > To Page twice, it toggles the feature on, then off again.

Clicking the Align to Page button does the same thing.



Tip: If the alignment toolbar isn't visible, you can choose View > Toolbars > Align to make it visible.

You can tell which mode the alignment tools are in by looking at the Align to Page button on the Align toolbar. If the button has a blue border around it, then Align to Page mode is on.



Without the Align toolbar, you'll have to check the Align menu in order to know whether Align to Page mode is on, which is why I like to make the Align toolbar visible whenever I plan on aligning any objects to the page.

3) Click the Align Center Horizontal button.



All three objects should now be centered horizontally on the page. Notice that they only moved horizontally, and didn't move up or down at all.

3) Click the Align Center Vertical button.



Whoa! The objects are centered vertically on the page all right...but now they're overlapping, with one object on top of the other.



This isn't exactly what we wanted, but if you think about it, it makes sense. We had three objects selected, and we told AutoPlay to center them vertically on the page. So that's exactly what it did. Each one was centered on the page, independently.

If you wanted the three objects to keep their positions relative to one another, and be centered vertically as a single unit, you'd have to group them first. We'll try that a bit later in this lesson, but for now, let's take advantage of the situation to cover another useful skill: arranging objects.

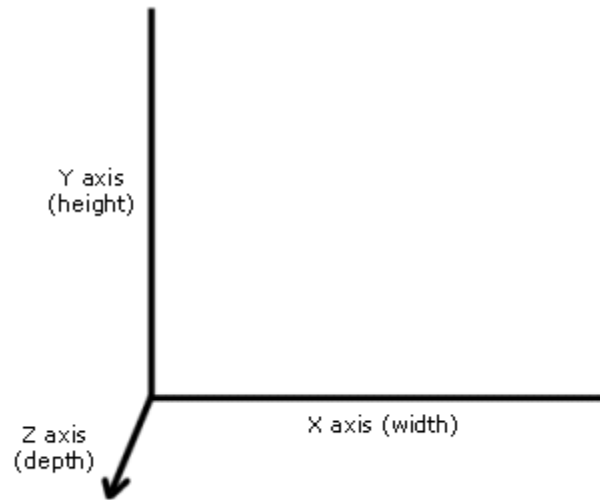
Arranging Objects

While we have these three objects overlapping, let's perform a quick experiment to illustrate a point.

As you can see, objects are drawn on top of each other. When objects overlap, the object in front essentially hides any objects that are behind it. (Depending on the object, you might be able to see other objects "through" the one that's on top. For example, our label objects don't hide each other very much, because label objects are fully transparent except for the text.) So what if you want a different object to be in front? Easy: just *arrange* the objects.

Arranging objects involves changing their positions in something called the *z-order*. The z-order is what determines which objects will be closer to the user, and which objects will be closer to the page.

It's called the "z-order" because it deals with how objects are arranged along a page's z axis.



In AutoPlay, changing the z-order involves moving individual objects along the z axis—either "forward" (further from the page) or "back" (closer to the page).



The Z-Order in Action

Note: Some object types are unable to appear behind other kinds of objects. For example, if you try to place a label object in front of a video object, it won't work; the video object will still show up in front. This is because the video object exists in its own little "mini-window" that floats on top of the page surface.

Z-Order = Draw Order

Objects are actually drawn onto the page one at a time. (It looks like they're all just instantly "there" because your computer can draw objects very quickly.) The z-order is what determines the order that the objects will be drawn in. The ones at the back are drawn first, and the ones at the front are drawn last, right over the others.

Some objects, like video and web objects, are drawn onto a separate window "surface" that is permanently suspended above the page and all of the "normal" objects on it. We call these objects *windowed objects*. Windowed objects always appear in front of non-windowed objects, regardless of where they are in the z-order.

1) Press Ctrl+Z to undo the last change.

Pressing Ctrl+Z undoes the last change that you made to the project, which in this case was to center the three objects vertically on the page.

Note: This is the same as choosing Edit > Undo.

By default, AutoPlay remembers the last 10 changes that were made, so at any given moment you should be able to "step back" like this up to 10 times in a row.

Tip: You can give AutoPlay a longer undo memory by choosing Edit > Preferences, clicking on the Document category, and setting Undo/Redo Levels to a higher value.

2) Choose File > Save to save the project.

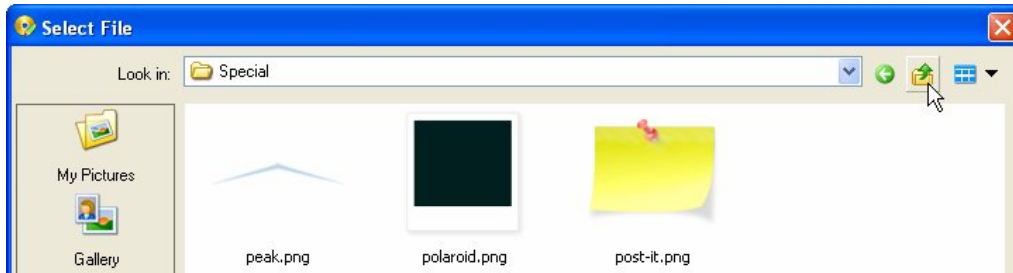
Saving the project is important, because it creates a point that you can come back to if you make a mistake or want to undo your changes. (We'll actually use this feature later in the lesson, to return the project to the state it's in right at this very moment.)

3) Right-click on the page surface and choose Image. In the File Select dialog, navigate up one level, and then navigate into the Icons folder. Select the 002BT15.png file, and click OK.

Choosing Image from the right-click menu opens the Select File dialog, allowing you to select an image file.

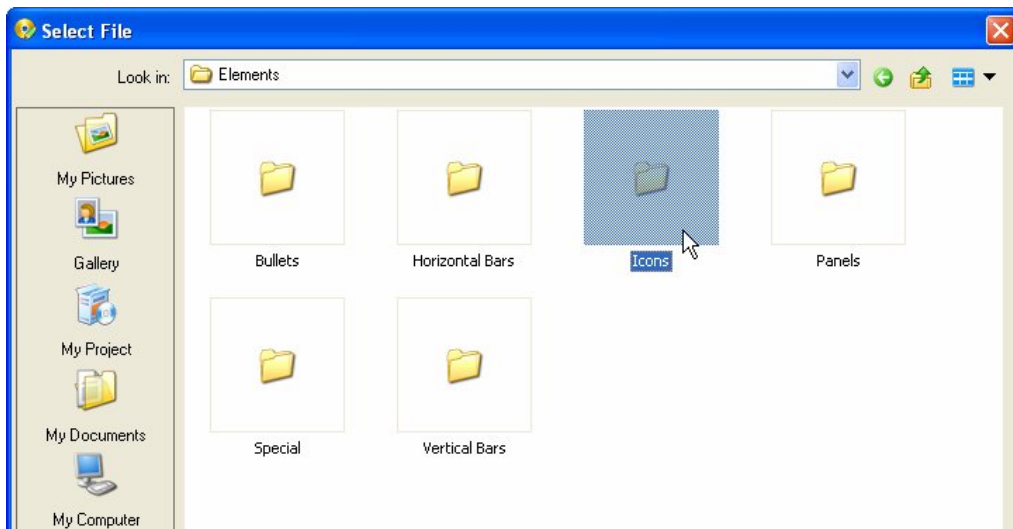
In this case, we want to temporarily add another image file to demonstrate how the arrange tools work. A good image for this purpose is 002BT15.png, located in the Gallery\Images\Elements\Icons folder. (Some of our image files have such romantic names.)

Since you're selecting a file for an image object, the Select File dialog automatically starts at the last location where you browsed for an image file. Unless you've browsed to a different location since we added the first image object in Lesson 2, the Select File dialog should open directly to the Gallery\Images\Elements\Special folder.



Click the move up button

The 002BT15.png file is located in the Gallery\Images\Elements\Icons folder. To get to the Icons folder, you need to click “move up” button to navigate one level up from the Special folder (which will put you in the Gallery\Images\Elements folder). Then, double-click on the Icons folder to navigate into it.

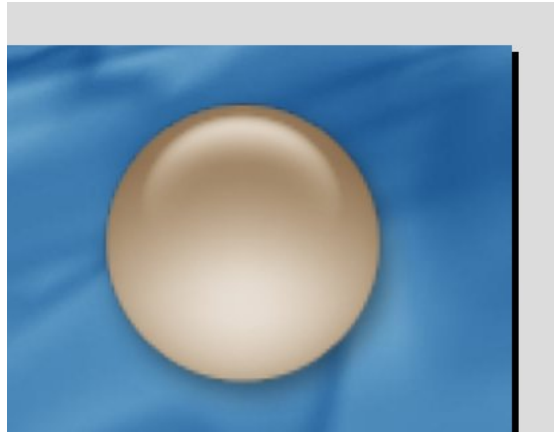


Double-click on the Icons folder

Once there, simply click on the 002BT15.png file, and click OK. The new image object will appear on the page.

4) Use the new image object to completely cover the logo.

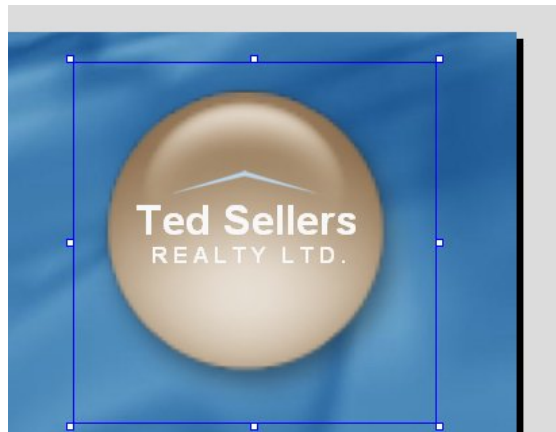
Using the mouse, resize the image object so it's big enough to cover the entire logo. Then drag it on top of the logo, so it covers all three of the logo's objects.



Tip: You can resize the object proportionally by holding the Shift and Ctrl keys down while dragging one of its resize handles.

5) Right-click on the image object and choose Arrange > Send to Back.

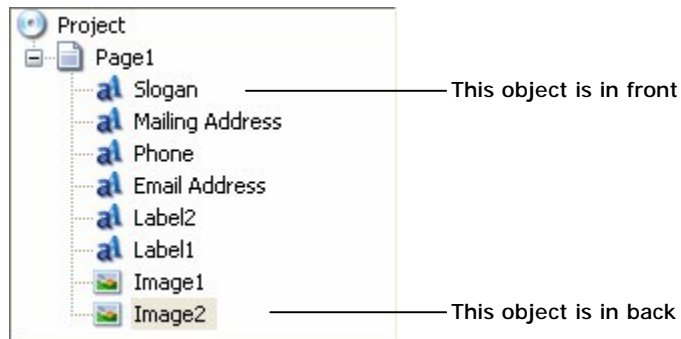
Abracadabra! The logo objects reappear.



Choosing Arrange > Send to Back moved the new image object to the very back of the z-order, putting it at the bottom of the “stack” of objects on the page. (Note that

this includes all of the objects on the page, and not just the four objects that happen to be overlapping.)

You can actually see the z-order in the Project Explorer. The objects in the project explorer tree are listed according to the z-order, with the “front” object at the top of the tree, and the “back” object at the bottom of the tree. When you sent the image object to the back of the z-order, it was moved to the bottom of the tree.

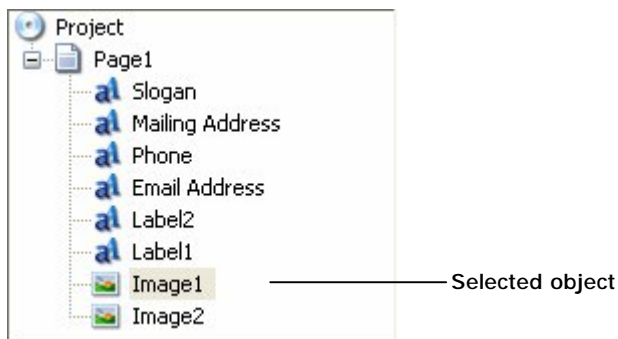


Tip: The other commands in the Arrange menu let you move an object forward or back by one “position” in the z-order stack, or send it straight to the front of the pile.

6) Choose Edit > Select > Next Object Above.

Choosing Edit > Select > Next Object Above selects the object that is one level “higher” in the z-order—in other words, the object that is one step further from the page.

Note: If you look at the project explorer, you can see which object you have selected and what position it occupies in the z-order.

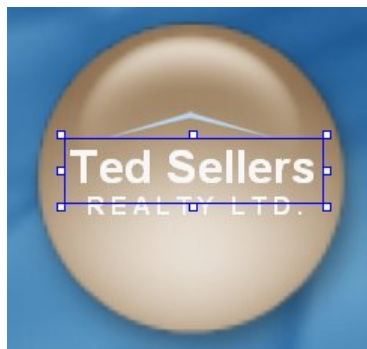


7) Press Shift+Tab.

You can also select the next object “higher” in the z-order by pressing Shift+Tab.

Tip: You can cycle through all of the objects on the page by pressing Tab and Shift+Tab. Pressing Tab selects one level “forward” in the z-order, and pressing Shift+Tab selects one level “back.”

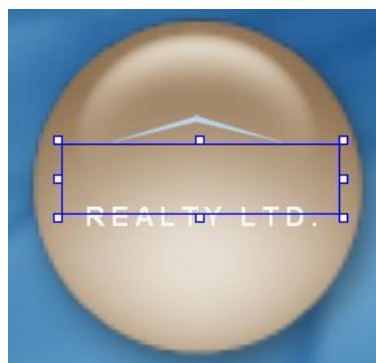
You should now have the Label1 object selected (the one with “Ted Sellers” on it).



8) Right-click on the Label1 object and choose Arrange > Send to Back.

Poof! The Label1 object is sent to the back of the z-order, and disappears behind the Image2 object.

Note that the Label1 object’s bounding box is still visible:

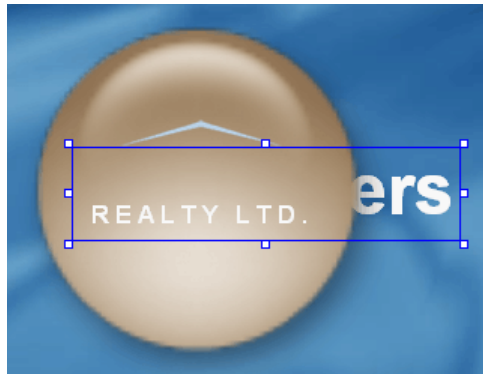


The bounding box is still visible because the Label1 object is still selected. When an object is selected, its bounding box is displayed in the foreground, on top of any objects that happen to be in the way. This gives you the same access to the object as

when it's "on top"—you can resize the label object by resizing the bounding box, or move it around by dragging it with the mouse, just like you would if it wasn't hidden.

9) Make the Label1 object larger by resizing its bounding box.

If you make the Label1 object large enough, it will become visible beyond the edge of the Image2 object.



Because the bounding box appears in the foreground, you can resize or reposition an object when it's selected, even if you can't see the object itself.

10) Click on the Image2 object, and choose Edit > Arrange > Bring Forward.

You have to click outside the Label1 object's bounding box in order to select the image object. Once an object is selected, clicking anywhere inside its bounding box won't deselect it, even if another object is in front.

When you bring the Image2 object forward one "step" in the z-order, the Image1 object disappears behind it. The only object from the logo that is still visible on top of the Image2 object is the Label2 object with "R E A L T Y L T D ." on it.

In the project explorer, the image object's name has moved one position higher in the list.



11) In the project explorer, drag the Image2 object above the Label2 object.

You can also arrange objects by dragging their names up or down in the project explorer. This is especially useful when you have many objects and you want to move one of them to a specific position in the z-order.

Tip: You can also arrange objects using four buttons on the Align toolbar.



12) Click on “Label1” in the project explorer.

Clicking on the name of an object in the project explorer selects the object, even if there’s another object in front of it.

Note: You can use the project explorer to select objects that are hidden behind other objects on the page.

13) Double-click on “Image2” in the project explorer. On the Settings tab, set the Opacity to 50, and click OK.

Double-clicking on an object’s name in the project explorer opens the Properties dialog for that object. It’s just like double-clicking on the object itself.

Changing the object’s opacity from 100 to 50 makes it 50% opaque (or, conversely, 50% translucent). In other words, it makes the object partially transparent.



Tip: The Opacity setting can be used to produce all kinds of neat effects. AutoPlay even lets you change an image object’s opacity *dynamically* at run time! (This can be done by using the Image.SetOpacity action which lets you set opacity “on the fly.”)

Getting Rid of Leftovers

No, this isn't an exercise in how to hide last night's overdone pot roast. Instead, it's a quick mention of a very handy tool that's built into AutoPlay: the *resource optimizer*.

If you're anything like me, you probably have to try a bunch of different files before you find the one that "fits" your project. You might add one image, decide it doesn't look quite right (chartreuse is so *passé* this year), and then go through five or six others until you find one that you're happy with (they say orange is the new blue).

Remember how AutoPlay keeps a local cache of everything, by automatically copying any files that you use into the project folder? Well, those files are left behind even after you delete the objects that caused them to be copied in the first place. After a while, those leftover files can begin to add up.

Which isn't a problem, exactly...your project will continue to work fine, even with those extra files inside it. They'll never actually *break* your project. However, they do take up space...and when you're trying to fit every last bit of data onto a CD, or if you're building a single-file executable that's going to be downloaded by half a kazillion people, it can truly pay to trim off any excess fat.

The resource optimizer is an easy way to get rid of all the files that have been left behind. It's like a hypersonic push broom sweeping through your project, clearing away the debris of the design process.

Our tutorial project hasn't been around long enough to accumulate much in the way of unused resources, but we do have at least one file that we aren't using any more. Or, we will once we delete the only object that uses it.

1) Select the Image2 object and press the Delete key.

You can select the Image2 object any way you want—click on it, use the project explorer, cycle through the objects on the page by pressing Tab, etc. The important thing is to make sure it's the only object that is selected, and then press the Delete key to remove it from the project. Bye-bye Image2!

2) Click on the Project pane, and navigate into the Images folder.

The Project pane should be tabbed with the Gallery and pane on the right side of the design environment. Clicking on the Project pane's tab switches to that pane.

The Project pane is like a window into the project folder. Within it you'll find all of the files that have been added to your project, neatly organized into sensible

categories. All of the image files that are added to the project end up in the Images folder. Once you navigate into that folder, you should see the 002BT15.png file (along with all the other image files that have been used so far).

Note that deleting the Image2 object, which was the only object in the project that used that file, didn't delete the file from the Images folder. Once a file has been added to the project, it's up to you to remove it. Of course, this is where the resource optimizer comes in.

3) Choose Tools > Optimize Resources. When the Remove Unused Resources dialog appears, make sure the Confirm deletion option is enabled. Click OK to begin the optimization process.

The Remove Unused Resources dialog lets you configure exactly which folders you want AutoPlay to search through—just in case there are some folders in the project that you don't want optimized.



Note: The Docs folder isn't searched by default, because that's where you normally put any "external" files that you want to distribute with your project. If you decide to optimize the Docs folder, be careful not to delete any files that you wanted to keep.

It's a good idea to leave the Confirm deletion option enabled, so you'll have a chance to override AutoPlay's choices before the files actually do get deleted.

As soon as you click OK, the resource optimizer will begin scanning the project directories, looking for files that aren't referenced anywhere in the project.

4) When the Confirm Delete dialog appears, verify the list of files that AutoPlay wants to delete, and click OK to delete them.

The resource optimizer is very, very fast, so the Confirm Delete dialog will appear almost instantly.

Clicking OK will delete all of the files in the list that have check marks next to them. Make sure you examine the list carefully—once a file is deleted, you can't get it back.

5) Observe that the file has been removed from the Project pane.

As you can see, the 002BT15.png was successfully removed in the previous step.

Tip: Use the resource optimizer to clean up any unnecessary files before publishing your project.

Now that we've covered optimizing and arranging, let's get things back to the way they were before we set off on this little detour.

6) Choose File > Revert to go back to the last save point.

When asked to confirm that you want to proceed, click Yes.

You should end up with the image object and the two label objects back where they were before you centered them vertically on the page.



Choosing File > Revert puts everything in the project back to the way it was the last time you saved the project. This includes all of the object settings and project settings that were saved with the project.

However, it does *not* include any changes to the *files* that your project references. In other words, if the files themselves are deleted or modified, choosing File > Revert will not undo those changes.

If you delete a file that your project used the last time it was saved, choosing File > Revert will not restore the file. It will restore any settings that refer to that file, but not the file itself. (If any objects need to display that file, they would display a placeholder message instead, alerting you that the file is not there.)

Tip: The File > Revert feature allows you to undo any number of changes, regardless of the current Undo/Redo level.

Grouping Objects

Now, what if you wanted to center the logo in the middle of the page, without having any of its objects overlap? In order to do that, you need to *group* the three objects together. Grouping the objects will allow them to be aligned as though they were a single object.

Objects that are grouped maintain their positions relative to one another when you use the alignment tools on them. They essentially act as though they were a single object, allowing you to align them as a group.

1) Select the three objects in the logo, and choose Edit > Group to group them together.

When you group the objects together, their bounding boxes turn green instead of blue. This is just to let you know that the objects are part of a group. (If it wasn't for the green bounding boxes, it would be impossible to tell they were grouped just by looking at them.)

Note: You can *ungroup* objects by selecting a group and choosing Edit > Ungroup.

2) Click on the page surface to deselect the objects, then click on one of the objects in the logo to select the group.

Grouping objects makes it easy to access them all at once. That might seem a bit obvious, but it's a really useful feature that can save you a lot of time. When you click on an object that is part of a group, all of the objects in the group are selected automatically.

Tip: If you find yourself always wanting to select and work on a particular bunch of objects, just group 'em. You can always ungroup them later if you want to work on them individually again.

3) Make sure you're working in Align to Page mode.

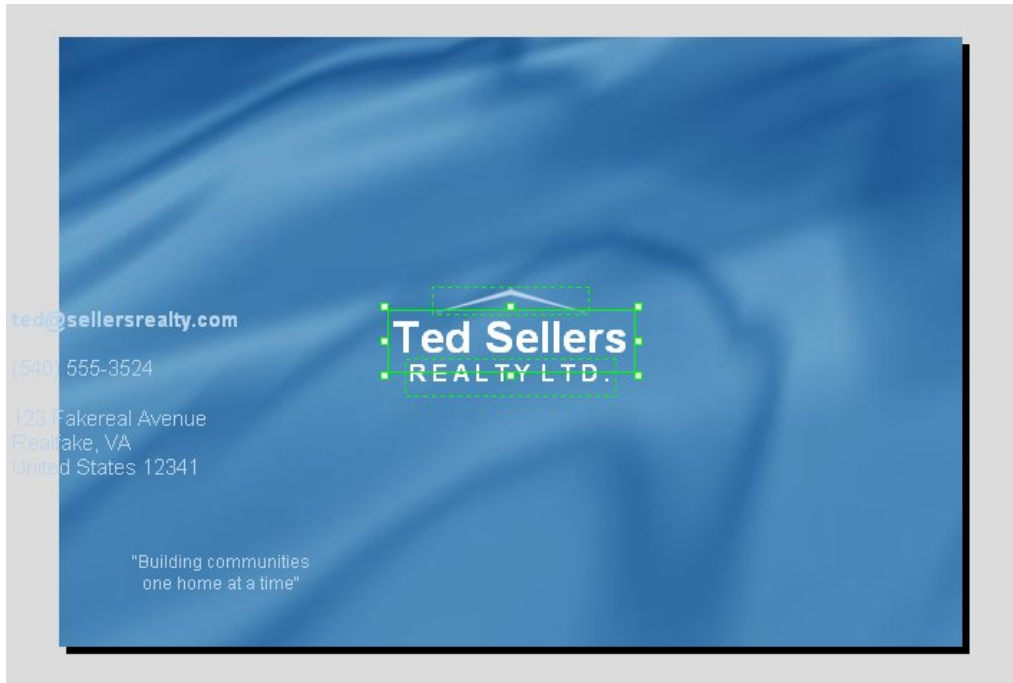
When you're working in Align to Page mode, the Align to Page button will be highlighted, like this:



If it isn't, either click on the button, or choose Align > To Page to toggle the feature back on.

4) Right-click on one of the objects and choose Align > Center Vertical.

Voila! This time, the whole group is centered on the page, with the three objects maintaining their positions relative to one another.



Whenever you align a group to the page (or to other objects, for that matter), the entire group is positioned as though it were a single object.

Note: A group is just a bunch of objects that are always selected together, and that maintain their positions relative to one another when you use the alignment tools on them. It's *almost* like the group becomes a single object.

5) Click on “Label1” in the project explorer. In the Position category of the properties pane, set Left to 346 and Top to 121.

In the default workspace layout, the project explorer is on the left side of the design environment, above the properties pane. The project explorer contains a list of all the pages, objects, etc. on the current page. You can use the project explorer to select objects by name.



When you click on “Label1” in the project explorer, the Label1 object is selected by itself. This is because the project explorer lets you select individual objects even if they’re part of a group.

This can be helpful when you want to change the settings of a single object within a group. By selecting the object in the project explorer, you can make the changes without having to ungroup the objects and then group them back together again when you’re done.

Note: Selecting a grouped object in the project explorer doesn’t remove that object from the group...it just selects the object on its own, without selecting the other objects in the group.

Note that the Label1 object’s bounding box is still green to remind you that it does belong to a group, even if the other objects in the group are not selected.

6) Click on “Image1” in the project explorer. In the Position category of the properties pane, set Left to 375 and Top to 102.

Once again, using the project explorer to select the object by itself lets you change its position without affecting the other objects in the group.

7) Click on “Label2” in the project explorer. In the Position category of the properties pane, set Left to 358 and Top to 156.

While we’re at it, we might as well move the Label2 object as well.

Changing the Left and Top values is the only way to move a grouped object without moving the other objects in the group. Even if you selected the object in the project explorer, moving the object with the mouse would move the entire group.

8) Click on the Image1 object (on the page).

When you click on the image object, the whole group is selected again.

Note: Clicking on a grouped object on the page selects all of the objects in the group.

Pinning Objects

Once you have objects placed where you want them, sometimes it’s nice to pin them so they can’t be moved or resized accidentally.

1) Right-click on one of the grouped objects and choose Pin.

This fixes the position and size of the three objects in the group. Note that when you pin an object, its bounding box turns red.



Pinned objects are surrounded by red bounding boxes by default

Note that the bounding box for the dominant object doesn’t have any resize handles on it any more, since you can’t resize pinned objects with the mouse.

2) Try to move the objects.

You might need both hands on the mouse to get them to move...

Heh, just kidding. You can’t move or resize objects when they’re pinned.

To unpin the objects, you would either right-click on one of them and choose Pin again, or choose Edit > Unpin to unpin all of the selected objects on the page at once.

For now, though, just keep the objects pinned so you don't have to worry about moving them by accident.

Distributing Objects

Distributing objects just means adjusting their positions so that they're spread out evenly between two points, or across the page. It's an incredibly useful technique for getting things placed neatly, with equal distances between the objects.

1) Make sure the Align to Page mode is off.

When Align to Page mode is on, the Align to Page button appears selected, like this:

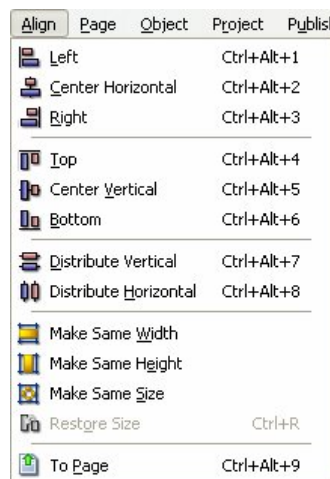


You want the button to be unselected, like this, instead:



If you need to toggle it off, just click the button, or choose Align > To Page.

You can also check the Align to Page mode by looking at the icon in the Align menu.

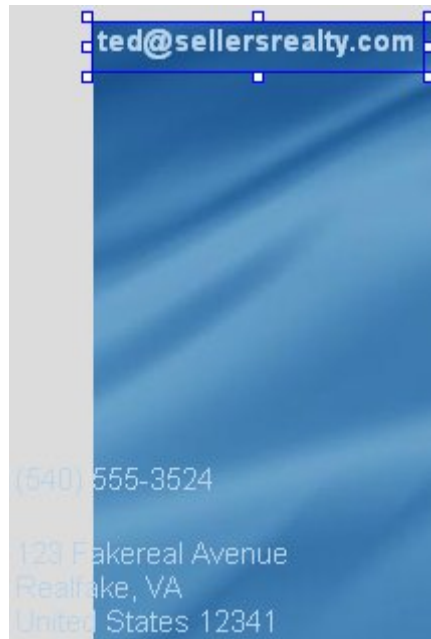


2) Drag the Email Address label object up to the top of the page.

You may notice that the object “snaps” into place as you move it near the upper left corner of the page. This is because AutoPlay’s snap-to-page feature is enabled by default. (You can toggle this feature on or off by choosing View > Snap to Page.) The snap-to-page feature causes objects to automatically line up with the page edge when you move them within a few pixels of it.

Tip: You can hold the Shift key while moving objects to suppress snapping.

The Email Address, Phone, and Mailing Address label objects should be positioned vertically, with the Mailing Address and Phone label objects close together, and the Email Address label object off by itself.



Don’t worry about lining things up carefully; you’ll use the alignment tools to take care of that later. (It’s actually easier to see what’s going on if the objects aren’t lined up so well.)

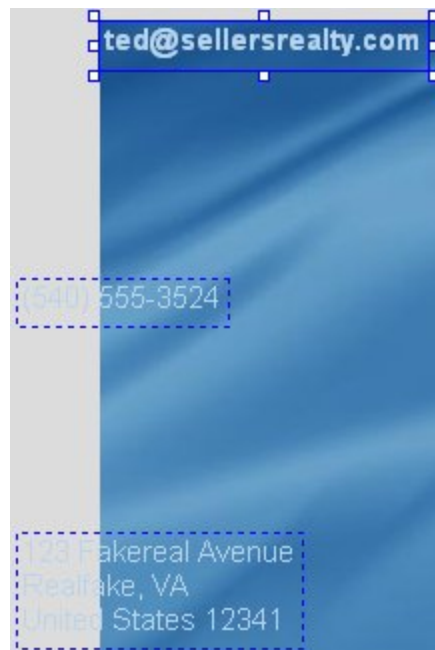
3) Select the Email Address, Phone, and Mailing Address label objects. When you have all three objects selected, choose Align > Distribute Vertical.

To select the three objects together, either drag-select them, or just click on one and then ctrl-click on the other two.

Tip: You can also select multiple objects by clicking and ctrl-clicking on their names in the Object Browser pane (View > Panes > Object browser).

Choosing Align > Distribute Vertical distributes the three objects evenly between the top and bottom ones, so there is an equal distance between all three of them.

In this case, the top and bottom objects stay put, and the Phone object (the one in the middle) moves up so it's placed exactly in between.



4) Move the bottom object up so it's right underneath the middle one.

In other words, move the Mailing Address label object right up under the Phone label object.

Note: You'll need to select the object on its own in order to move it.



5) Select all three objects and choose Align > Distribute Vertical again.

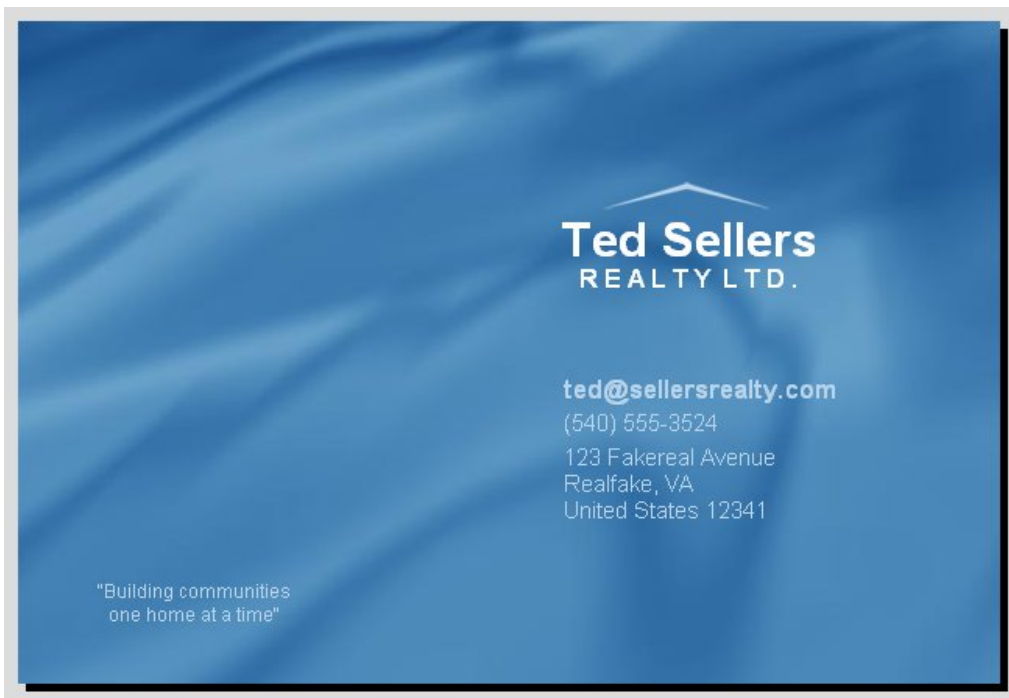
Once again, the middle object moves so the selected objects are spaced evenly apart.



6) Select the Email Address object by itself and use the properties pane to increase its font size to 13.

We want the email address to stand out a bit from the other label objects. Making the email address a bit bigger should do the trick.

7) Move the three label objects closer together, and then drag them down below the logo. Use the alignment tools to get the label objects spaced evenly apart and aligned along their left sides.



Start off by eyeballing it, moving the objects closer together and dragging them down to the lower right side of the page, using the above image as a guide.

Once you have the objects more or less in place, select all three label objects and use Align > Left to align them along their left edges. Then use Align > Distribute Vertical to even things up.

Locking Objects

Earlier I showed you how to pin objects so they can't be moved or resized accidentally. Pinning objects prevents the objects from being moved or resized with the mouse, but still lets you access their properties and align other objects to them.

Sometimes, though, it would be nicer if the objects would just get out of the way. For example, if you have a large object on the page with other objects on top of it, or lots of objects in close proximity, it can be difficult to drag select specific objects without accidentally selecting some you don't want to. You could ctrl-click on all the objects that you want to select, one at a time, but it would be better if you could just drag select the objects you're working on and have any "finished" objects be ignored.

This is where locking objects comes in. When you lock an object, it becomes completely unselectable. It's like the object becomes part of the background. you can click on it or even drag-select around it, but it won't become selected at all. It's almost like it isn't there.

Note: Locking an object has no effect on how the object behaves at run time. The user will still be able to interact with the object the same way, whether it's locked or not. It only affects how the object behaves while you're working on the project.

If you're wondering how you would ever get the object "back," don't worry...you can unlock an object by right-clicking on it and choosing Lock from the right-click menu.

To illustrate this, let's lock all of the objects on the page, except for one label object that you will be working with in another lesson.

1) Press Ctrl+A to select all of the objects on the page.

When you press Ctrl+A, all of the objects on the page are selected.

Note: This is the same as choosing Edit > Select > All.

2) Choose Edit > Lock.

All of the objects become locked. Since locked objects can't be selected, their bounding boxes disappear.

3) Try to select one of the objects.

Go ahead. Bet you can't select any.

No, really...you can't. Locked objects don't respond to left mouse button clicks at all. You can't even select them using the project explorer.

It's like the objects aren't even there. But they still are, of course...they're just locked.

4) Right-click on the Email Address label object and choose Lock to unlock it.

When you right-click on a locked object, a menu appears so you can unlock it. Note that there is a check mark next to the Lock command to indicate that the object is currently locked.



Choosing Lock from this menu toggles the object's "locked" status off—in other words, it unlocks the object. It also automatically selects the object for you, so you can see that it has been unlocked.

Tip: You can unlock all of the locked objects on a page at once by choosing Edit > Unlock All.

5) Choose File > Save to save the project.

Be sure to save the project so the work you've done in this lesson isn't lost.

Lesson 3 Summary

In this lesson, you learned how to:

- Select multiple objects
- Move more than one object at a time
- Change the settings of several objects at once
- Align objects to each other
- Align objects to the page
- Arrange objects in the z-order
- Remove unused files from the project
- Group objects together
- Pin objects (so they can't be moved or resized)
- Distribute objects evenly
- Lock objects (so they stay out of your way)

Lesson 4:

Buttons, Actions and Pages

This lesson is all about interactivity. Interactivity is important—it lifts your AutoPlay application above the limitations of mere pamphlets and traditional business cards that dwell in the lowly realm of paper. Interactivity makes your application seem alive and responsive, and lets it actually perform important tasks for the user, just like any other Windows program.

There are several ways to make your applications interactive, but the three most important tools at your disposal are buttons, actions and pages. Buttons are one of the coolest features of AutoPlay. They look great, are easy to use, and instantly give your application a truly professional polish. Actions, and the events that you add them to, let you make your application *do* stuff. Pages are the surfaces that you put buttons and other objects on. And since your imagination shouldn't have to fit on one page, we'll show you how to add more pages to your project.

4

What You'll Learn

In this lesson, you'll learn how to:

- Add interactive buttons
- Match the width and height of several objects at once
- Change the appearance of the button text
- Duplicate multiple objects
- Add simple actions like `Application.Exit` and `File.Open`
- Add a blank page to the project
- Duplicate an existing page
- Add navigation buttons using `Page.Jump` actions
- Copy objects from one page to another
- Send email with a Quick Action

How Long Will It Take?

This lesson takes approximately 30 minutes to do.

Starting the Lesson

If you're continuing from Lesson 3, you should still have AutoPlay running with the Tutorial project open. If so, you're ready to move on to the next exercise: Adding Buttons.

Otherwise, you'll need to open the project file that you saved at the end of Lesson 3.

1) Open the Tutorial.am7 file that you saved in Lesson 3.

When you save a project, AutoPlay automatically creates a project folder for it inside your "My Documents\AutoPlay Media Studio 7.0\Projects" folder. This project folder is where everything that belongs to the project is stored—including the project file, which contains all of the settings used in the project.

The project folder and the project file always have the same name that you gave to the project when it was created. The project file's name ends with a ".am7" file extension.

Since you named this project "Tutorial" in Lesson 1, the name of the project folder will be Tutorial, and the name of the project file will be Tutorial.am7. So, the path to the project file should be something like:

```
...\My Documents\AutoPlay Media Studio 7.0\Projects\Tutorial\Tutorial.am7
```

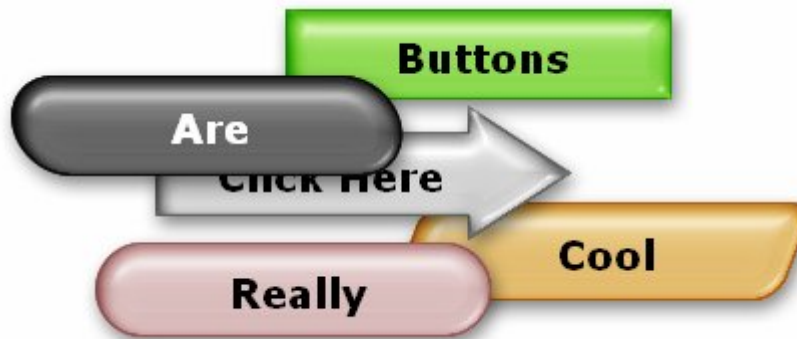
To open the project, you just need to open that project file.

Adding Buttons

Buttons are special *interactive* objects. They respond automatically to the user by changing their appearance when you move the mouse over them or click on them. In fact, each button has four different appearances, or “states,” built into it: Up, Down, Highlight, and Disabled.

There’s a different image for each state built right into the button file. By switching between these images, a button can appear animated and interactive—glowing when the user moves the mouse over it, for instance, and appearing pushed in when the user clicks on it. With full support for alpha transparency and variable opacity, buttons can even come with built-in drop shadow effects and have smooth, rounded edges.

Buttons are a great place to put actions, since they usually look like something you would want to click on. And to help the user know what the buttons will do, each button can have custom text on it, just like a label object. The text can even change color in each of the different states, so it fits in with each state’s appearance perfectly.



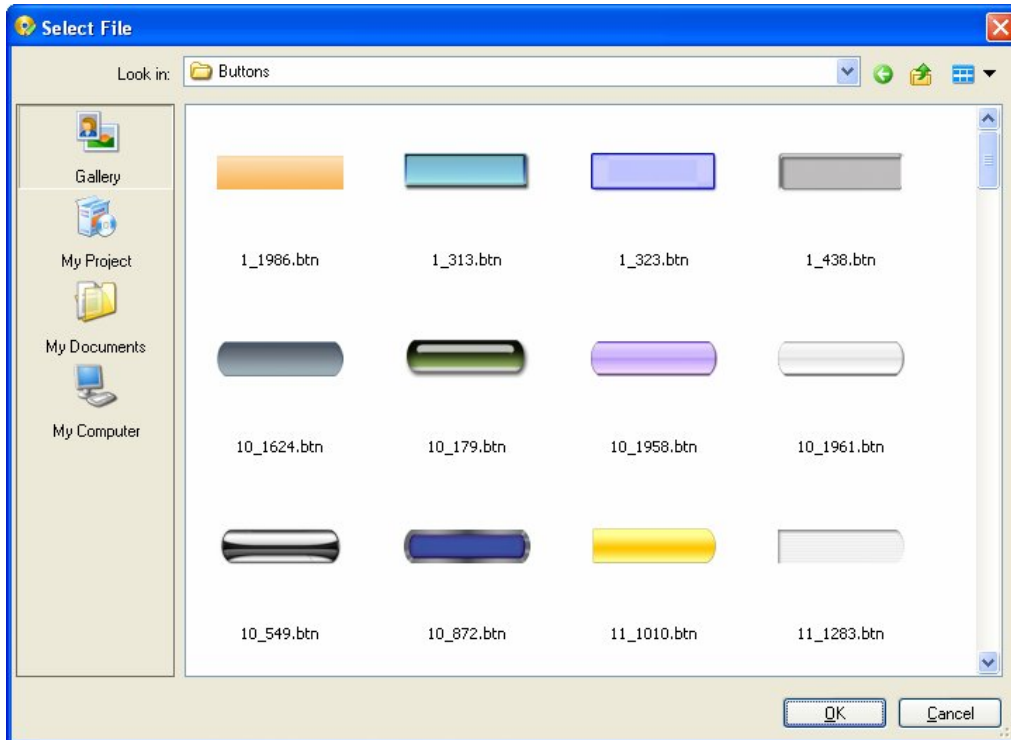
Buttons come in all shapes, sizes and colors. The professionally-designed buttons that come with AutoPlay resize well and even look great when they’re stacked on top of each other, thanks to the built-in drop shadows which give our buttons a nice, three-dimensional appearance.

In short, buttons rock.

Tip: If you need more buttons, you can purchase add-on packs from Indigo Rose. Or just build your own buttons using the AutoPlay Media Studio Button Maker, which you can access by choosing Tools > Button Maker.

1) Choose Object > Button to add a new button object. When the Select File dialog appears, click the Gallery button.

Choosing Object > Button opens the Select File dialog so you can select a button file to add.



Tip: You can also add a button object by clicking the New Button Object button.



2) Select the grey_rounded.btn file and click OK.

To select the grey_rounded.btn file, just find it in the list of button files and click on it. (You will have to scroll the list to find it. As you're scrolling, have a look at the other buttons that are included. There are a lot of cool buttons in there.)

When you click OK on the Select File dialog, the dialog closes and the button is added to the project. Like all objects that you add in this way, it starts out positioned in the upper left corner.



The button looks a bit different than it did in the Select File dialog, though. Now it has the words “Click Here” on it. This is just the button’s default text.

Note: You’ll usually want to change this text to something a bit more meaningful, like “Play Video” or “Install WidgetMaster 2.0”—in other words, something that describes what clicking on the button will actually do.

We’ll change this button’s text when we get to step 4, but for now the default text is okay.

3) Resize the button so it’s 196 pixels wide and 63 pixels tall. Position it 14 pixels from the left of the page, and 89 pixels from the top.

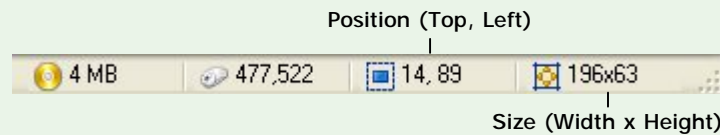
To resize the button object, you can either drag the resize handles, or set the width and height directly in the Position category of the properties pane.

To position the object, either move it into place by dragging it, or edit its Top and Left settings directly.

A screenshot of a software interface's properties pane. The pane is titled "Position" and contains a table with four rows: "Left", "Top", "Width", and "Height". Each row has a numerical value in the second column. The "Top" row is currently selected, indicated by a blue highlight and a small up/down arrow icon to its right.

| Position | |
|----------|-----|
| Left | 14 |
| Top | 89 |
| Width | 196 |
| Height | 63 |

Tip: If you prefer the drag method, you can use the position and size readouts on the status bar to set the object’s size and position precisely.



Now let’s change the text to something that better describes what this button does. (Or in this case, what the button’s *going* to do, once we add an action to it later in this lesson.)

4) In the properties pane, change the Text setting to *Ted Sellers Online*.

The Text setting is found inside the Object category. This should be the first category at the top of the properties pane.

Note: If the Object category isn’t visible, then the button object isn’t selected—in that case, click once on the button object to select it.

To change the text that is displayed on the button object, highlight the contents of the Text setting and type the new text in. (In this case, highlight the words “Click Here” and type in the words “Ted Sellers Online” instead.)

Tip: A quick way to select all of the text in a setting is to double-click on the name of the setting in the left-hand column of the properties pane. In this case, that means double-clicking on the word “Text” itself. Whenever text in a field is highlighted, anything you type will instantly replace the highlighted text.

When you’re done typing the new text in, press Enter.

Note: Don’t worry if the text is too big to fit on the button...we’ll change the button’s font size in a moment.

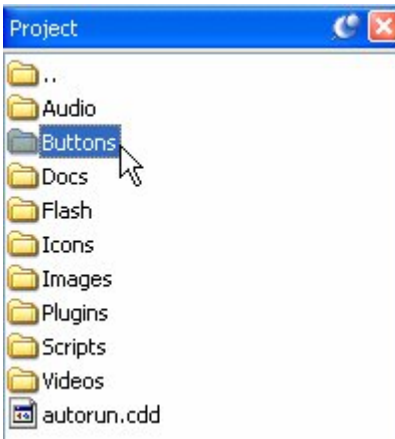
(Later in this lesson, we’ll add an action to this button to make it open Ted’s website.)

5) Use the Project pane to add another `grey_rounded.btn` button.

In Lesson 2, you opened the Project pane by choosing View > Panes > Project Browser. It should still be tabbed on the right side of your screen, together with the Gallery pane. In that case, all you need to do is click on the Project pane’s tab to bring it to the front.



If the Project pane isn't visible anywhere on your screen, choosing View > Panes > Project Browser will toggle it on. (There will be a check mark next to Project Browser in the View > Panes menu when the Project pane is visible. If that check mark is already there, it means the Project pane *is* actually somewhere on the screen already.)



Double-click on the Buttons folder

Double-click on the Buttons folder in the Project pane. This will show you a list of all the button files that have been added to your project so far. Since you've only added one kind of button, there should only be one item in this list: the `grey_rounded.btn` file that you added earlier.



Selecting the button file

Note: This was explained more fully in Lesson 2, but here are the basics to refresh your memory. Every project you create has its own folder, called the project folder. Each project folder is divided into subfolders. When you add a button to your project, a copy of the file is placed inside the "Buttons" subfolder. You can use the Project pane to navigate to this subfolder.

When you click on the `grey_rounded.btn` file on the Project pane, a fully working preview of the button appears on the Preview pane. (If the Preview pane isn't visible, use the View > Panes > Resource Preview menu item to toggle it back on.)

You can use the Preview pane to see how the button will respond to the mouse in your completed application.



Trying it out in the Preview pane

Move the mouse over the button in the preview area. See how it changes its appearance slightly? This is the button's "highlight" state. (Buttons become "highlighted" when you move the mouse over them.)

Click on the button in the preview area, and it changes its appearance again. This time, you're seeing the button's "down" state. (Buttons are typically pressed "down" when you left-click on them.)

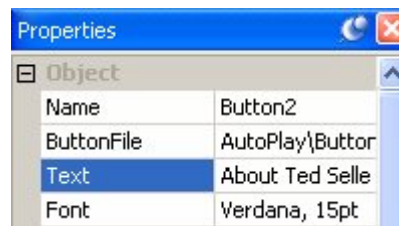
Move the mouse away from the button in the preview area, and it returns to normal. This is the button's "up" state.

Note: The Preview pane allows you to try a button before you add it to your project.

To add the button to the page, just drag the `grey_rounded.btn` file from the Project pane onto the page. A new button object will be placed at the spot where you drop the file. Move the new button object to just below the first one.

Tip: You can also drag a button directly from the Preview pane.

6) In the properties pane, set this button's text to *About Ted Sellers*.



When you change the Text setting, “About Ted Sellers” appears on the button object. (Later on, we’ll make this button jump to another page that has information about Ted on it.)

Matching the Width and Height

The new button object that you added doesn’t have the same width and height as the first one. This is because the new object still has its default width and height—it hasn’t been resized yet like you resized the “Ted Sellers Online” button.

You could resize the “About Ted Sellers” object the same way you resized the “Ted Sellers Online” button, but there’s an even faster way to do it. Using the alignment tools, you can get one object (or many objects!) to match another object’s width or height in no time flat.

1) Select both buttons. Click on the “Ted Sellers Online” button to make it the dominant object in the selection.



The first step in matching an object’s width or height is to select the objects that you want sized the same, with the object that is already sized correctly as the dominant object. We want the size of the new “About Ted Sellers” button to match the size of

the “Ted Sellers Online” button, so the “Ted Sellers Online” button needs to be the dominant object.

2) Choose Align > Make Same Width.

This makes the “About Ted Sellers” button the same width as the “Ted Sellers Online” button...

3) Choose Align > Make Same Height.

...and now the “About Ted Sellers” button is the same height as the “Ted Sellers Online” button, too.

Tip: You can also just choose Align > Make Same Size to match the width and height together in one step.

Changing Text Settings

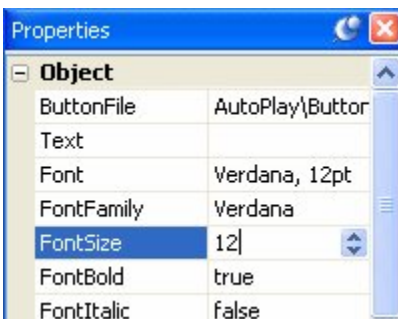
Much like label objects, button objects give you a great deal of control over the appearance of the text that appears on them. You can change the font family and size, make the text bold or italic, and also change the color.

In fact, each button has a different text color for each state the button can assume. This allows the text color to change as the appearance of the button changes. Not only does this contribute to the interactive appearance of the object—making the text brighten as the mouse moves over the button, for example—but it allows the text color to be adjusted to complement the changing color of the button itself. (Black text on a blue button would disappear if the button’s down appearance was also black.)

Let’s change the text settings on these buttons a bit so the text will change color when the user interacts with the button.

1) With both objects still selected, change the font size to 12, and the font family to Arial.

The default font size of 15 points is a bit large for these buttons. To change the size, change the FontSize setting from 15 to 12.

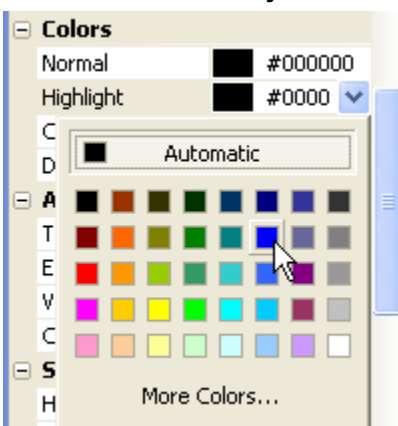


Changing the font size for both buttons at once

Verdana doesn't look that good at 12 points, so let's change the font to Arial. To change the font family, just click on the Family setting, click the select button, and select Arial from the list.

Note that, since two button objects are selected, the properties pane only shows the settings that can be changed for both objects at once.

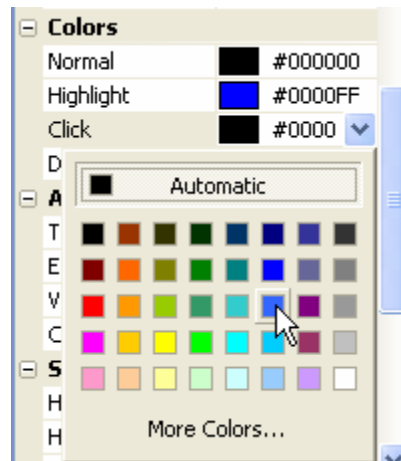
2) Change the Highlight color for both objects to Blue (#0000FF).



This is the text color that will be used for the button's Highlight state, which is shown when the mouse pointer moves over the button.

You won't be able to see this effect until you actually see the project running, though; while you're working on the project, the button is always in its Normal state, so the words "Ted Sellers Online" will still be black.

3) Change the Click color for both objects to Light Blue (#3366FF).



This text color will be used when the button is in its Click state, which is shown when the left mouse button is clicked on the object.

4) Choose Publish > Preview. When your application opens, move the mouse over the buttons and click on them.

Hey, it works! Each button changes its appearance and text color as you interact with it. Pretty cool, huh?

Note: If you have your computer speakers turned up, you may hear sound effects as you move the mouse over the buttons and click on them. I'll show you how to change these default sounds in Lesson 8. In the meantime, don't be freaked out by the beeps and clicks. (If you hear voices, though, you have my permission to be as freaked out as you want to.)

5) Exit from the preview.

To exit from the preview application, either click the close button on the application's title bar, or press Alt-F4 while the application is selected.

When you exit from the application, you should be returned to the AutoPlay design environment. If not, use Alt-Tab to bring AutoPlay back to the foreground.

Duplicating Objects

Once you have a couple objects set up the way you want, it's easy to add more objects with the same exact settings. Just duplicate 'em!

1) Select both button objects, and either choose **Edit > Duplicate**, or press **Ctrl+D**.

Both objects are duplicated instantly. Welcome to button cloning! Note that you can easily duplicate multiple objects at once.

The new button objects have the same settings as the originals, with only two exceptions: the names are different (because they have to be), and they're positioned a bit down and to the right (so it's easier to see the new objects).

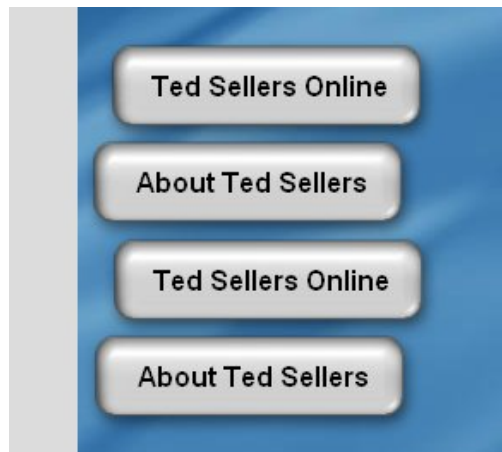
You can't see the object names in the properties pane at the moment—object names are hidden when you have multiple objects selected. But they *are* different.

Note also that the new button objects are selected, and the originals aren't. This makes it easy to move the new objects somewhere else—you don't have to bother with deselecting the old ones first.

Tip: When you need to add lots of similar objects, you can keep pressing Ctrl+D to add more and more “clones” to the page.

2) Move the new buttons below the two originals.

Just drag the two objects down together so the four buttons form a sort of column on the page.



3) Change the text of the third button to *Video Presentation*.

Before you change the text on the third button, you need to deselect the other button first. Otherwise, you'll end up changing the text on both buttons.

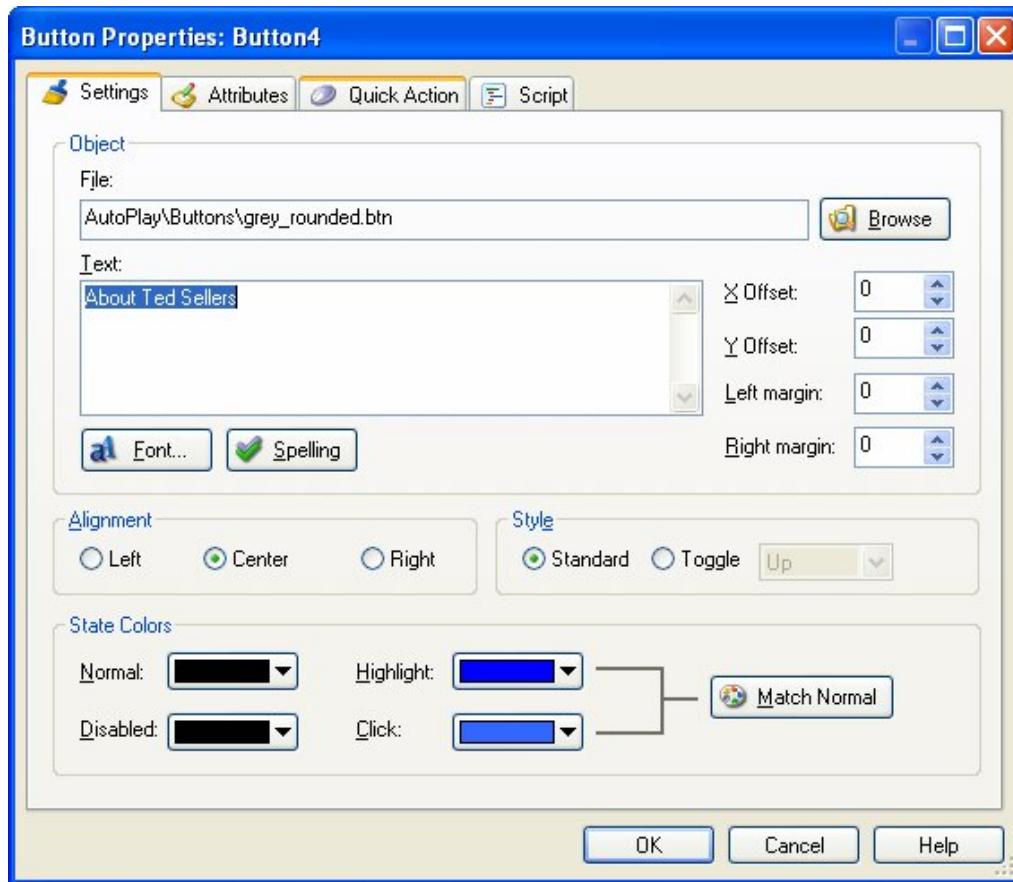
To deselect the other (fourth) button, just ctrl-click on it. This will remove it from the current selection, leaving only the third button selected. Or, if you prefer, you can click on the page surface to deselect everything, and then click on the third button (Button3) to re-select it alone.

Once you have Button3 selected by itself, change the Text setting from "Ted Sellers Online" to "Video Presentation" by double-clicking on it and changing the text field.

This button will be used to jump to a page with a short video on it. (We'll add the action to perform this magic later.)

4) Double-click on the bottom button, type *Exit*, and click OK.

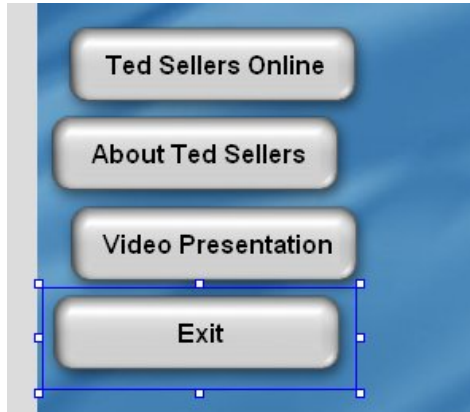
Double-clicking on a button object opens the Button Properties dialog and automatically highlights all of the text in the Text setting for you.



When you type in the word *Exit*, it replaces the existing text.

Tip: A quick way to change the text in a button object is to double-click on it, type in the new text, and click OK.

As you've probably guessed, this button will be used to exit the application.



Lining Them Up

Before we add any actions to these buttons, let's line them up nicely on the left side of the page.

1) Make sure the top button (Button1) is still positioned at 14, 89.

To confirm an object's position, just select it (by clicking on it) and either look at the position readout on the status bar, or look at the Left and Top settings in the Position category of the properties pane.

Note: "14, 89" is shorthand for 14 pixels from the left, and 89 pixels from the top.

2) Move the bottom button (Button4) to 14, 272.

To move the "Button4" button object, either drag it into place (using the position readout on the status bar as a guide), or set its Left and Top settings to 14 and 272, respectively.

3) Select all four buttons. Right-click on the top button object (Button1) and choose Align > Left.

Right-clicking on the top button object will make it the dominant object. When you choose Align > Left, all four buttons are lined up with that object.

4) Right-click on the top button object and choose Align > Distribute Vertical.

When you choose Align > Distribute Vertical, the two middle objects will be repositioned so there is an equal distance between all four of the objects.



Adding Simple Actions

Now let's make the buttons do something.

In AutoPlay, getting an object to do something means adding an *action* to one of the events that the object can respond to.

Events are just things that can happen when your application is running. For example, most objects have an On Click event, which is triggered when the left mouse button is clicked on that object. To make something happen when the object is clicked, you simply add an action to its On Click event.

Actions are just commands that tell the application to do something. There are actions to do all sorts of things, like changing the text in a label object, running external program files, telling a video object to start playing, or jumping to another page.

Quick Actions are special "easy-to-use" actions that take care of simple tasks without any scripting. For example, most objects let you configure a single Quick Action that will be performed when the object is clicked. Adding a Quick Action is an easy way to make something happen when you click on an object.

Every object has its own events, and each event can have its own actions. The actions that you add to an object's event are only performed when that specific event is triggered—in other words, events are *object specific*. This means that your application can do completely different things when the user clicks on different objects. For example, you could make clicking on one object start playing an audio file, while clicking on another object makes the application “jump” to another page. Each click would trigger the specific object's “On Click” event, causing that event's list of actions to be performed.

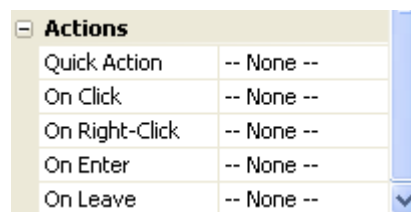
1) Click on the page surface, then click on the “Exit” button (Button4).

In order to add an action to the “Exit” button, you need to have that button object selected by itself. Clicking on the page surface deselects all of the objects, so you can select the “Exit” button object on its own.

2) Make sure the Actions category is open in the properties pane.

The Actions category is the last category in the properties pane. When you have an object selected, all of the places where actions can be defined for the object are listed in this category.

Tip: You might need to scroll to the bottom of the properties pane in order to see all of the items in the Actions category.



These places generally correspond to the events that the object can detect and that you can respond to. (The lone exception is the Quick Action, which is essentially a simpler version of the On Click event.)

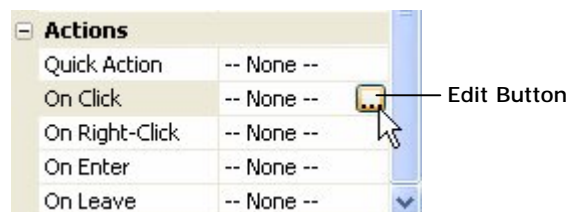
It looks like this button object has five places where actions can be defined:

- Quick Action, which is where you can choose from a list of the most common responses to a mouse click on the object and “program” it using a streamlined configuration interface
- On Click, which is triggered when you click on the button
- On Right-Click, which is triggered when you right-click on the button
- On Enter, which is triggered when the mouse moves onto the button (when the mouse pointer “enters into the object’s space”)
- On Leave, which is triggered when the mouse moves off of the button (when the mouse pointer “leaves the object’s space”)

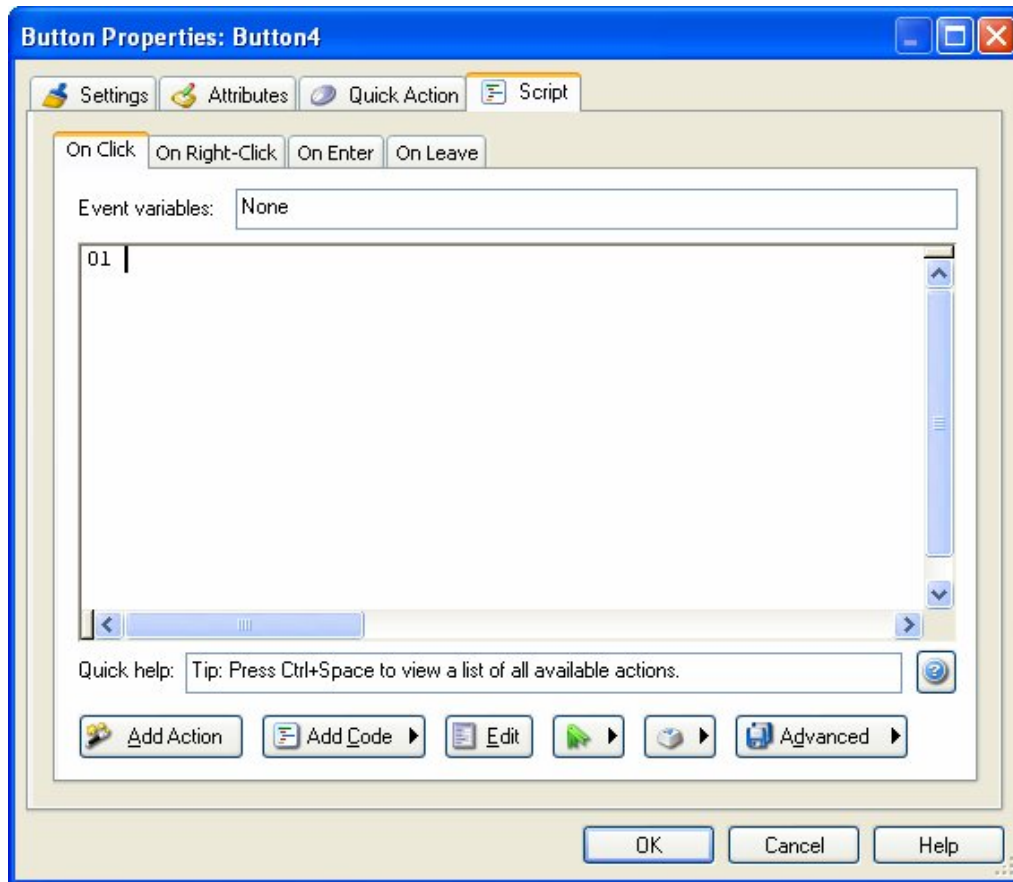
Note: Any transparent parts of the object are ignored as far as On Enter and On Leave are concerned. The mouse has to actually enter or leave a non-transparent part of the button for the event to be triggered.

Right now, all six items have “-- None --” next to them, which indicates that no actions will be performed when these events are triggered for this object.

3) Click on the On Click setting, then click the edit button to bring up the script editor.



Clicking the edit button opens the script editor.



The script editor is where you can add actions to each of the object's events. There are four tabs on the script editor—one for each of the events that this button object can respond to.

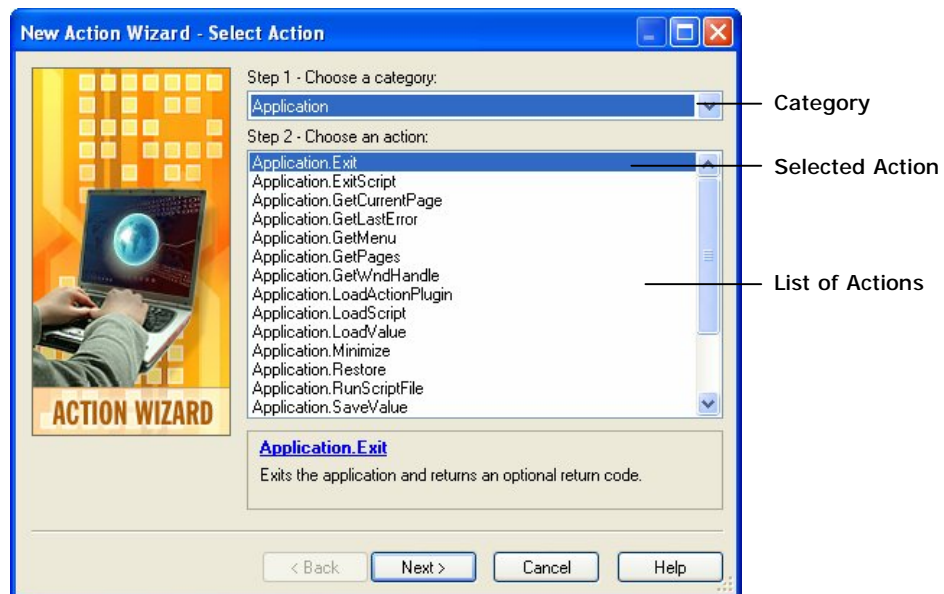
Note: There is always one tab on the script editor for each event an object can respond to. If you select an object that has five events, there will be five tabs in the script editor.

When you click the edit button in the properties pane, you're automatically taken to the corresponding tab on the script editor. For instance, since you clicked the edit button for the On Click setting, the script editor opened directly to the On Click tab. If you had clicked on the On Leave setting instead, and then clicked the edit button, the script editor would have opened to the On Leave tab.

4) Click the Add Action button. When the New Action wizard appears, switch to the Application category and then click on the action called Application.Exit.

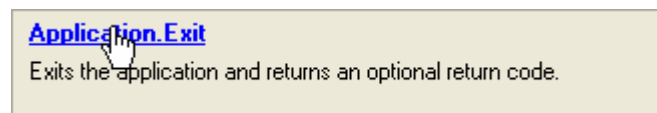
The New Action wizard will walk you through the process of adding an action to the script editor. The first step is to choose a category using the drop-down list. (In this case, it isn't really necessary, since the action we want to add is already visible in the default "All Actions" category. But it doesn't hurt to become familiar with the way the wizard works.)

When you choose the "Application" 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.



Click the action name to access action-specific help

You can click on this blue text to get more information about the action from the online help. (Go ahead and click on the blue “Application.Exit” to see the online help topic for this action. When you’re done, just close the online help window and return to the AutoPlay design environment.)

5) Click Next to advance to step 3 in the action wizard.

Some actions have one or more settings that you can configure in step 3 of the action wizard. When that is the case, you can click the Next button to advance to step 3, where you can customize the action.

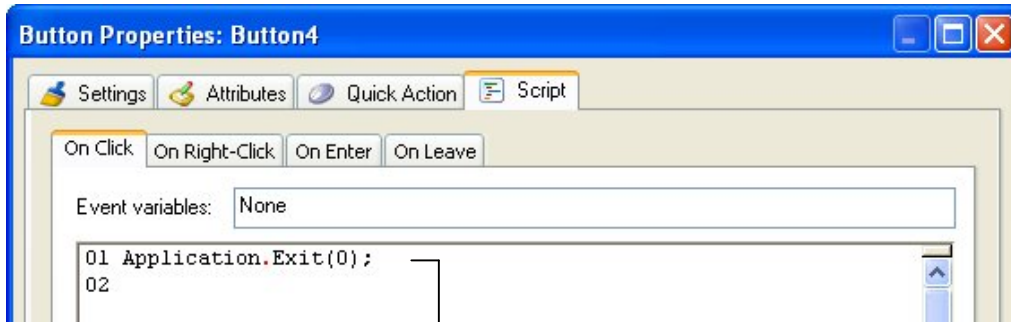
In this case, the Application.Exit action allows you to specify a return code that will be returned by your AutoPlay application after it exits. (This allows you to pass information back to the program or batch file that launched the application, for example if an error occurred you could return an error code so the batch file could proceed accordingly.)



For our purposes, the default return code of 0 is fine. (This is the standard return code that programs use to indicate that everything’s okay.)

6) Click Finish to add the Application.Exit action to the On Click event, and then click OK to close the script editor.

Clicking Finish on the New Action wizard will close the wizard and add the action to the script editor.



The action that was added

When you click OK on the script editor, the script editor will close, and the On Click setting in the properties pane will have “1 Line” next to it. The properties pane always shows how many lines of script there are in each of the selected object’s events.

| Actions | |
|----------------|------------|
| Quick Action | -- None -- |
| On Click | 1 Line |
| On Right-Click | -- None -- |
| On Enter | -- None -- |
| On Leave | -- None -- |

7) Choose Publish > Preview. When the application opens, click on the “Exit” button to close it.

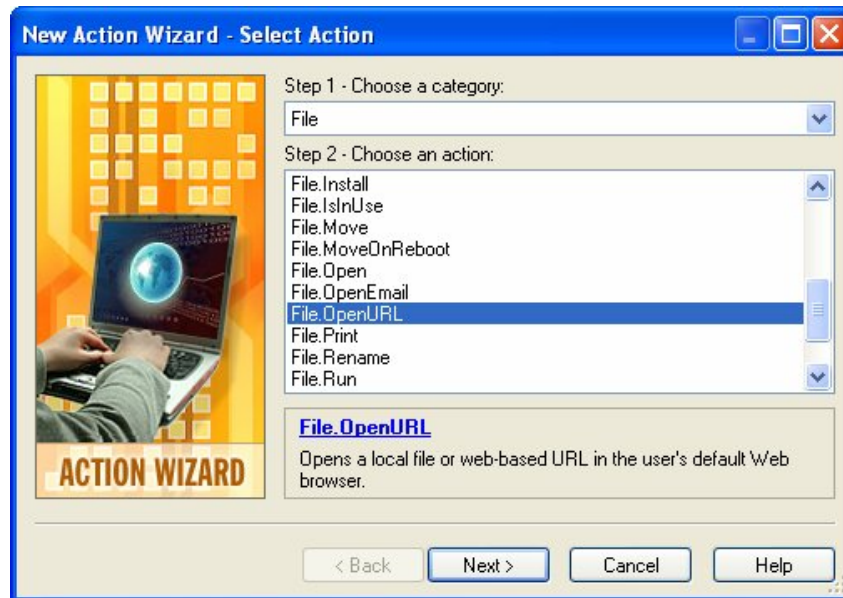
It works! Clicking on the “Exit” button triggers that object’s On Click event, which causes the On Click script to be performed. Since you put an Application.Exit action in the script for that event, the application closes.

Note: The Application.Exit action causes an immediate exit from the application.

8) Select the “Ted Sellers Online” button and add a File.OpenURL action to its On Click event. Set the action’s URL parameter to “<http://www.autoplaystudio.com>” and leave the WindowMode parameter set to SW_SHOWNORMAL.

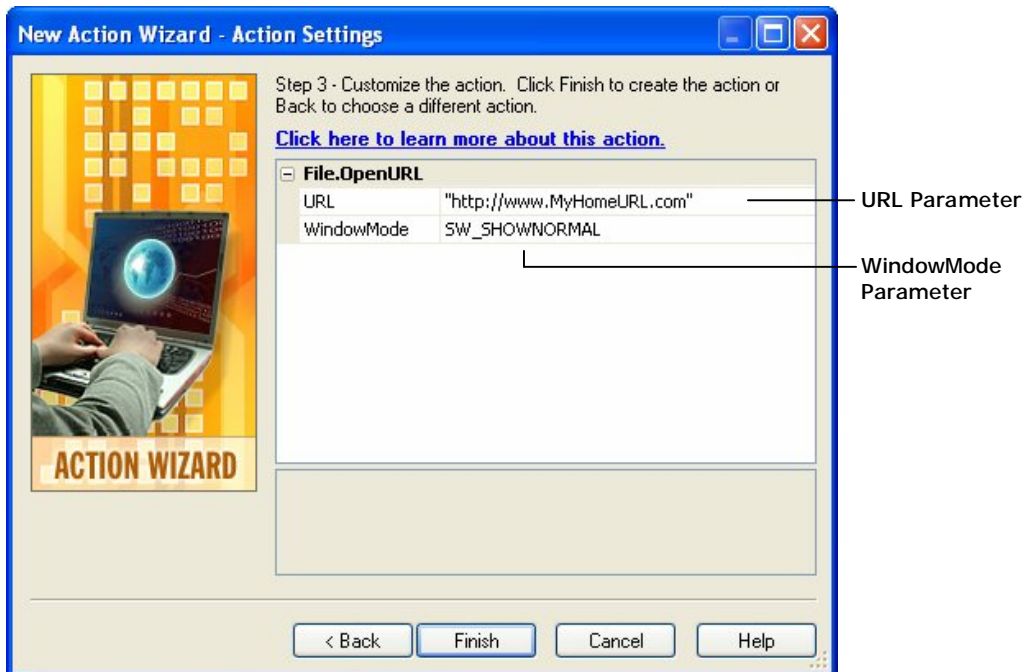
Adding a File.OpenURL action to the “Ted Sellers Online” button object (the one named Button1) is a lot like adding an Application.Exit action to the “Exit” button object. Just select the button object, click on the On Click setting, and click the edit button to bring up the script editor.

When the script editor opens, click the Add Action button to open the New Action wizard. Choose the “File” category from the drop-down list to see the list of File actions, and then click on File.OpenURL.



Once you have the File.OpenURL action selected, click Next to move on to step 3 of the New Action wizard. This is where you can configure the action’s *parameters*.

Parameters are just values that get “passed” to an action. They tell the action the information it needs to know in order to do what we want it to do. For instance, in the case of our File.OpenURL action, the action needs to know *what* URL we want it to open. So, the first parameter lets you specify the URL of a website.



Since Ted doesn't actually have a website, we'll use the AutoPlay Media Studio website for this parameter, just for testing purposes. To do that, simply double-click on the URL setting and type `"http://www.indigoroze.com/ams"` into the text field. (Feel free to substitute another website's URL if you want.) When the action is performed, it will open the user's default web browser and navigate to this URL.

Note: Make sure you include the quotes! Text strings need to be quoted in action parameters.

The other parameter, WindowMode, lets you specify how the user's default web browser should open up—either normally, minimized, or maximized. It's set to SW_SHOWNORMAL by default, which is fine for our purposes.

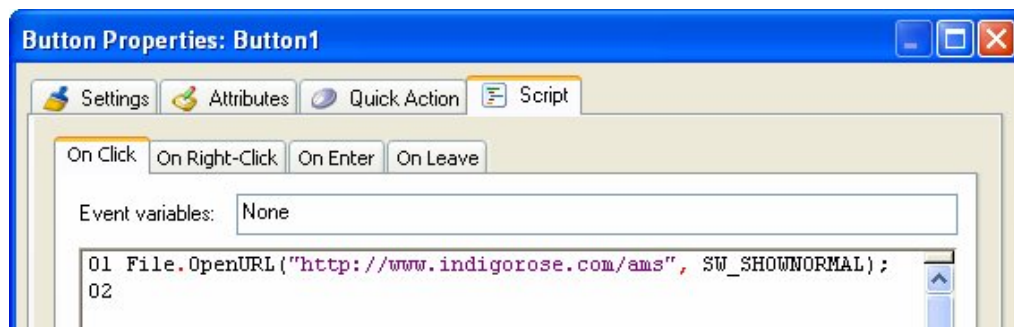
| File.OpenURL | |
|--------------|---------------------------------|
| URL | "http://www.indigoroze.com/ams" |
| WindowMode | SW_SHOWNORMAL |

Constants

SW_SHOWNORMAL is a *constant*. A constant is just a name that represents a value, essentially becoming an “alias” for that value. Constants are often used to represent numeric values in parameters. It’s easier to remember what effect SW_MAXIMIZE has than it is to remember what happens when you pass the number 3 to the action.

Tip: The File.OpenURL action can also be used to open local HTML files. Just click on the URL parameter, click the browse button, and use the Select File dialog to select a file. The file will be copied into your project’s Docs folder and accessed from there. (Note that if the local html file has any links to other local html files, you will need to copy those files into the Docs folder on your own, or the links won’t work.)

Once you’ve set the action’s parameters, click Finish to close the New Action wizard. The File.OpenURL action will appear in the list on the script editor. Note that the parameters you provided are listed between parentheses after the action’s name, in the same order they appeared in on the New Action wizard, separated by a comma.



Finally, click OK to close the script editor.

9) Choose Publish > Preview. When the application opens, click on the “Ted Sellers Online” button to open the URL you specified in step 8. When you’re done browsing that website, close the web browser, and then click on the “Exit” button object to close the application.

The File.OpenURL action opens whatever URL that you specify in the program that is registered as the default web browser on the user’s system. In most cases, this will be some version of Internet Explorer or Mozilla Firefox, but it could also be Netscape Communicator, or even Opera. The important thing is that it will open the website in the web browser that the user is familiar with.

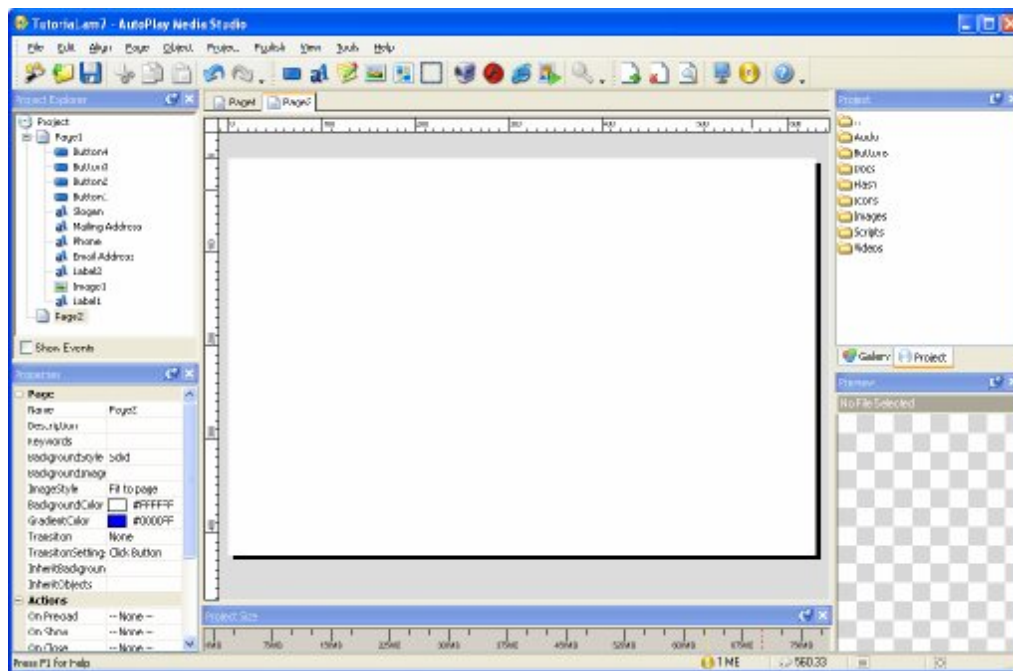
Note that the preview application remains running in the background while you browse the website. The File.OpenURL action is an excellent way to let someone visit a web page without losing their “place” in your application.

Adding Pages

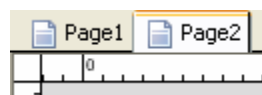
The other two buttons are going to jump to other pages in the application, so let’s start off by adding those pages.

1) Choose Page > Add to add a new blank page.

When you choose Page > Add, a new page is added to the project.



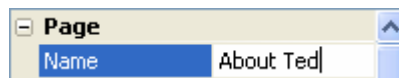
You can see that a new page has been added by looking at the top of the work area, where a second page tab has appeared.



When you add a new page, it is automatically selected, so the new page's name also appears in the properties pane. This new page's name is Page 2. (The first page was named Page1, and the new page gets the next available number.)

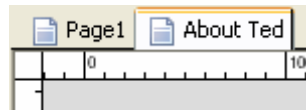
2) In the properties pane, change this page's name to *About Ted*.

To change the page name, just highlight the text in the Name setting and type in the new text.



(Remember to press Enter when you're done to confirm the change to the Name setting.)

When you edit the Name setting, the name of the page also changes on the page tab.



Tip: You can click on the page tabs to switch between pages.

3) On the Project pane, navigate to the Images folder, and locate the file named 630B1151.jpg. Drag this image file onto the page surface. When asked if you want to set this image as your page background, click Yes.

If you remember from Lesson 2, whenever you add an image to your project, a copy of the image file is kept in the project's Images folder. You can access these "cached" images by using the Project pane.

Since the Project pane remembers where it was the last time you used it, it should still be in the Buttons folder. (We navigated to the Buttons folder earlier in this lesson.) You'll need to navigate out of that folder, and then navigate into the Images folder.

Remember that you can navigate out of a folder by double-clicking on the special up folder (the one with two dots next to it). This moves you up one level in the folder hierarchy.

Once you're in the Images folder, all of the image files that you've used in the project will be listed on the Project pane. This includes the file named 630B1151.jpg which you used for the background image on Page1. To use this image as the background

image for the new About Ted page, just drag it onto the page surface. When asked if you want to set this image as your page background, click Yes.

Tip: If you don't want to be asked this question in the future, click on the "Don't ask me again" checkbox. If you do that, the same answer will automatically be used in the future—either Yes or No, depending on which button you click.

4) Choose Page > Duplicate to add another page just like this one.

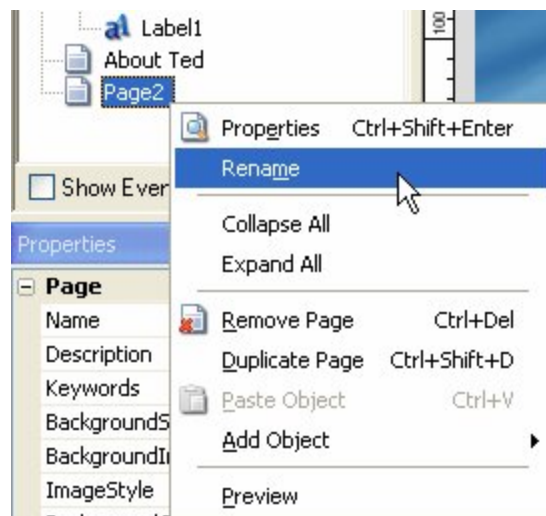
Like duplicating objects, you can duplicate the current page. When you duplicate a page, all of its settings and objects are duplicated as well. The only thing that is different on the new page is its name.

Note: If you're wondering why the new page is called Page2, and not Page3, it's because AutoPlay always gives new pages (and objects) the smallest available number when it names them. So, when you renamed Page2 to "About Ted," the name "Page2" became available again.

Tip: You can also duplicate a page by right-clicking on the page tab and choosing Duplicate.

5) In the project explorer, right-click on the new page (Page2), and choose Rename. Change the page's name to *Video*.

In the default workspace layout, the project explorer is on the left of the design environment.



Right-clicking on the page's name in the project explorer and choosing Rename from the context menu allows you to rename the page directly in the project explorer, just like renaming a file in Windows.



Simply type in the new name, and press Enter.

Tip: You can also change a page's name by double-clicking on the page surface and editing the Name setting on the Page Properties dialog.

Adding Navigation Buttons

Now that we have three pages in the project, we need a way for the user to go from one page to another—in other words, to *navigate* the application. This is incredibly easy to do: you just add a Page.Jump action to whatever event you want to use as the “trigger” for the change.

For this project, we'll use button objects to trigger the page changes—specifically, the On Click events of two buttons on Page1, and of one button each on the About Ted and Video pages.

Note: You don't need to use buttons for this...you could use label objects instead, or paragraph objects, or any combination of objects that have On Click events. For this example, though, we'll use buttons because they're cool.

Here's how it's going to work: on Page1, we're going to make the “About Ted Sellers” button (Button2) jump to the page named “About Ted,” and the “Video Presentation” button (Button3) jump to the page named “Video.”

On the About Ted page, we'll add a button that will jump back to Page1, and we'll add one on the Video page, too.

Let's start with the two buttons on Page1.

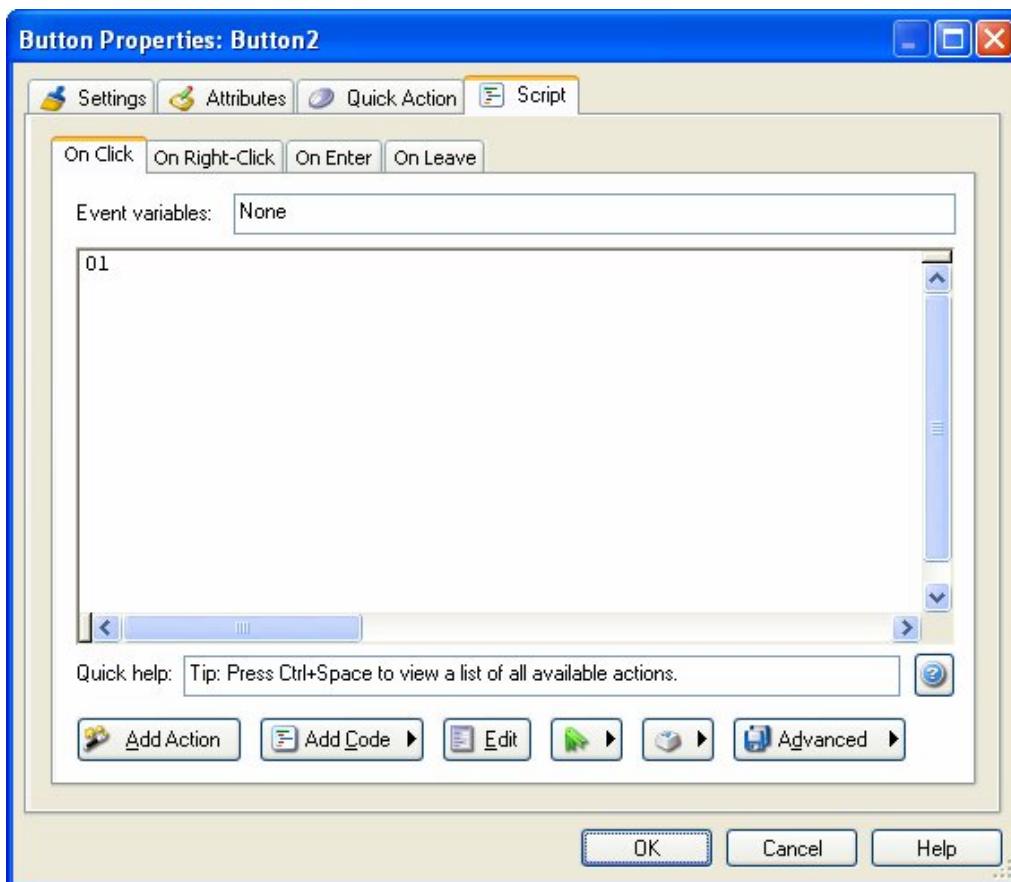
1) In the project explorer, click on the icon for Page1.

When you click on the Page1 icon, Page1 reappears in the work area. Clicking on a page in the project explorer switches to that page, just like clicking on the page tab.

2) Double-click on the “About Ted Sellers” button (Button2). Click on the Script tab to switch to the script editor, and click on the On Click tab.

Double-clicking on a button object opens the Button Properties dialog.

Each Button Properties dialog has four tabs: a Settings tab, for object-specific settings; an Attributes tab, for settings that most objects have in common; a Quick Action tab, where you can specify a single ‘simple’ action that will be performed when the object is clicked; and a Script tab.



The Script tab is, in fact, where the script editor is located.

Note: “The script editor” is just another way of saying “the Script tab on the Properties dialog.” One refers to *what* it is, and the other refers to *where* it is.

Since button objects can respond to five different events, there are five tabs on the script editor (or, depending on how you want to look at it, five tabs on the Script tab). Clicking on the On Click tab displays the action script for the object's On Click event, which at the moment is empty.

Note: This is the exact same place you are taken to when you click on the On Click event in the properties pane and then click the edit button.

3) Click the Add Action button, choose the Page category, select the Page.Jump action, and click Next. Set the PageName parameter to "About Ted", and click OK.

The Page.Jump action will cause the application to close the current page and open another one. You just need to tell the action which page you want it to open, which you do by setting the PageName parameter.



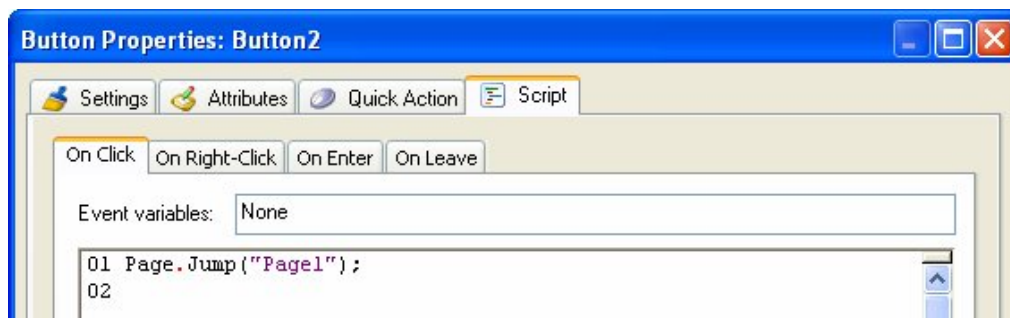
An easy way to change the PageName parameter to "About Ted" is to click on the PageName field, and then click the select button to bring up a list of all the page names in the project. (Note that since page names are text strings, and string

parameters must be quoted, all of the page names in the drop-down list are quoted for you.) Choose "About Ted" from the list, and you're done.



You can also just type "About Ted" into the PageName field if you want.

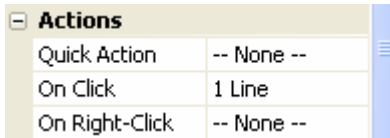
Once you click Finish, the wizard will close and the action will appear on the script editor.



4) Click OK to close the script editor.

Clicking OK closes the script editor and confirms the changes that you've made in it.

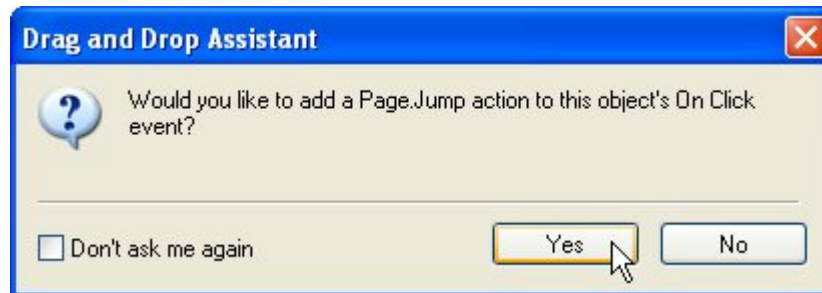
In the properties pane, "1 Line" will appear next to the On Click event, to indicate the number of lines that are currently in that event's action script.



| Actions | |
|----------------|------------|
| Quick Action | -- None -- |
| On Click | 1 Line |
| On Right-Click | -- None -- |

5) Drag the "Video" page icon from the project explorer onto the "Video Presentation" button object (Button3). When asked if you want to add an action to that object's On Click event, click Yes.

This time, we've taken advantage of the drag and drop assistant to add the page jump for us. When you drag a page icon from the page manager and drop it on an object that has an On Click event (like a button or a label), AutoPlay asks if you want to add an action to that event to jump to the page you dragged.



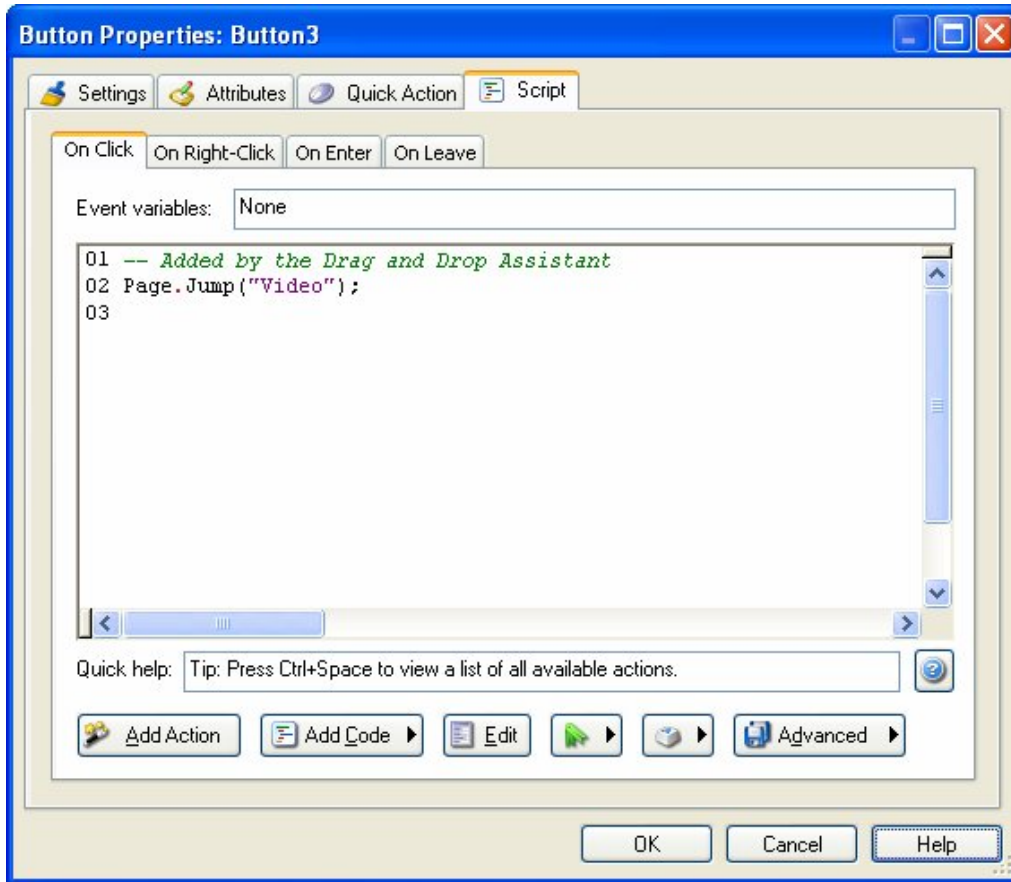
When you click yes, AutoPlay adds the following Page.Jump action to the object's On Click event:

```
-- Added by the Drag and Drop Assistant  
Page.Jump( "Video" );
```

6) Double-click on the "Video Presentation" button object (Button3). When the script editor opens, click on the On Click tab, and confirm that the Page.Jump action was added.

Double-clicking on the button object will open the Button Properties dialog to the same tab that you used last time you had the dialog open. In this case, it was the Script tab, home of your friendly neighborhood script editor.

On the On Click event's tab, you can see the action that the drag and drop assistant added for you.



7) Click OK to close the script editor.

That takes care of the two navigation buttons on Page1. Now we just need a way to get back to Page1 from the other two pages. Instead of adding a new button to each page, and having to set up their highlight and click colors from scratch, let's copy one of the buttons from Page1 for a bit of a head start.

Copying Objects

You'll often find yourself wanting to use the same object on more than one page, or set up a new object on one page that is just like an object on another page. In order to copy an object from one page to another, you need to copy it into the clipboard, and then paste it back into the project.

Note: You can make copies of objects on the same page by duplicating them, but if you want to duplicate an object to another page, you need to copy and paste it.

Tip: You can also *move* an object from one page to another by *cutting* it instead of copying it. To cut an object, just press Ctrl-X, or choose Edit > Cut. Cutting an object is just like copying one, except that the original object doesn't remain behind—it's removed.

1) Make sure the “Video Presentation” button object is still selected, and press Ctrl+C to copy the object into the clipboard.

When you press Ctrl+C, the currently selected item is copied into the clipboard. (The clipboard is just the place in your computer's memory where Windows puts stuff while you copy and paste them.) You can copy and paste objects, just like copying and pasting text in a word processor.

Tip: You can also copy the currently selected object by choosing Edit > Copy.

2) Switch to the About Ted page. Click on the page surface, and press Ctrl+V to paste the object onto the page.

To switch to the About Ted page, just click on its page tab at the top of the work area.

When you press Ctrl+V, a new (copied) button object appears on the page, in the same position as the original. All of the settings are carried over from the original. In this case, since there are no other button objects on this page, even the object's name is copied. (Just like the original, the new button object's name is also Button3.)

Remember the first rule of object naming? No two objects can have the same name, *on the same page*. When you copy and paste an object, AutoPlay first checks to see if that name is already taken by another object on the page; if it isn't, then it lets the object keep the same name as the original. If the name is already in use on the page, then the object is given a new name, just like it would if you added it from scratch.

Tip: You can also paste an object from the clipboard by choosing Edit > Paste.

3) In the properties pane, click on the ButtonFile setting, then click the browse button. When the Select File dialog appears, click the Gallery button, and select the grey_pill.btn file. Click OK when you're done.

This is a good trick to remember when you're working with button objects, especially if you've customized the text colors like we have: you can change the appearance of a button object without losing any of its settings by selecting a different button file in the properties pane.

This makes it easy to create buttons with consistent highlight and click colors. Just copy an object with the settings already configured, and paste it in as a new object...then change the object's file, or text, or anything else. You're basically using an existing button as the template for a new one.

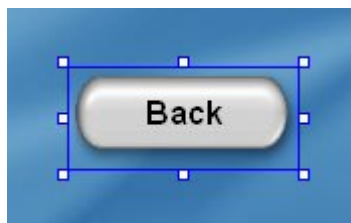
Note: You can do the same thing by duplicating an object, and then changing the new object's settings...but only copying and pasting lets you copy an object to another page.

The grey_pill.btn file is similar to the grey_rounded.btn file that the original object used, but it's more pill shaped, with round ends, instead of only rounded corners.



4) In the properties pane, change the button object's Text to *Back*. Set the Width to 114, the Height to 50, and the Left setting to 55.

This button is going to be used to go back to Page1, the main "table of contents" page of our application. We want to leave enough room for the other stuff we're going to put on this page, so we've made the button small and put it on the left side.



5) Use the alignment tools to center the object vertically on the page.

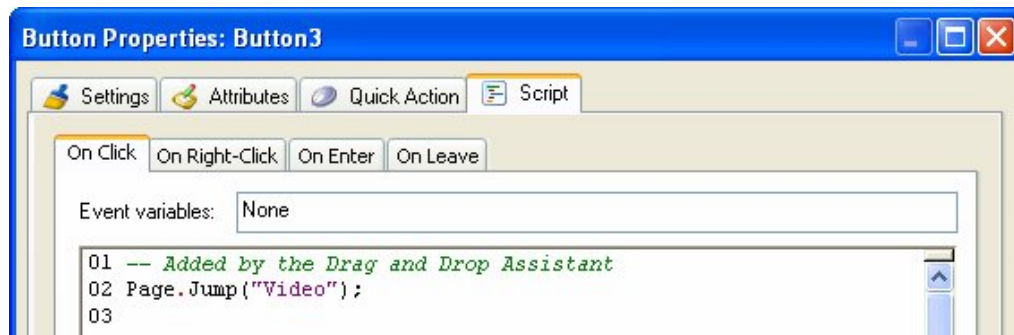
To center the object on the page, first make sure you're in Align to Page mode. Then, right-click on the object and select Align > Center Vertical.

6) In the object's On Click event, change the Page.Jump action's PageName parameter from "Video" to "Page1".

This object already has a Page.Jump action on its On Click event, but it's currently configured to jump to the Video page. (The action was copied along with the object's other settings.) Since our "Back" button is already *on* the Video page, we need to modify the action so it will jump somewhere else. In this case, we want the action to jump back to Page1.

Open the script editor and access the object's On Click event. (There are two ways to do this: you can either double-click on the object, click on the Script tab, and click on the On Click tab...or, you can click on the On Click setting in the properties pane, and then click the edit button.)

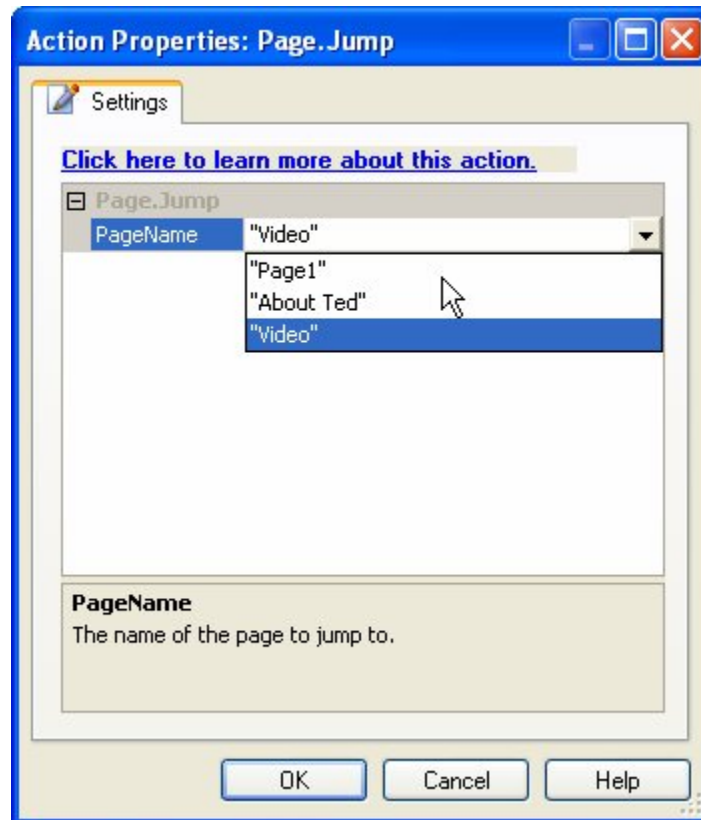
You should see the existing Page.Jump action.



You could remove the existing action and then add a new one, but since you'd end up adding another Page.Jump action anyway, you might as well just edit the existing action. (Only the parameter needs to change.)

To edit the action, just double-click on it. Double-clicking on the action opens the Action Properties dialog, where you can modify the action's current parameters. Click on the PageName parameter, click the select button, and choose "Page1" from the drop-down list.

Tip: You can also just edit the action's text directly.



Changing the PageName parameter from "Video" to "Page1"

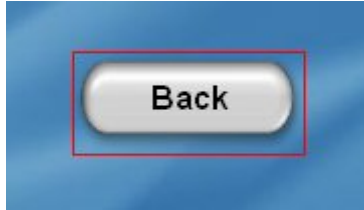
Finally, click OK to finish editing the action, and then click OK to close the script editor and confirm the change.

Tip: There's another action you could use to jump back to Page1: a Page.Navigate action with the JumpType parameter set to PAGE_FIRST would jump to the first page listed in the project explorer...which in this case happens to be Page1.

7) Right-click on the object and choose Pin.

Pinning the object prevents it from being moved or resized, so you won't have to worry about moving it by accident. (We already have it right where we want it.)

When you pin the object, its bounding box changes from blue to red (so you can tell that it's pinned), and the resize handles disappear.



Of course, if you ever change your mind, you can always just unpin the object and move it somewhere else.

Tip: You can also pin an object by selecting it and pressing Ctrl+P.

8) Copy the button object from this page to the Video page.

To copy the object, just select it (it should already be selected), press Ctrl+C, switch to the Video page (by clicking on its page tab), and press Ctrl+V.

The new object will end up in the same position as the original, with all of the same settings—including the Page.Jump action. Since the original object was pinned, the new object is pinned as well.

Note: You can copy and paste objects when they're pinned.

That's it for this page. You don't even have to adjust the object's Page.Jump action, since it's already configured to go to Page1, which is what we want the "Back" button on this page to do as well.

Trying It Out

Now that we have all this cool navigation built into the project, let's see how it works.

Of course, before we test anything, we should save the project just in case anything goes wrong.

1) Save the project.

Saving the project should be second nature to you by now, so go ahead and choose File> Save, or press Ctrl+S, or click the Save button on the Standard toolbar.



It doesn't matter which method you use. Heck, if you're really cautious, use all three,

and then write “Saved in triplicate” in your work log. (Or scribble it on your napkin at lunch. Then fold the napkin and giggle...but not too loudly, or someone will think you’re odd. Believe me, I know.)

2) Choose Publish > Preview and try out the buttons.

Now it’s time to see these actions in action!

Tip: If you’re feeling impatient, choosing Publish > Preview may be too slow. You need results now! In that case, just press the F5 key on your keyboard.

When the application appears, try clicking on the “About Ted Sellers” button. The application “jumps” to the About Ted page, with the lonely little Back button on the left. Click on the Back button, and you’re instantly taken back to Page1.

The same thing happens if you click on the “Video Presentation” button. Note that although the Video page looks exactly like the About Ted page, you’re actually going to two different pages. (In lessons 6 and 7, we’ll add some objects to those pages so they don’t look the same any more.)

When you’re done navigating the application, return to Page1 and click on the “Exit” button. (Or, just press Alt+F4 to exit the application.)

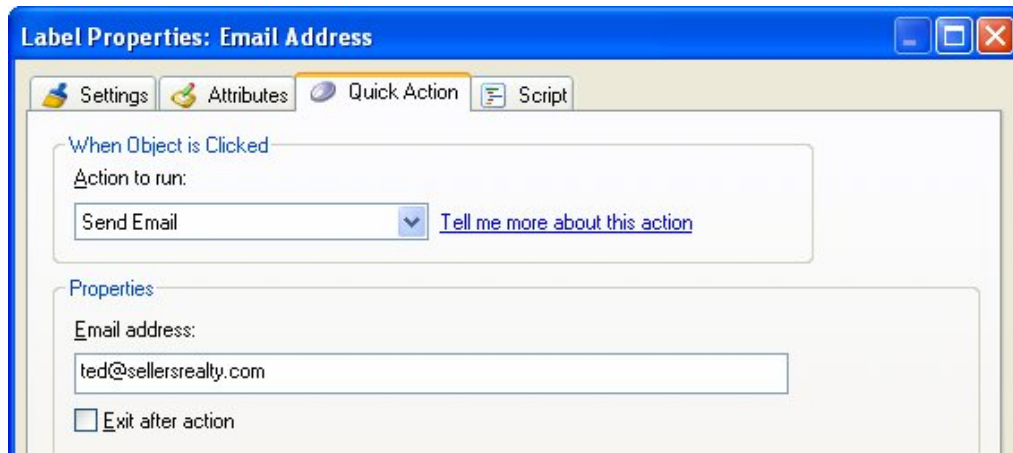
Sending Email

Before we move on to the next lesson, there’s one more action we need to add: our Email Address label object needs an action to make it actually send an email.

1) Switch to Page1, and double-click on the Email Address label object. Add a quick action to send an email to “*ted@sellersrealty.com*”.

A quick action is a simpler alternative to placing script on an object’s On Click event. To add a quick action, double-click the object you want to add it to (in this case, the Email Address label object). When the Properties dialog appears, click on the Quick Action tab, and select ‘Send Email’ from the “Action to run” drop-down list.

Notice that when you select Send Email, the properties specific to that quick action are displayed below. Change the “Email address” property to *ted@sellersrealty.com*. Be sure to leave the “Exit after action” option unchecked.



Tip: You can automatically fill in the email's subject field, too. Just put “?subject=” (without quotes) after the email address, followed by the text that you want in the subject line. For example:

```
"ted@sellersrealty.com?subject=Please send me real estate market info"
```

...would send an email to ted@sellersrealty.com with “Please send me real estate market info” as the subject line.

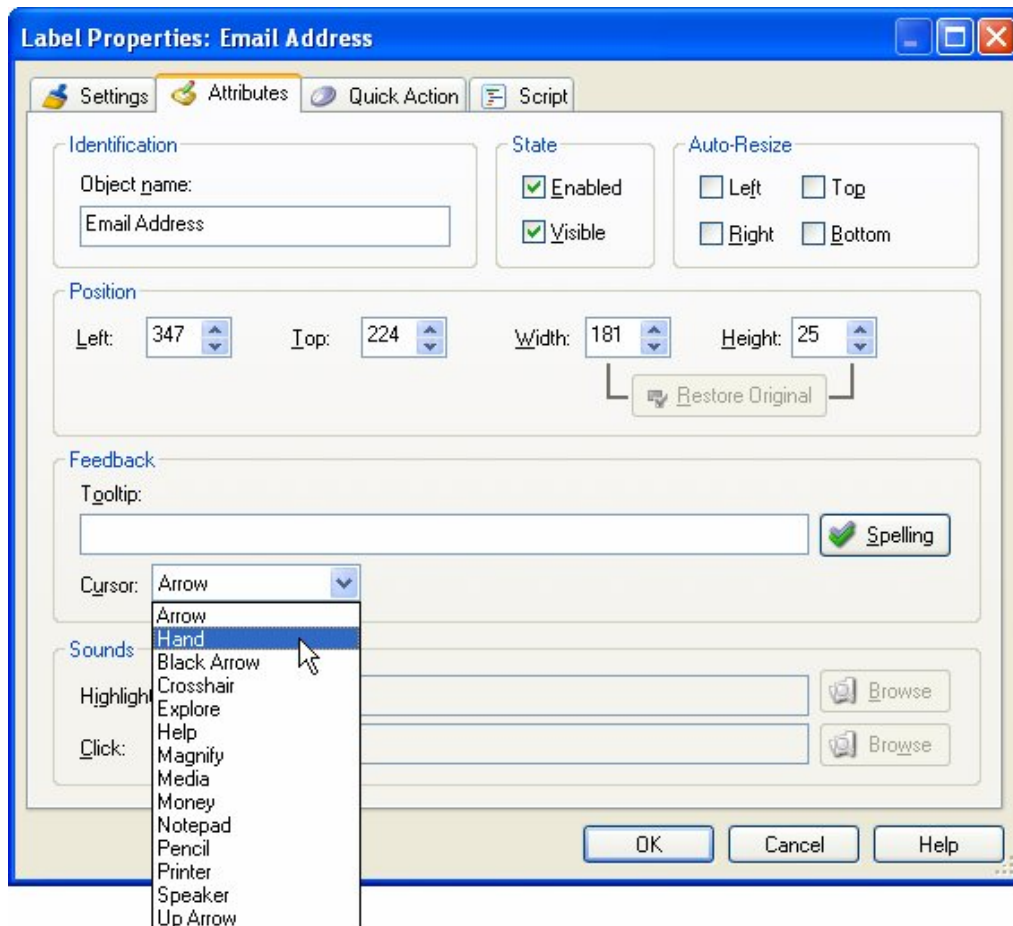
If you're a web monkey like me, you're probably thinking that this is just like a mailto: link in HTML. If so, you're right...and yes, that means you can specify the default body text, and everything else that you can do in a regular mailto: link, too.

2) Click on the Attributes tab. In the Feedback category, change the Cursor setting from “Arrow” to “Hand.”

You can use the Cursor setting to change the type of cursor that will be used when the mouse moves over an object in your application.

By default, label objects have their Cursor setting set to “Arrow,” which is the same type of cursor that appears when the mouse is over the page surface. Changing it to “Hand” helps show that the object is clickable. (The “Hand” cursor is the one that is used for button objects. It's also the cursor that appears when you hover over hyperlinks in a web browser.)

Note: “Cursor” is just another name for the mouse pointer.



Once you've changed the Cursor setting, click Ok to close the Properties dialog.

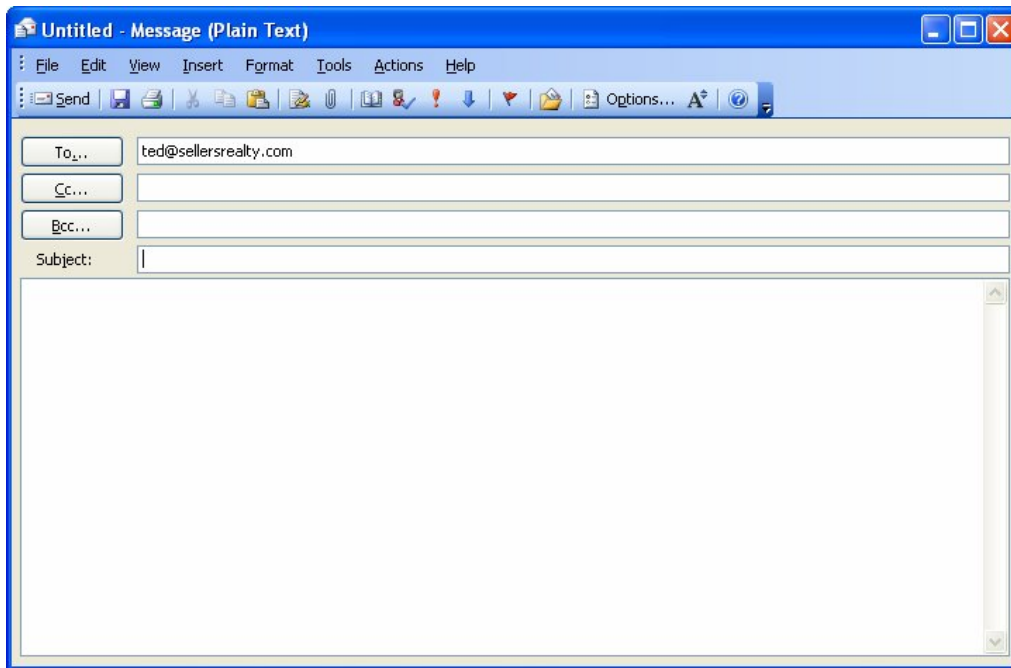
3) Save the project, and then press F5. When the preview application appears, move the mouse over the Email Address label object.

See how the mouse pointer changes from the arrow to the hand? That's the Cursor setting at work.

Note: You should always save your work after you've made a change that you want to keep.

4) Click on the Email Address label object.

If your system is properly configured, this will start a new email message using your default email program.



Notice how the email address that we supplied is already entered into the “To:” field. To send this message, you would just need to fill in the subject, type in your message, and click Send.

Since ted@sellersrealty.com isn’t a real email address, you don’t want to actually send *this* message, so just close the email (cancel it).

Finally, click on the “Exit” button to exit from the preview application, and return to the AutoPlay design environment.

Lesson 4 Summary

In this lesson, you learned how to:

- Add interactive buttons
- Match the width and height of several objects at once
- Change the appearance of the button text
- Duplicate multiple objects
- Add simple actions like `Application.Exit` and `File.Open`
- Add a blank page to the project
- Duplicate an existing page
- Add navigation buttons using `Page.Jump` actions
- Copy objects from one page to another
- Send email with a Quick Action



Lesson 5:

Status Text

Status text is text that appears when you move the mouse over an object, and then disappears when you move the mouse off of the object. It's a nice way to add a bit more description or instruction to interactive objects.

5

What You'll Learn

In this lesson, you'll learn how to:

- Add a paragraph object
- Turn off the paragraph object's vertical scroll bar
- Add actions to make the text dynamic, so it changes in response to events
- Add actions to the page's On Show and On Preload events
- Move an action from one event to another

How Long Will It Take?

This lesson takes approximately 20 minutes to do.

Starting the Lesson

If you're continuing from Lesson 4, you should still have AutoPlay running with the Tutorial project open. If so, you're ready to move on to the next exercise: Adding a Paragraph Object.

Otherwise, you'll need to open the project file that you saved at the end of Lesson 4.

1) Open the Tutorial.am7 file that you saved in Lesson 4.

When you save a project, AutoPlay automatically creates a project folder for it inside your "My Documents\AutoPlay Media Studio 7.0\Projects" folder. This project folder is where everything that belongs to the project is stored—including the project file, which contains all of the settings used in the project.

The project folder and the project file always have the same name that you gave to the project when it was created. The project file's name ends with a ".am7" file extension.

Since you named this project "Tutorial" in Lesson 1, the name of the project folder will be Tutorial, and the name of the project file will be Tutorial.am7. So, the path to the project file should be something like:

```
...\My Documents\AutoPlay Media Studio 7.0\Projects\Tutorial\Tutorial.am7
```

To open the project, you just need to open that project file.

Adding a Paragraph Object

Paragraph objects are similar to label objects, but they have a few important differences:

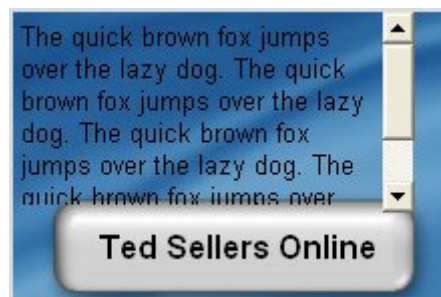
- The font size doesn't change when you resize the bounding box...instead, the bounding box determines the area that the text can be displayed in.
- The object's dimensions stay the same even if you change the text with an action at run time.
- You can set a background color for the object.
- You can put a border around the object.
- You can display scrolling text in it, complete with scroll bars.

Paragraph objects are ideally suited for displaying status text. They allow you to specify an area that the text will appear in, and guarantee that the text will never extend beyond it. And, because the bounding box remains constant, you can center the text horizontally within the area, and the center point won't shift when the text changes.

Tip: As you'll see in Lesson 6, the paragraph object's optional scroll bars make it perfect for displaying lots of text, too.

1) Choose Object > Paragraph and click OK.

The new paragraph object appears in the upper left corner of the page. Note the default text ("The quick brown fox jumps over the lazy dog," repeated a few times), which serves as a reminder that paragraph objects are made for presenting long passages of text.



Tip: You can also add a paragraph object by right-clicking on the page surface and choosing Paragraph from the right-click menu.

2) In the properties pane, change the object's name to *Status Text*.

This object is going to be the target of some Paragraph.SetText actions, so we might as well give the object a more meaningful name. Since it's going to be displaying status text, we might as well call it that. (If we were going to have more than one object with status text on the screen, we would need to be even more descriptive in the name, like "Status Text Top" or "Navigation Status" or something. But we're only going to need one status text display, so we can get away with calling it "Status Text.")

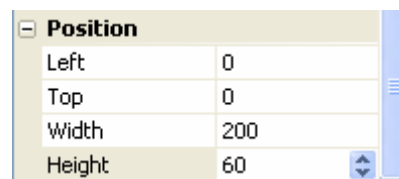
3) Drag one of the resize handles to make the object bigger.

When you make the object's bounding box bigger, the text reflows to take advantage of the new space. Note that the text wraps around when it reaches the edge of the bounding box, just like a paragraph. (That's why it's called a paragraph object.)

Unlike a label object, when you resize a paragraph object, the text size doesn't change. Only the size of the bounding box changes. The text just adapts to the new shape.

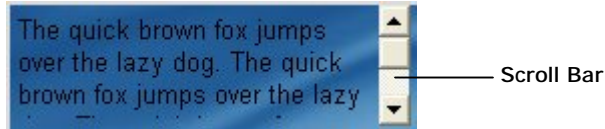
4) Resize the paragraph object so its width is 200 pixels and its height is 60 pixels (200 x 60).

The easiest way to resize the object to a specific size is to change the Width and Height settings in the Position category of the properties pane. In this case, just set the Width to 200, and the Height to 60.



You can also just resize the object with the mouse until you get the dimensions right, but I find doing it that way takes longer.

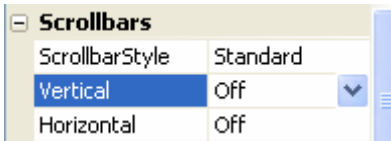
Notice that once again the font size doesn't change when you make the paragraph object smaller; only the bounding box gets smaller.



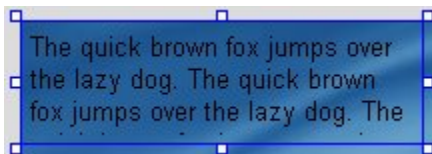
Since the bounding box is now too small for all of the text to fit, a scroll bar appears on the right side of the paragraph object. (It wasn't there before, because the object's vertical scroll bar is set to Auto. When it's set to Auto, the scroll bar only appears if there isn't enough room in the object for all of the text.)

5) In the Scrollbars category of the properties pane, change the Vertical setting from "Auto" to "Off."

The Vertical setting controls the vertical scroll bar.



When you turn off the vertical scroll bar, the text is essentially "chopped off" at the bottom edge of the paragraph object's bounding box. (The object is sized too small to fit all of the text, so it just displays as much of the text as it can.)

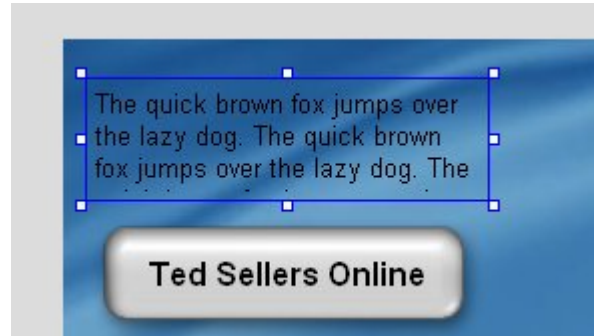


6) Use the alignment tools to center the object horizontally above the four buttons. In the Position category, set the object's Top to 20.

To center the paragraph object above the four button objects, first select it along with one of the button objects. (It doesn't matter which one.) Make sure Align to Page mode is off, then right-click on the button object and choose Align > Center Horizontal.

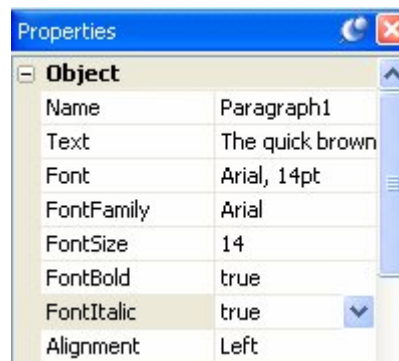
Note: Make sure the button object is dominant so the paragraph object will be aligned to *it*, and not the other way around.

Finally, select the paragraph object by itself, and set its Top setting (in the Position category) to 20.



7) Set the object's font to 14 point Arial, bold and italic.

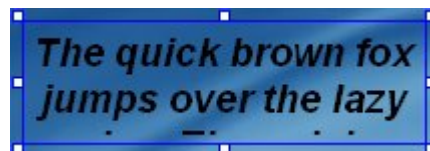
An easy way to do this is to change the FontSize to 14, and then double-click on FontBold and FontItalic to set them both to true.



Note that the bounding box stays the same size when you change the paragraph object's font size. (If this were a label object, the bounding box would have grown to accommodate the larger text.)

8) Change the object's alignment to "Center."

Now each line is centered horizontally inside the bounding box.

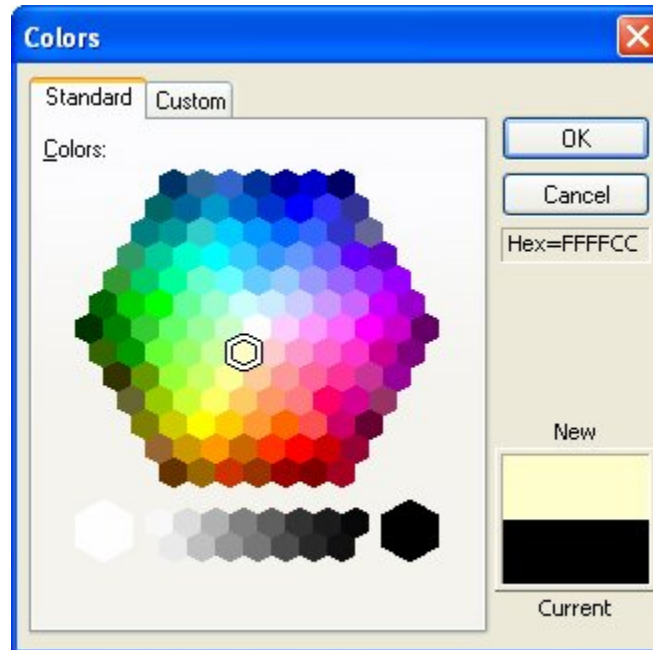


9) Set the Normal, Down and Highlight colors to #FFFFCC.

We don't want the text to change color on mouse-overs and clicks, so the Normal, Down and Highlight colors need to be the same.

There are two really quick ways to change the colors to #FFFFCC: you can double-click on the object, set the Normal color using the Standard tab of the Colors dialog, and then click Match Normal; or, you can type the hexadecimal into the properties pane, copy it into the clipboard, and paste it into the other two color settings.

For the first method, you double-click on the object, click on the Normal color chooser, click on More Colors, and select the color immediately down and to the left from the center. Click OK to close the Colors dialog, then click Match Normal, and click OK to close the Paragraph Properties dialog.



For the second method, you double-click on the Normal color setting in the properties pane to highlight the hexadecimal text. Then type in #ffffcc, double-click on the text that you just entered, and press Ctrl+C to copy it into the clipboard. Then double-click on the Highlight color value, and press Ctrl+V to paste the text that you copied. Do the same for the Click color, and then press Enter (or just click on another setting), and you're done.

Tip: Both of these methods are a lot easier to do than they are to describe. If they sound kind of complicated, they really aren't. Changing colors like this is something you'll do pretty often, so it pays to learn faster ways to do it. Try each method out and see which one you like best.

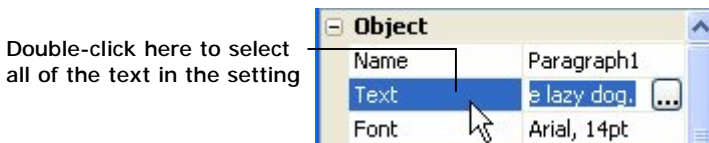
Making the Text Dynamic

Now for the fun part. We're going to make the text in our paragraph object change as the mouse moves over the interactive objects on the page. To do this, we'll give those objects On Enter and On Leave actions. (Which is another way of saying that we'll add some actions to their On Enter and On Leave events.)

1) Click on the paragraph object and double-click on the left column of the Text field in the Properties pane. Press the Delete key to erase the text, then press Enter to accept the change.

We want the paragraph object to start out empty when you first launch the application. The easiest way to do this is to just delete the text from the paragraph object, so that it is empty to begin with.

A quick way to highlight all of the text in a setting on the properties pane is to double-click on the name of the setting, in the left column of the properties pane. When you double-click on the name of a setting, all of the text in the setting is selected.



Once you have the text selected, pressing the Delete key deletes it. Pressing Enter makes the change permanent.

Tip: If you edit the text in a field by mistake, and you haven't pressed Enter yet (or deselected the setting by clicking somewhere else), you can cancel the change by pressing Esc.

Another quick way to delete the text is to double-click on the object and switch to the Settings tab. (Double-clicking on the object opens the Paragraph Properties dialog and automatically highlights all of the text in the Text field for you.) Then press Delete to erase the highlighted text, and click OK to confirm the change and close the dialog.

2) Select the “Ted Sellers Online” button object (Button1), and add a Paragraph.SetText action to its On Enter event. Set the action’s ObjectName parameter to “Status Text” and its Text parameter to “Visit our\nWeb site”.

The On Enter event will be triggered whenever the mouse moves onto the button object, “entering” its space.

Tip: If you click on the On Enter setting in the properties pane, and then click on the edit button, you are taken directly to the On Enter event in the script editor.

A quick way to add the Paragraph.SetText action is to click the Add Action button, and then type the letters *par* on the keyboard. As you type in these letters, the category drop-down will skip ahead to the first category that matches what you typed. Once the Paragraph category is shown, click on the Paragraph.SetText action in the list, and click Next to advance to the second page of the New Action wizard.



The `ObjectName` parameter lets you specify the name of the object that you want this action to operate on. In this case, we want to change the text in the paragraph object named “Status Text.”

The `Text` parameter lets you specify the text that will be displayed in the object. In this case, we want to display a message indicating what the “Ted Sellers Online” button is for.

The “\n” inside the message represents a newline character. It basically means to start a new line, just like pressing Enter in a text editor. (This is known as an *escape sequence*. An escape sequence is a special “code” that represents a character that can’t just be typed into a string normally. In this case, the code \n represents the invisible character you type by pressing the Enter key.)

Note: Remember to click Finish on the New Action wizard and then click OK on the script editor to finish adding the action.

3) Choose Publish > Preview. When the application opens, move the mouse over the “Ted Sellers Online” button a few times.

When you move the mouse onto the “Ted Sellers Online” button the first time, the text in the paragraph object changes. This is the `Paragraph.SetText` action at work.

Note that the text in the paragraph object doesn’t have any quotes around it. The quotes that you placed around the text in the action’s parameters aren’t included in the text when it’s displayed; they are only there to tell the action that you’re giving it a string.

Note: In computer lingo, a “string” is a sequence of characters, including text and anything else that can be typed. Referring to a sequence of letters and symbols as a “string” supposedly dates back to the late 1800s, when compositors would “string together” letters as part of the printing process. (In fact, it’s said that they were actually paid by the foot, and not by the word.)

When you move the mouse off the button, the text remains. This is because there’s nothing telling `AutoPlay` to change the text when the mouse moves off the object.

If we want the text to disappear when you move the mouse off the button, we need to add an action to the object’s `On Leave` event.

4) Exit the preview. Add a Paragraph.SetText action to the “Ted Sellers Online” button’s On Leave event. In the action’s parameters, set ObjectName to “Status Text” and Text to “”.

The On Leave event is triggered whenever the mouse moves off of the button object, “leaving” its space.

Setting the Text parameter to "" tells the Paragraph.SetText action to replace the text in the paragraph object with “nothing.” This is what programmers call an *empty string*. You can use an empty string like "" whenever you want to clear the text out of an object with an action.



5) Choose Publish > Preview, and try out the “Ted Sellers Online” button again.

Voila! Now the text disappears when you move the mouse off of the object.

6) Exit the preview, and add On Enter and On Leave actions to the other interactive objects.

Use the following text for the On Enter events: *"Learn more\nabout Ted"* for the About Ted Sellers button (Button2), *"Watch a video presentation"* for the Video Presentation button (Button3), *"Exit from this business card"* for the Exit button (Button4), and *"Email Ted"* for the ted@sellorsrealty.com label object (Email Address).

Use an empty string ("") for all of the On Leave events.

Tip: When you add an action to an On Enter event, you can switch right to the On Leave tab of the script editor to add an action to the object's On Leave event too. You don't have to actually exit the script editor and re-open it to assign an action to both events.

7) Preview the project, and try out the status text by moving the mouse over the interactive objects.

When you move your mouse over the interactive objects, the text in the paragraph object should appear and disappear.

8) Click on the "About Ted Sellers" button. When the About Ted page appears, click on the Back button to return to Page1.

This is interesting—when you return to Page1, the "Learn more about Ted" text is still showing in the paragraph object. But the mouse isn't even on the About Ted Sellers button any more.

To understand why this happens, you just need to walk through the series of events.

First things first: when you moved the mouse over the About Ted Sellers button, this triggered the object's On Enter event. That event has a Paragraph.SetText action, which changed the text in the paragraph object to "Learn more\nabout Ted."

Then, you clicked on the About Ted Sellers button, triggering the object's On Click event...which set off the Page.Jump action, sending you to the About Ted page.

On the About Ted page, you clicked on the Back button, which triggered that object's On Click event. That event's Page.Jump action sent you back to Page1.

When you returned to Page1, the paragraph object was still showing the "Learn more about Ted" text, because that's the last thing it was told to display. The paragraph object was never told to display the empty string, because the About Ted Sellers button's On Leave event was never triggered. The Page.Jump action took you to the

other page before the mouse had a chance to move off the button object and trigger its On Leave event.

Note: Objects always maintain their settings until they're changed by an action...even if you jump to another page and come back.

9) Exit the preview.

Before moving on to the next exercise, exit the application and return to the AutoPlay design environment.

Adding Page Actions

In order to clear out the paragraph object when you return from another page, we need to add an action to one of the page's events.

There are three page events that we could use to clear out the paragraph object: On Preload, On Show, or On Close (there are more events on the page, but these are really the only ones that apply).

- On Preload is triggered after the page has been "created" in memory, but before it is actually displayed on the screen.
- On Show is triggered right after the page appears on the screen.
- On Close is triggered when the page is closed.

Let's try using the On Show event first.

1) Click on the page surface. Add a Paragraph.SetText action to the On Show event. In the action's parameters, set ObjectName to "Status Text" and set Text to "".

We want the Paragraph.SetText action to clear out the paragraph object named "Status Text" every time this page is shown. Setting the paragraph object's text to "" will replace any existing text in it with an empty string, i.e. nothing.

2) Preview the project. Watch the paragraph object as you navigate to and from the About Ted page.

When you return to the first page, see how the text in the paragraph object is visible for a split second before it disappears? (If you have a really fast computer, the effect might not be that noticeable, so watch carefully.) This is because the On Show event isn't triggered until *after* the page is displayed. As a result, you may see the paragraph

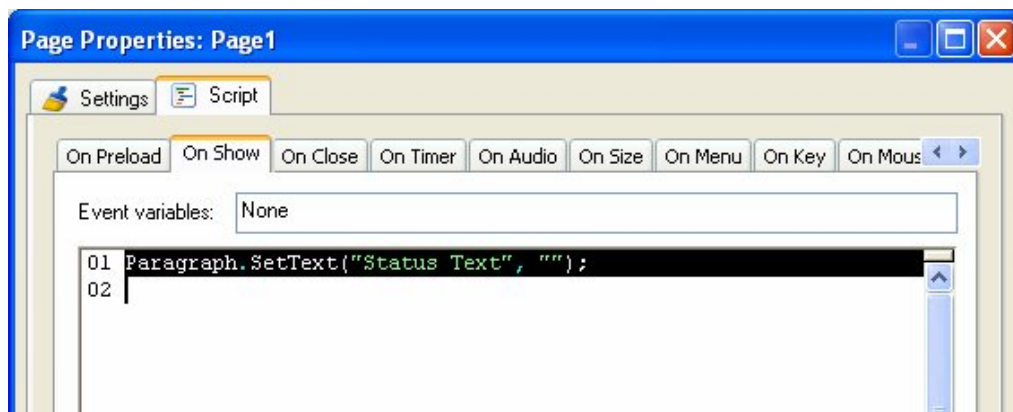
object with the old text in it before the Paragraph.SetText action has a chance to change it.

This works, but it doesn't look very professional. It would be better if we could clear out the paragraph object before the page was shown. An easy way to do that is to put the Paragraph.SetText action on the page's On Preload event instead. That way, the paragraph object is cleared before the page is displayed.

3) Exit the preview, click on the page surface, and move the action from the On Show event to the On Preload event.

The easiest way to move an action from one event to another is to cut and paste it.

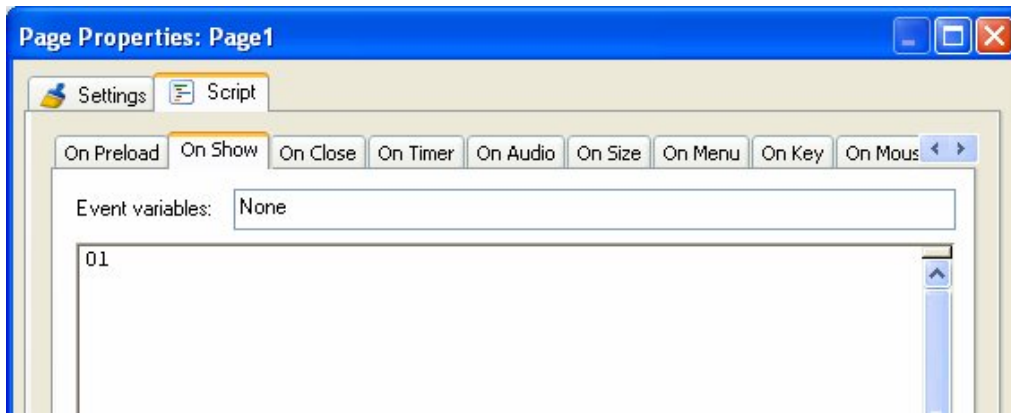
To cut and paste the Paragraph.SetText action, click on the On Show setting, and then click on the edit button. When the script editor appears, select the line of text that contains the action. This is just like selecting a line of text in a text editor, or in Word. You can use the mouse to highlight all of the text, or you can use the keyboard, or any combination of the two.



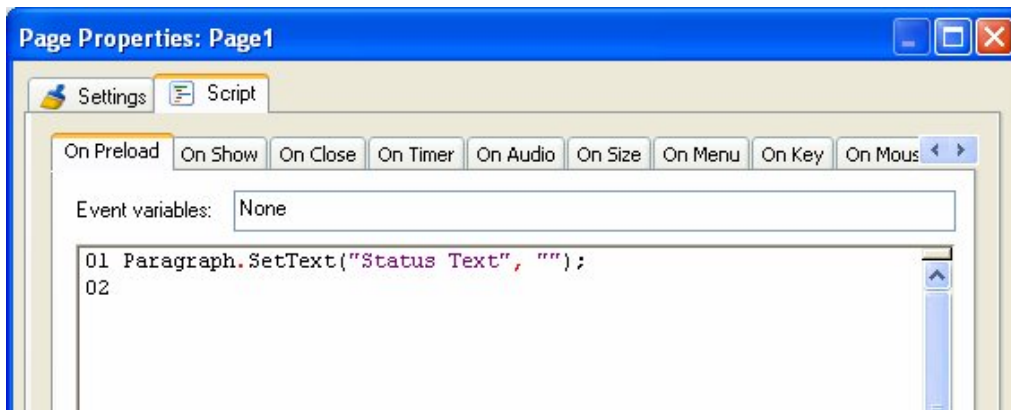
Tip: An easy way to select a line of text is to position the cursor at the start of the line, and then press Shift+Down. (Hold the shift key, and press the cursor down key.) This will select the entire line, including the newline character at the end.

Once you have the whole line selected, press Ctrl+X to cut the text. It will be removed from the script editor, and placed in the Windows clipboard.

Note: You can use this same technique to remove an action from the script editor. Since actions are just text, all you have to do is select the text and delete it, and the action is "gone."



To paste the action, just click on the On Preload tab, and then press Ctrl+V. The text will be copied from the Windows clipboard and inserted into the script editor.



Finally, click OK to close the script editor and accept the changes you have made to the events.

4) Preview the project. Navigate to and from the About Ted page.

Now when you go to another page and come back, the paragraph object is already cleared out when you get there. No more flickering!

5) Exit the preview. When you return to the design environment, right-click on the paragraph object and choose Lock.

We aren't going to have to make any more changes to the paragraph object, so you might as well lock it; that way you don't have to worry about selecting it by accident whenever you're working with the other objects on this page.

Locking an object is a good way to get it "out of the way." Once it's locked, you can left-click on it, and even drag-select around it, and AutoPlay will act like the object wasn't even there.

This is especially helpful since the object doesn't have any text in it. With no text inside it, the object is completely invisible unless you select it. (You can see an invisible object when it's selected, because the bounding box gives it away.) Being invisible makes it more likely that you'll click on the object by mistake. So lock it.

Tip: If you ever need to work on the object again, just right-click on it and choose Lock again to unlock it.

6) Save the project.

Now that you have the navigation buttons working the way you want them to, it's a really good time to save the project.

Note: Never forget this important step! You will avoid future heartache by saving the project whenever you make any changes that you want to keep. So, choose File > Save, press Ctrl+S, or click the Save button...whatever method you prefer.

That's it! Job well done.

Lesson 5 Summary

In this lesson, you learned how to:

- Add a paragraph object
- Turn off the paragraph object's vertical scroll bar
- Add actions to make the text dynamic, so it changes in response to events
- Add actions to the page's On Show and On Preload events
- Move an action from one event to another



Lesson 6:

Scrolling Text

Scrolling text is just text that can be scrolled, either vertically, horizontally, or in both directions. In AutoPlay, scrolling text can be displayed by using a paragraph object, which has built-in support for vertical and horizontal scroll bars.

Scrolling text is useful whenever you want to display more text than you can fit in a given space. For example, if you need to display a long license agreement, or the contents of a readme.txt file, or an article on cubicle horticulture penned by your favorite geek auteur.

6

What You'll Learn

In this lesson, you'll learn how to:

- Add a panel image to serve as a backdrop and border for some text
- Replace the text in a paragraph object by shift-dragging a text file
- Add a scrollable paragraph object
- Use a custom scroll bar style

How Long Will It Take?

This lesson takes approximately 15 minutes to do.

Starting the Lesson

If you're continuing from Lesson 5, you should still have AutoPlay running with the Tutorial project open. If so, you're ready to move on to the next exercise: Adding a Video Object.

Otherwise, you'll need to open the project file that you saved at the end of Lesson 5.

1) Open the Tutorial.am7 file that you saved in Lesson 5.

When you save a project, AutoPlay automatically creates a project folder for it inside your "My Documents\AutoPlay Media Studio 7.0\Projects" folder. This project folder is where everything that belongs to the project is stored—including the project file, which contains all of the settings used in the project.

The project folder and the project file always have the same name that you gave to the project when it was created. The project file's name ends with a ".am7" file extension.

Since you named this project "Tutorial" in Lesson 1, the name of the project folder will be Tutorial, and the name of the project file will be Tutorial.am7. So, the path to the project file should be something like:

```
...\My Documents\AutoPlay Media Studio 7.0\Projects\Tutorial\Tutorial.am7
```

To open the project, you just need to open that project file.

Adding a Panel Image

A panel image is just an image object that you put behind another object to achieve some kind of custom background or picture frame effect. Although you can use any image for this purpose, AutoPlay comes with a number of images that were designed to do just that.

Tip: You can order more panel images directly from our web site. There are literally thousands of such images available.

Panel images are especially helpful at providing a sort of “mini-background” behind a bunch of text, to make the text easier to read than it would be on the page background itself. (This comes in handy for those times when you’re using a really wild image for the page background.) They can also be used to provide a fancy border to “section off” an area of the page.

Note: The paragraph object has an optional border and background color, which can serve the same purpose as a panel image in many cases. For really cool, professional borders and backgrounds, though, a panel image is the way to go.

1) Switch to the About Ted page.

We’re going to add the panel image and the paragraph object to the About Ted page, so before you do anything else, click on the “About Ted” page tab at the top of the work area.



2) Switch to the Gallery pane and click the Images button. Double-click on the up folder, and then navigate into the Panels folder.

The Gallery pane is tabbed with the Project pane on the right side of the AutoPlay program window. You can switch between these tabbed panes by clicking on the appropriate tab.

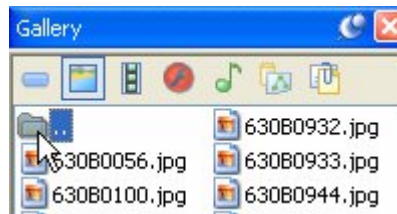
Note: Earlier we made the Gallery and Project panes visible. If they are not visible, you can make them visible by using the View > Panes submenu.

The Gallery pane lets you browse the library of files that came with AutoPlay. This library is organized into different sections, each one accessible with a button at the top of the pane. Clicking on the Images button at the top of the pane displays the gallery's image collection.

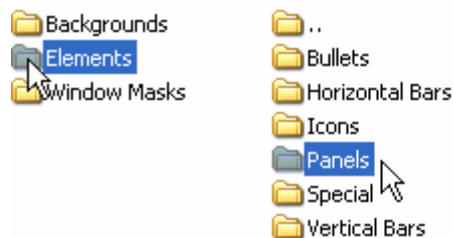


Like the Gallery tab on the Select File dialog, the Gallery pane remembers the last folder you accessed for each file type it contains. The last time you used the Gallery pane to add an image was in Lesson 2, when you used it to drag a background onto the page. Unless you've navigated somewhere else with it since then, the Gallery should still be "in" the Backgrounds folder.

To navigate to the Panels folder, you need to first move "up" one level by double-clicking on the up folder.



Once you've moved up one level, you can proceed to navigate into the Elements folder, and then into the Panels folder.



Note: Navigating in the Gallery pane is just like navigating in the Select File dialog, or in Windows. To reach the files inside a folder, just double-click on it.

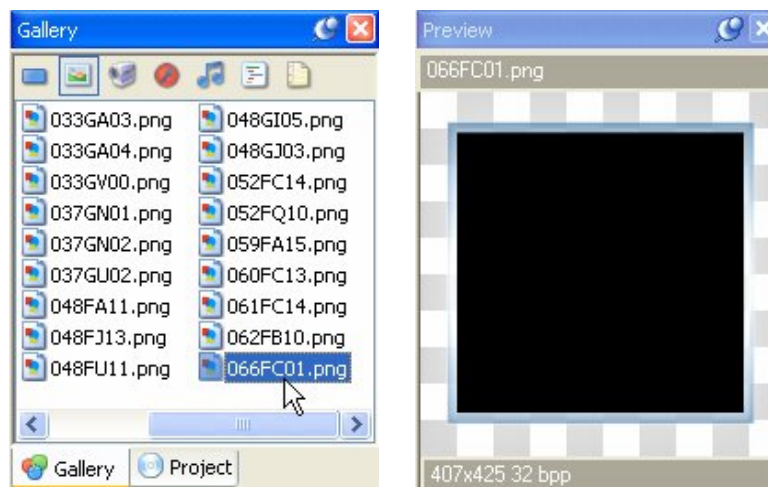
3) Preview some of the images in the Panels folder by clicking on the filenames.

As you can see, panel images come in a lot of different shapes, colors and textures. This is just a tiny sample of the panel images that are included in the add-on content packs that you can purchase from the Indigo Rose website.

4) Select the 066FC01.png file and drag it onto the page.

When you're picking an image to serve as a background for text, look for one that has a smooth texture, since "busy" patterns will make it harder to distinguish the letters. You want to pick one with colors that will contrast well with the color of the text. In this case, we'll be using white text, so we don't want to pick anything too bright; white text on a bright background would be difficult or even impossible to read.

A good candidate for this project would be 066FC01.png. It has a solid black background in the middle that will contrast nicely with white text, and a nice shiny blue "frame" around it that will look good against the page background we're using.

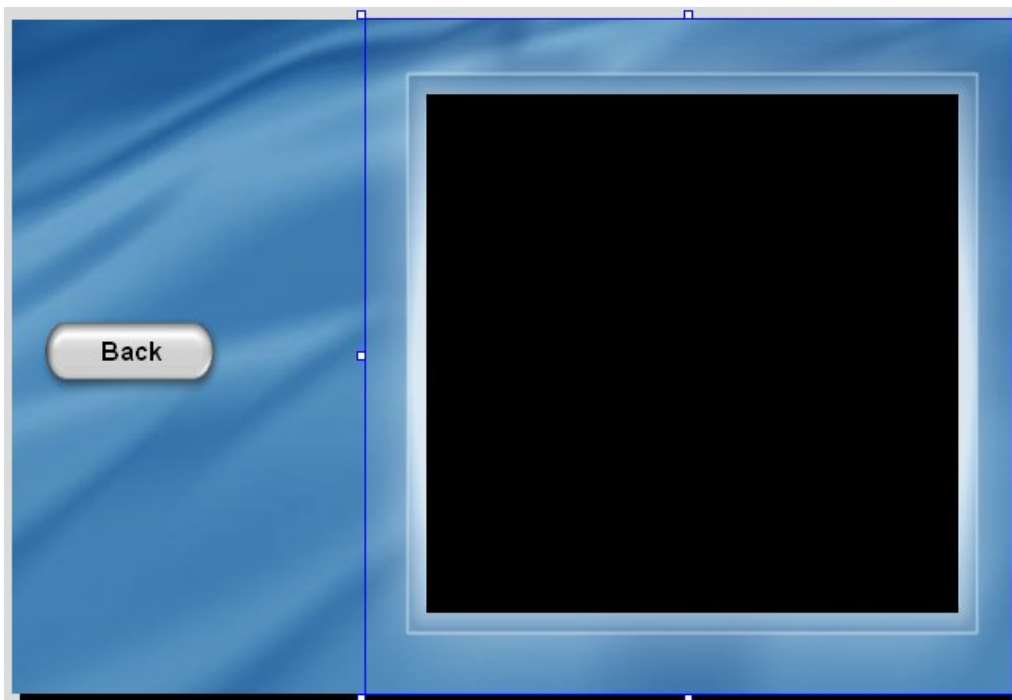


It also has a subtle "shine" effect around the edges of the frame. In order to achieve this lighting effect, the image was made a little bit larger than the frame on all sides. In fact, the image was sized to fit perfectly on a page that is 425 pixels tall—just like the pages in our project. This makes it really easy to position the image on our page.

5) Use the alignment tools to center the image object vertically on the page and align it with the right edge of the page.

First, make sure Align to Page mode is on. (An easy way to do this is to right-click on the image, and choose Align. If Align to Page mode is on, the To Page item will have a box around it.) Then, right-click on the object and choose Align > Center Vertical. Finally, right-click on the object and choose Align > Right.

Voila! The image is now positioned perfectly on the right side of the page.



Tip: If View > Snap to Page is on, you can also just drag the object into place with the mouse, and it will “snap” into place as you approach the top right corner of the page.

6) Right-click on the image object and choose Pin.

Pinning the image object will make it easier to work “around” the object without moving it by accident. (We’ll be placing a paragraph object right on top of the image object.)

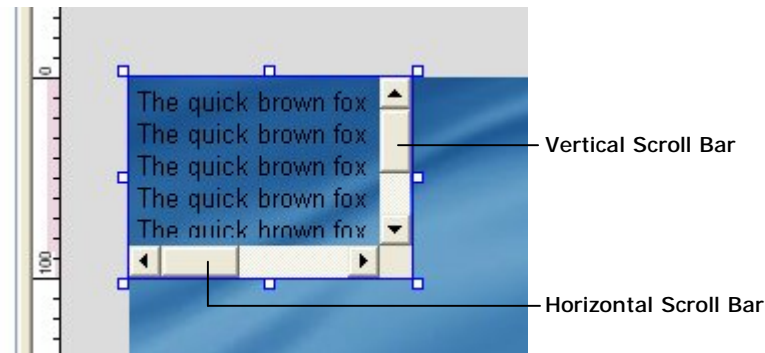
Tip: You can also pin an object by selecting it and pressing Ctrl+P.

Adding a Scrollable Paragraph Object

The paragraph object has the built-in ability to display text that is longer or wider than the object itself, with scroll bars to let the user slide the “hidden” parts into view.

In fact, all you need to do to make a paragraph object scrollable is to put more text in it than it has room to display. (By default, new paragraph objects have their vertical scroll bar set to “Auto,” which means they will automatically display a vertical scroll bar if the text is long enough.)

There are two basic kinds of scroll bars: vertical, and horizontal. (Vertical scroll bars let you scroll text up and down, while horizontal scroll bars let you scroll text left and right.)



Vertical scroll bars are useful when you’re displaying long text documents—in fact, they let you put as much text in a paragraph object as you want. The size of the paragraph object just determines how many lines of text the user can see at any given time.

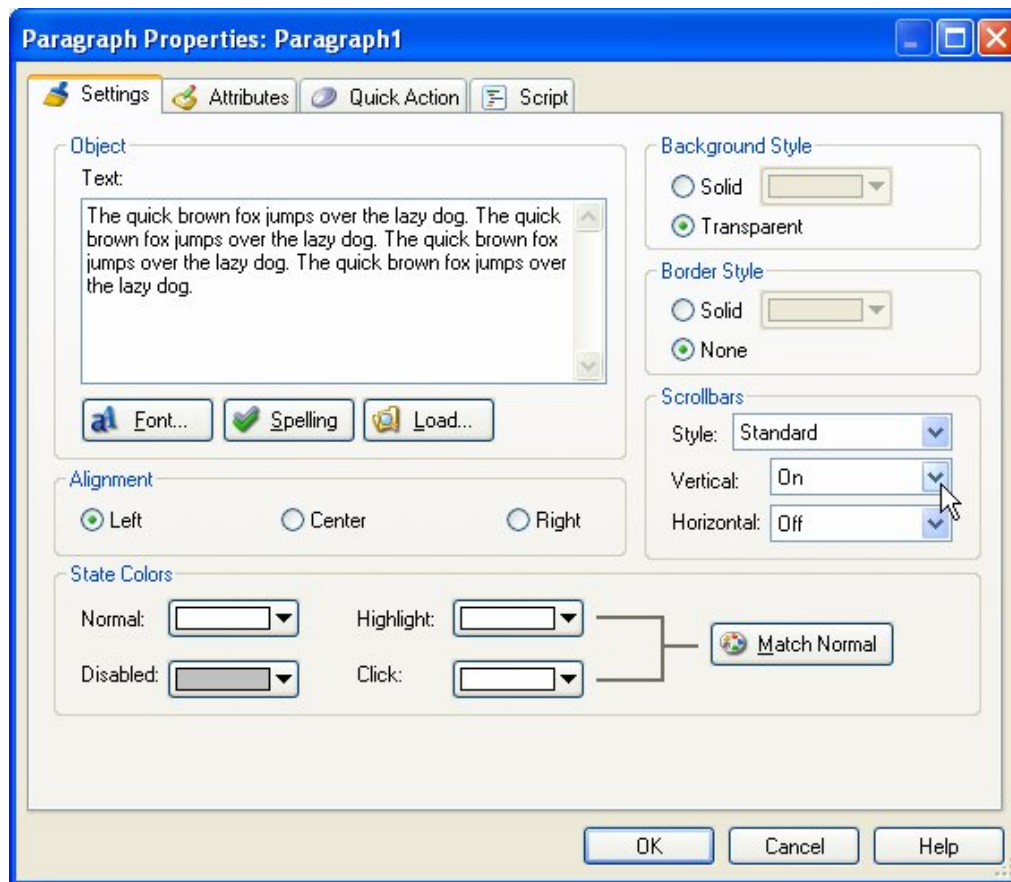
Horizontal scroll bars are useful when you’re displaying text with very long lines, and you don’t want the text to “wrap” at the right edge of the paragraph object. Instead, the user can drag the horizontal scroll bar left and right to see all of the text on a line. Although most people find this disconcerting, it can be useful for text that isn’t meant to wrap, such as text arranged in fixed-width columns, or program code samples.

Note: Most users don’t like having to scroll text horizontally. You should avoid using horizontal scroll bars unless you have a good reason to.

1) Press Ctrl+3 to add a new paragraph object. Double-click on the new object, set its Normal color to white, and click the Match Normal button.

We want the paragraph object to have white text, and we don't want the color to change whenever the user moves the mouse over it, so we need to make sure the Highlight and Click colors both match the Normal color.

2) Change the Vertical scroll bar setting to "On" and click OK.



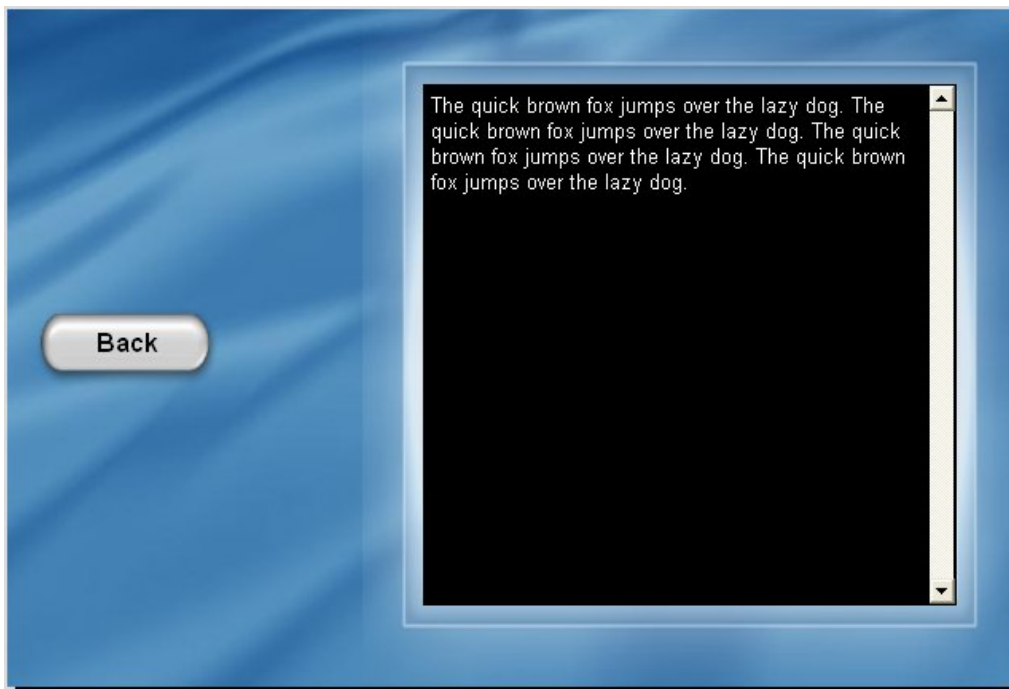
After you click OK, the paragraph object that was added in the upper left corner of the page will reflect the new settings. Notice that the sample text is in fact white, and the object has a vertical scroll bar.

Forcing the vertical scroll bar on regardless of the object's size will make it easier to

position the object in the next step, since the scroll bar affects the amount of room the text has, which in turn can affect where you want to place the object to “center” it on top of the panel image.

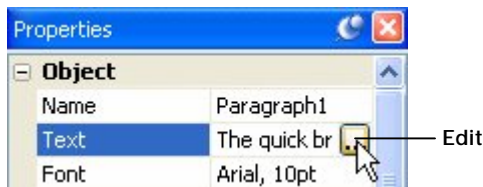
3) Position the paragraph object over the panel image and resize it so it fills most of the black space inside the frame.

You want the paragraph object to be smaller than the panel image, so it will fit inside the blue frame that is built into the image.

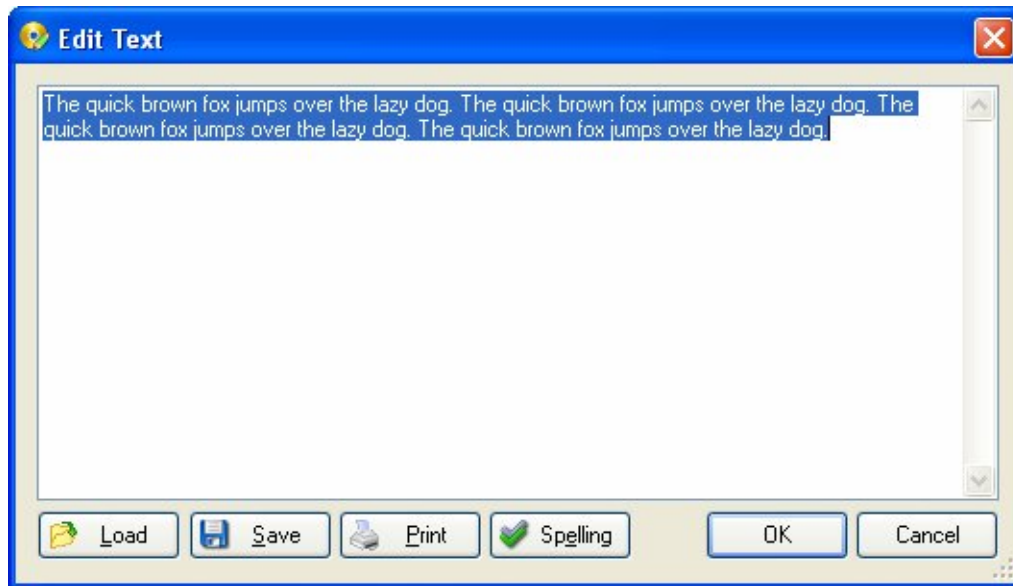


Now let's load some text into the paragraph object.

4) In the properties pane, click on the Text setting, and then click the edit button.



When you click on the edit button, the Edit Text dialog appears.

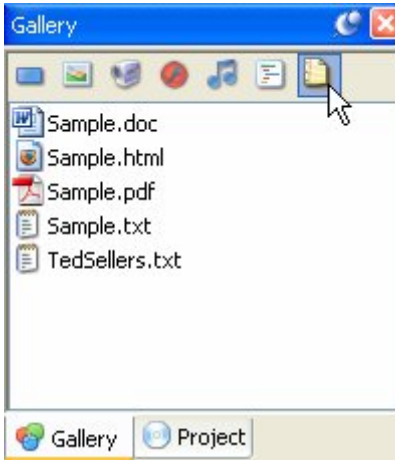


At the bottom of the Edit Text dialog, there are four buttons: Load, Save, Print and Spelling.

- The Load button lets you import text from any text file on your system.
- The Save button lets you save the current contents of the Edit Text dialog to a text file.
- The Print button lets you print the current contents of the dialog.
- The Spelling button interactively checks the spelling of the text, letting you choose from a number of suggested spellings for any misspelled words.

We could use the Load button to load a text file into the paragraph object, but since the file we want to load is in AutoPlay's gallery, there's an even easier way.

5) Click Cancel to exit the Edit Text dialog, and click the Docs button at the top of the Gallery pane.



Clicking the Docs button displays the sample document files that are shipped with AutoPlay. The file we want to load into the paragraph object is TedSellers.txt.

6) While holding the Shift key, drag the TedSellers.txt file onto the paragraph object.

Normally, when you drag a text file onto the page, a new paragraph object is created for you automatically—with the entire contents of the text file inside it.

That's pretty darn cool. In fact, we could have used that method to create the paragraph object in the first place.

However, if you hold the Shift key down as you drag a text file onto a paragraph object, you'll *replace* the text in the paragraph object with the text from the file.

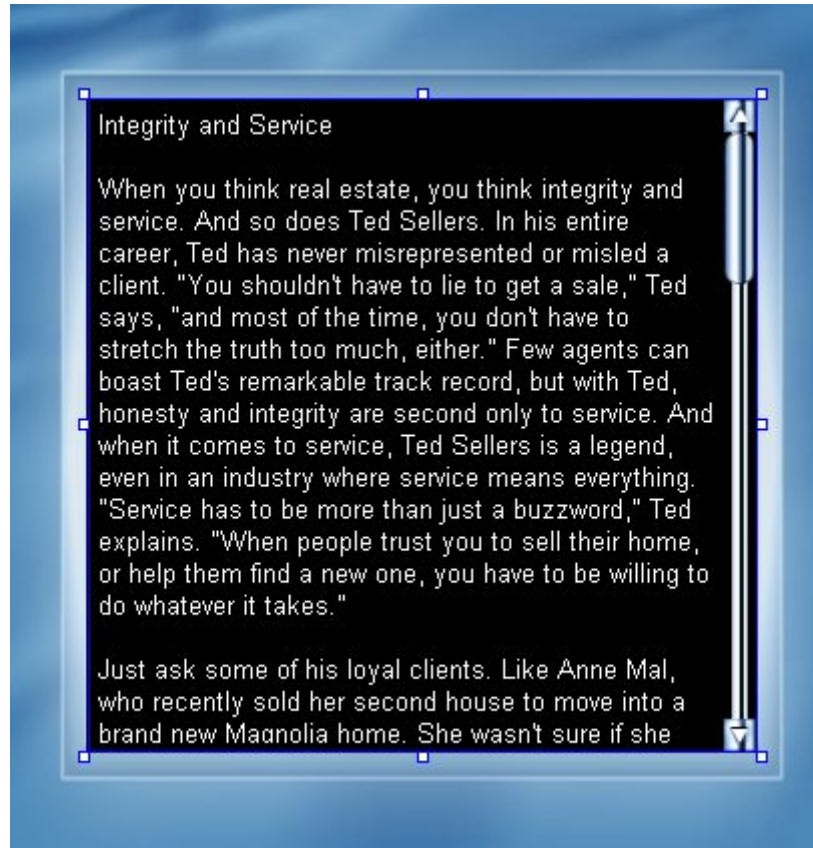
Which you have to admit, is also pretty nifty.

When you drag the TedSellers.txt file onto the paragraph object with the Shift key held down, the contents of the TedSellers.txt file are instantly loaded into the paragraph object.

Tip: There are all sorts of little time-saving tricks like this built into AutoPlay. If you want to slip a few more of them up your sleeve—you know, just in case—search for “Drag and Drop Assistant” in the help file.

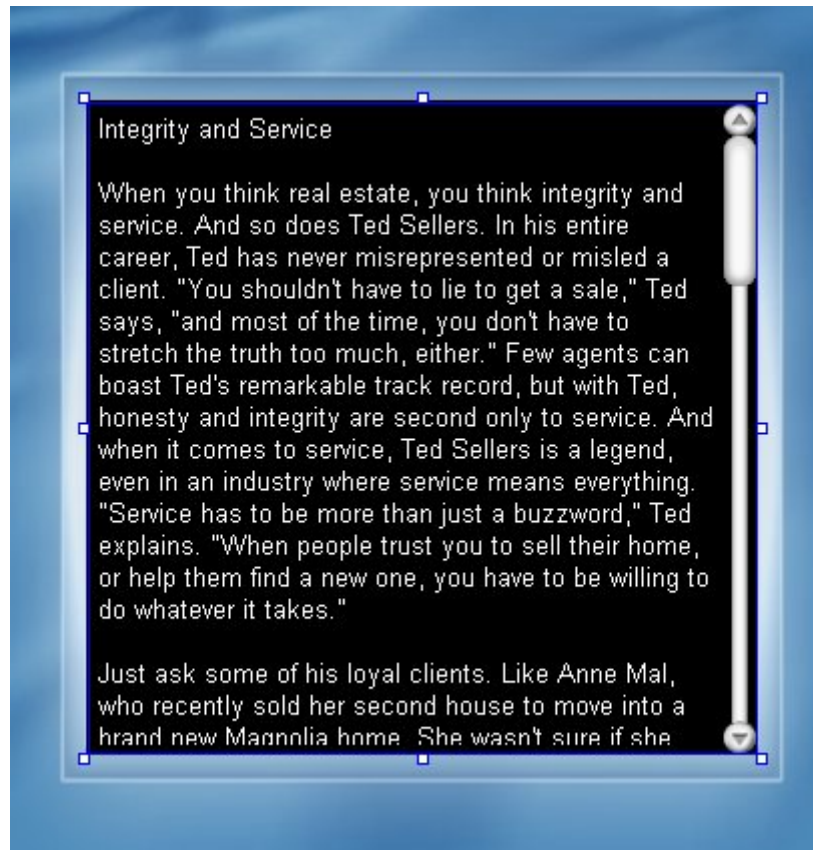
7) In the properties pane, change the ScrollbarStyle setting from “Standard” to “Chrome.”

The paragraph object allows you to choose between a standard, Windows-style scroll bar, or a custom scroll bar style, like the cool Chrome one that comes with AutoPlay.



8) Try some of the other scroll styles, and select one that looks good with the panel image you chose.

Go ahead and pick whichever scroll bar style looks best to you. (I chose Corporate, since it looks good with the dark background of the panel image we're using.)



Each scroll bar style is actually just an image file with different “scroll bar parts” arranged in a row. Each individual part of the scroll bar is represented by a different section of the image. AutoPlay extracts the parts from this image, and then uses them to build the scroll bar.



The Corporate scroll bar skin

This kind of image file containing different interface parts is usually referred to as a *skin file*. Indeed, the scroll bar styles are similar to the skin files used by other “skinnable” programs, like Winamp.

Note: AutoPlay’s scroll bar images are located in the Plugins\Scrollbars subfolder, inside the folder where AutoPlay was installed. You can add more styles to the list of available scroll styles by copying compatible images into that folder.

Tip: You can also build your own scroll bar skins for AutoPlay using a paint program. The easiest way is to use one of the existing scroll bar images as a template. Simply make a copy of an existing skin file, and then modify the copy, replacing each section of the image with your own corresponding scroll bar part.

Trying It Out

Now that we have everything set up the way we want, let’s see some scrolling text in action.

1) Save the project.

It’s always a good idea to save the project before you preview it.

2) Preview the project. When the preview application opens, click on the About Ted Sellers button to go to the About Ted page. Try scrolling the text in the paragraph object up and down.

Pretty cool, isn’t it!

3) Exit the preview.

That’s it for this lesson.

Lesson 6 Summary

In this lesson, you learned how to:

- Add a panel image to serve as a backdrop and border for some text
- Add a scrollable paragraph object
- Replace the text in a paragraph object by shift-dragging a text file
- Use a custom scroll bar style



Lesson 7:

Video

If a picture's worth a thousand words, how many words is a video worth?

In this lesson, I'll show you how to add a video to your project, cover some of the basic video settings, and teach you how to control a video using a handful of simple actions.

7

What You'll Learn

In this lesson, you'll learn how to:

- Add a panel image to frame a video
- Add an attractive text banner to the page
- Add a video object
- Customize the video object's built-in control panel
- Make your own video controls from scratch
- Control the video with a few simple actions

How Long Will It Take?

This lesson takes approximately 25 minutes to do.

Starting the Lesson

If you're continuing from Lesson 6, you should still have AutoPlay running with the Tutorial project open. If so, you're ready to move on to the next exercise: Adding a Text Banner.

Otherwise, you'll need to open the project file that you saved at the end of Lesson 6.

1) Open the Tutorial.am7 file that you saved in Lesson 6.

When you save a project, AutoPlay automatically creates a project folder for it inside your "My Documents\AutoPlay Media Studio 7.0\Projects" folder. This project folder is where everything that belongs to the project is stored—including the project file, which contains all of the settings used in the project.

The project folder and the project file always have the same name that you gave to the project when it was created. The project file's name ends with a ".am7" file extension.

Since you named this project "Tutorial" in Lesson 1, the name of the project folder will be Tutorial, and the name of the project file will be Tutorial.am7. So, the path to the project file should be something like:

```
...\My Documents\AutoPlay Media Studio 7.0\Projects\Tutorial\Tutorial.am7
```

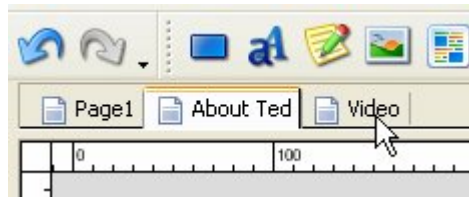
To open the project, you just need to open that project file.

Adding a Panel Image

As you saw in the previous lesson, a panel image is just an image object that you put behind another object to achieve some kind of custom background or picture frame effect. Let's use another panel image to provide a "frame" for our video object.

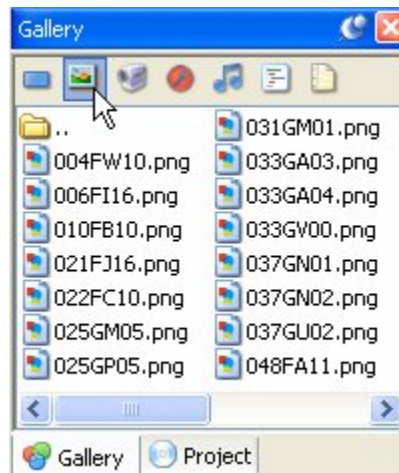
1) Switch to the Video page.

First things first. We're going to add this panel image to the Video page, so click on the "Video" page tab at the top of the work area.



2) Click the Images button on the Gallery pane.

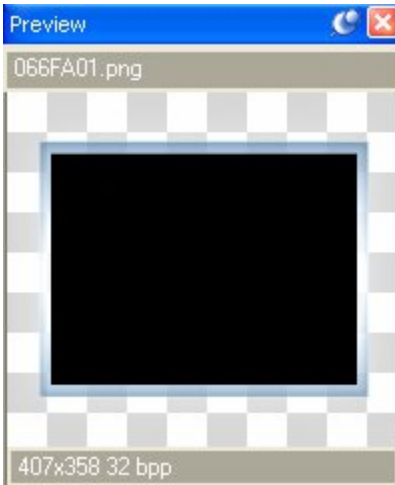
Clicking the Images button at the top of the Gallery pane displays the collection of images the gallery.



Since you used the Gallery pane to add an image from the Panels folder in the previous lesson, the Images section of the Gallery pane should already be displaying the contents of the Panels folder.

3) Select the 066FA01.png file and drag it onto the page.

The 066FA01.png file is similar to the 066FC01.png panel image that we used in Lesson 6, but its frame is designed to fit around a standard 320x240 video.



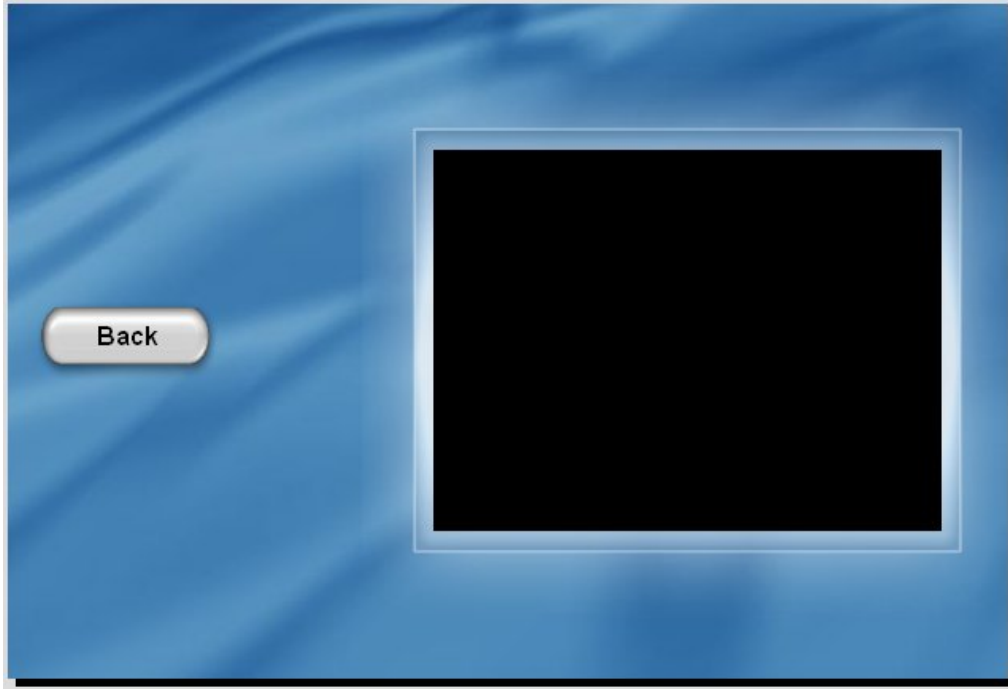
4) Use the alignment tools to center the image object vertically on the page and align it with the right edge of the page.

Make sure Align to Page mode is on. (An easy way to do this is to right-click on the image, and choose Align. If Align to Page mode is on, the To Page item will have a box around it.)

Then, right-click on the object and choose Align > Center Vertical.

Finally, right-click on the object and choose Align > Right.

Voila! The image is now positioned perfectly on the right side of the page.



5) Right-click on the image object and choose Pin.

Pinning the image object will make it easier to work “around” the object without moving it by accident.

Tip: You can also pin an object by selecting it and pressing Ctrl+P.

Adding a Text Banner

The video we’re going to add is a short presentation about a new home design called the “Magnolia.” To help introduce the video, let’s place an attractive text banner across the top of the page.

In order to make the text stand out, we’ll use a paragraph object with white text on a blue background.

1) Right-click on the page surface and choose Paragraph.

Make sure you right-click on the actual page surface, and not on part of the panel image. (If you select the panel image by mistake, try right-clicking further to the left.)

Choosing Paragraph from the right-click menu will add a paragraph object to the page.

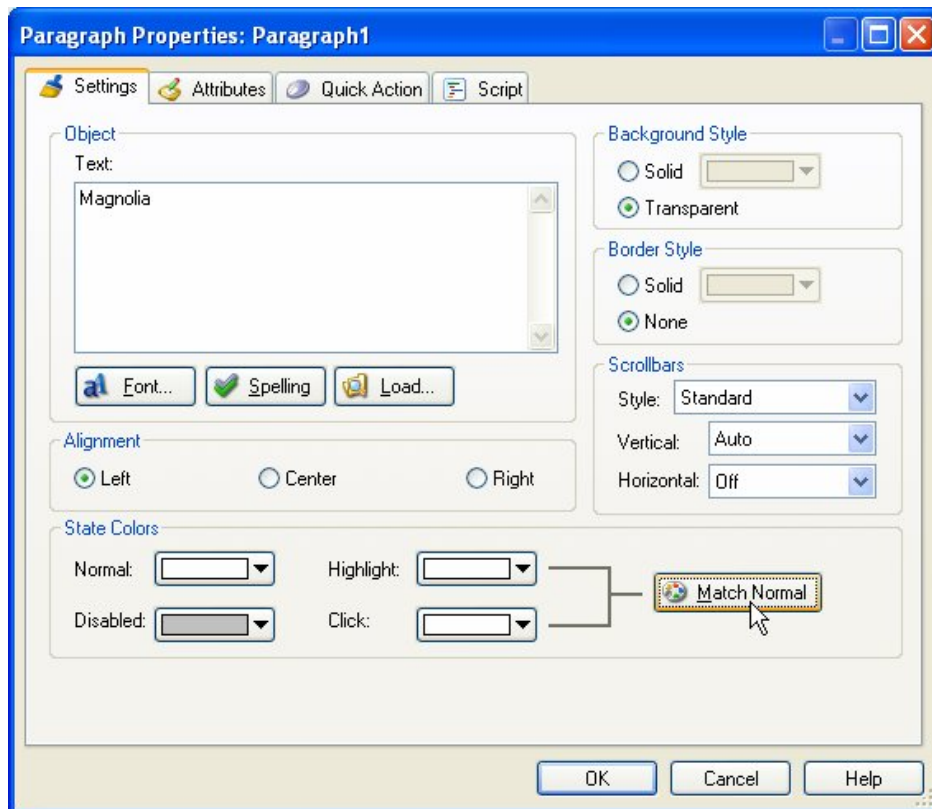
2) Double-click on the new paragraph object to open the Paragraph Properties dialog. Change the object's text to *Magnolia*.

When you double-click a paragraph object, all of the text in the Text setting is automatically highlighted for you, so you should be able to just type the text right in.

Note: You want this text to replace the existing default text, so make sure the old text is highlighted before you type the new text in.

3) Set the Normal color to white, and click the Match Normal button.

We want the paragraph object to have white text, and we don't want the object to appear interactive, so we need to make sure the Highlight and Click colors both match the Normal color.



4) Click OK to close the Paragraph Properties dialog.

When you click OK, the Paragraph Properties dialog closes and the new paragraph object appears where you right-clicked on the page.

5) In the properties pane, change the Alignment setting from “Left” to “Center.”

A quick way to change this setting is to double-click it.

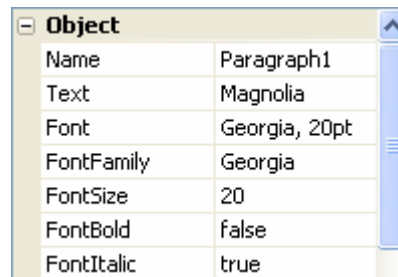
Double-clicking the Alignment setting advances it to the next item in its list.

Since paragraph objects start out with their Alignment set to “Left” by default, double-clicking on the Alignment setting switches it to “Center.” (If you double-clicked again, it would change from “Center” to “Right.”)

This is a quick way to change settings that give you two or three options to choose from.

6) Change the FontFamily to Georgia, FontSize to 20, and FontItalic to “true.” Leave FontBold set to “false.”

We want the paragraph object to use an italicized, unbolded, 20-point Georgia font.



| Object | |
|------------|---------------|
| Name | Paragraph1 |
| Text | Magnolia |
| Font | Georgia, 20pt |
| FontFamily | Georgia |
| FontSize | 20 |
| FontBold | false |
| FontItalic | true |

7) Set BorderStyle to “Solid” and BackgroundStyle to “Solid.” Leave the BorderColor black (#000000), and set BackgroundColor to #5987C0.

In order to see the object’s background color, you need to set its BackgroundStyle setting to “Solid.” Setting BorderStyle to “Solid” adds a thin black border around the object.

Note: The border color is already set to black so it doesn’t need to be changed.

To set the object’s background color, double-click on the BackgroundColor setting, type #5987c0, and press Enter.

8) In the Scrollbars category, set Vertical to “Off.”

We don't want the paragraph object to have a vertical scroll bar. Although we'll be making the paragraph object larger than the text inside it, which should prevent the scroll bar from appearing even if it's set to "Auto," it doesn't hurt to turn the scroll bar off just to be sure.

9) Set Left to 255, Top to 17, Width to 345 and Height to 44.

This will place the paragraph object above the panel image, near the top of the page.

10) Right-click on the paragraph object and choose Lock.

Locking the object will keep it out of your way for the rest of this lesson.

Adding a Video Object

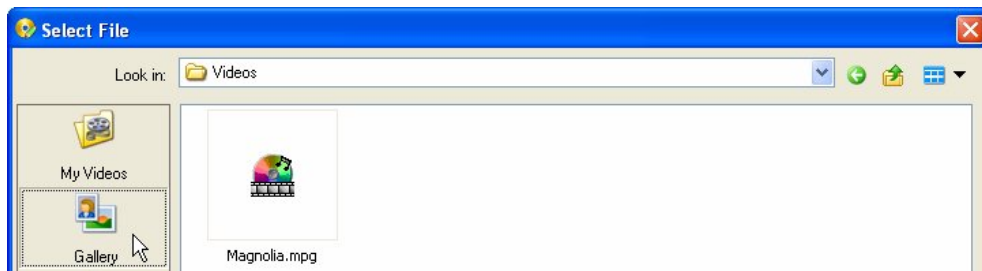
AutoPlay's video object lets you display full-motion video right on the page. You can use it to show all sorts of media files, from training videos and product previews, to home movies and music videos.

Tip: You can also use a File.Open or File.Run action to display a video in an external viewer program. Of course, displaying a video right on the page results in a more seamless user experience.

Note: Video objects will always appear on top of other kinds of objects, no matter how you arrange the objects on the page (i.e. regardless of the z-order). There are technical reasons why this is so, but suffice it to say that it has to do with the way videos are drawn on the screen.

1) Choose Object > Video. When the Select File dialog appears, click the Gallery button.

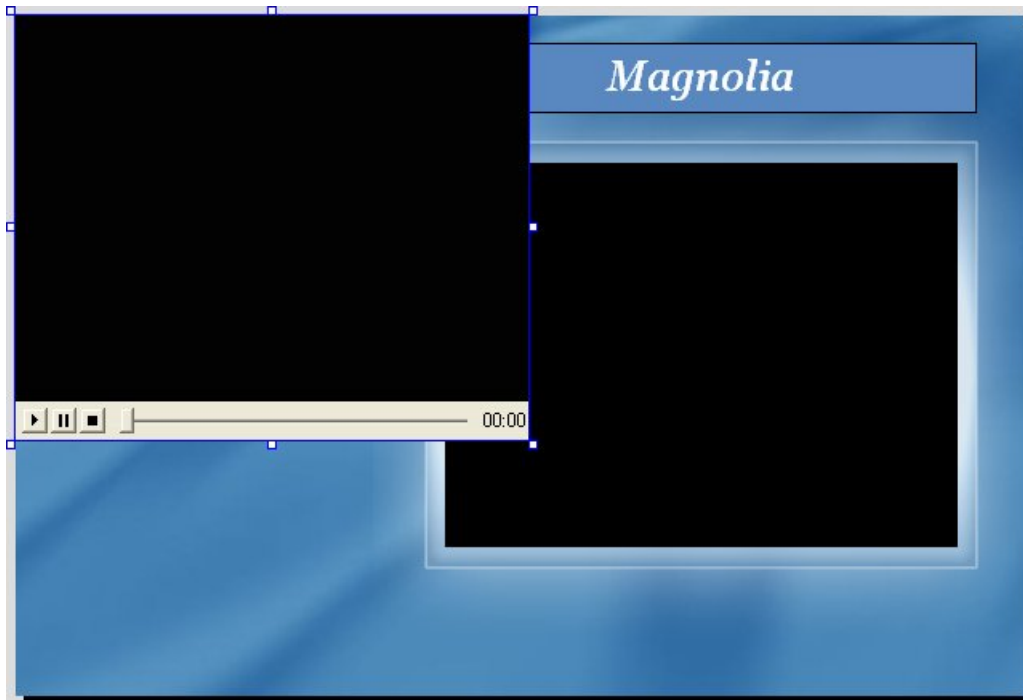
Choosing Object > Video opens the Select File dialog. Clicking the Gallery button allows you to access a small sample video that comes with AutoPlay.



2) Select the Magnolia.mpg file, and click OK.

After you click OK, the new video object appears in the upper left corner of the page. A single frame from the video appears in the object, so that as you work on the project you can have a general idea of what it's going to look like when it starts.

(In this case, the frame happens to be black, since the Magnolia.mpg video begins with a fade from black.)



Notice the standard video controls at the bottom of the video object. This is the object's *control panel*. On the control panel, there is a play/pause button, a stop button, a position slider, and an elapsed time readout.

Note: The play button toggles between play and pause: it switches to the pause button when the video is playing, and then switches back to the play button when the video is paused.

3) In the properties pane, change the ControlStyle setting from “Standard” to “Basic Blue.”

The ControlStyle setting lets you select from a list of control panel styles. You can use this setting to change the appearance of the built-in video controls, or to hide the control panel completely.



Each style in the list corresponds to a skin file located in the Plugins\Transports subfolder, inside the folder where AutoPlay was installed.

A skin file is just an image with different “control panel parts” arranged in a row. Each individual part of the control panel is represented by a different section of the image. AutoPlay extracts the parts from this image, and then uses them to build the control panel.

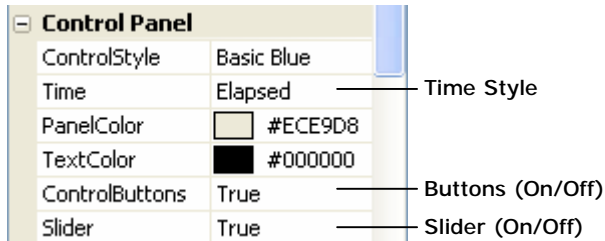


The Basic Blue control panel skin

Tip: You can build your own control panel skins for AutoPlay using a paint program. The easiest way is to use one of the existing images as a template. Simply make a

copy of an existing skin file, and then modify the copy, replacing each section of the image with your own corresponding control panel part.

4) Try changing the settings inside the Control Panel category.



The Control Panel category has six settings inside it: ControlStyle, Time, PanelColor, TextColor, ControlButtons, and Slider. You can use these settings to show or hide the different parts of the control panel and adjust what they look like.

Note: The control panel, and the controls on it, are all completely optional. You can choose to hide the whole control panel, or hide any of the individual parts.

Experiment with these settings to see some of the different control configurations you can come up with.

5) In the Special category, check that AutoStart is set to “True,” and that Loop and Border are both set to “False.”

The AutoStart setting controls whether the video will begin playing automatically whenever the page it’s on is shown. We want the video to start playing as soon as the user jumps to the Video page, so make sure this is set to “True.”

The Loop setting controls whether the video will automatically restart from the beginning when it reaches the end. In this case, we just want the video to stop when it reaches the end, so leave it set to “False.”

The Border setting controls whether the video object will have a black border around it. You can turn it on if you want, but I think this video looks fine without a border.

In short: make sure all these settings are set to their default values. (Okay, I’ll admit it...this step was just an excuse to point out what those settings do.)

6) In the Object category, make sure VideoScalingMode is set to “Maintain Aspect.”

Video Scaling Mode controls whether the video will maintain its original aspect ratio, or will “stretch” to fill the size and shape of the bounding box. Stretching the video usually makes it look much worse, but it can be helpful if you *really* don’t want any black bars around the video, and don’t want to resize the object to make the black bars go away. (It’s usually better to keep Scaling Mode set to “Maintain Aspect,” and just adjust the size of the video object instead to eliminate any black bars.)

Tip: Preserving the original aspect ratio is important for image files and videos, which often look distorted if they’re stretched disproportionately. An easy way to preserve an object’s aspect ratio is to right-click on the object and choose Keep Aspect. (If you need to, you can restore the object’s original aspect ratio by right-clicking on it and choosing Restore Size.)

7) In the Position category, set Width to 320, and Height to 240. Position the object so it fits just inside the frame of our panel image. Once it’s in place, press Ctrl+P to pin the object.

The left, right, and bottom sides of the video object will probably need to overlap the frame of the panel image by just a smidgeon. But that’s okay.

Tip: A trick for setting the object’s position precisely is to drag it more or less into place with the mouse, and then move it in tiny increments with the cursor keys. By moving the object back and forth across the inside edge of the frame, one pixel at a time, you can see the exact position where the top of the object matches the top of the black background on the panel image.

If you’d rather just set the object’s position directly, set Left to 268, and Top to 93.

Once it’s in place, pressing Ctrl+P will pin it so that it can’t be moved or resized any more.

Note: You don’t *have* to pin the object, but this is something I like to do when I have an object where I want it. I find it easier to work around objects if I don’t have to worry about moving them by accident.

8) Choose Page > Preview to preview the current page.

Choosing Page > Preview temporarily builds and displays the current page, so you can see how it will look and work when it is seen as part of the application.

Tip: You can also preview a page by right-clicking on its page tab and choosing Preview.

Note that the video starts playing automatically as soon as the page is shown. This is because the video object's AutoStart setting is set to "True." (The same thing will happen each time the users navigate to this page in the full application.)

Previewing the page is often faster than previewing the whole project, because AutoPlay doesn't have to build the whole project—it only has to build one page. More importantly, you don't have to navigate all the way from the start page just to get to the page that you want to test. (Which also means you can try out a page before you've added any Page.Jump actions to get to it from the start page.)

9) With the preview application selected, press Alt+F4.

Pressing Alt+F4 when the preview application is selected (i.e. when it's the "foreground application" in Windows) immediately closes the application.

You can use this method of exiting from the preview when your application doesn't have a title bar.

Of course, since your application does have a title bar, you can also exit from the page preview by clicking the Close button.



Adding Custom Video Controls

In addition to the built-in video controls on the video object's control panel, you can use AutoPlay's video actions to provide your own "homemade" video controls.

1) Select the video object. In the properties pane, set ControlStyle to "None."

Changing the ControlStyle setting from "Basic Blue" to "None" removes the control panel from the object completely.



2) Select the "Back" button object (Button3) and press Ctrl+D to duplicate it. Press Ctrl+P to unpin the new object, and move it down below the video object.



Now, we *could* just add a new button object, and set it up to be like the other buttons in the project, but using the Back button as a template is much easier. This way, we don't have to set the object's Normal, Click and Highlight colors—they're already set for us.

Note that the new object's name is Button1, since 1 is the first number that isn't used in a button object's name on this page.

Wait a minute...isn't this the *second* button that you've added to this page? Indeed it is! But since the name "Button1" hasn't been taken yet, it gets used first. Remember how the "Back" button carried its name over when you copied and pasted it in lesson 4? Since that button's name is Button3, this leaves the first two default names "open" for new button objects to use on this page.

3) In the properties pane, change the object's Width to 55 and its Height to 27. Change the font Family to Verdana, and the font Size to 6. Set Bold and Italic both to "True." Change the object's Text to *PLAY*.

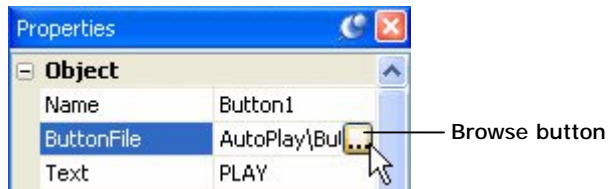


This is going to be our play button. Setting the font to 6-point Verdana makes the text small enough to fit on a very small button, without being too hard to read. (Making the text use only capital letters helps get the most visibility out of such a small font.)

4) Change the object's ButtonFile setting so it uses the green_pill.btn file instead of the grey_pill.btn file.

Without changing any of the object's other settings, we want to make this button object use the green_pill.btn file instead of the grey_pill.btn file.

To do so, first click on the ButtonFile setting to make the browse button appear.



Next, click the browse button to open the Select File dialog. If you need to, switch to the gallery in the Select File dialog by clicking the Gallery button. Locate the `green_pill.btn` file in the list, select it, and click OK.

Tip: You can also just double-click on the `green_pill.btn` file to select it and close the Select File dialog all at once, without having to click OK.

Instead of a grey pill-shaped button, you should now have a green one.

5) In the properties pane, set YOffset to 1.

The YOffset setting allows you to fine-tune the vertical position of the text on the button object. It specifies the number of pixels that the text should be offset vertically from its default position.

Sometimes, changing an object's size or font settings will cause the text to be rendered off-center. In this case, the text was being drawn a little bit too high with the default offset of 0. Setting the Y offset to 1 moves the text down by one pixel, so that everything is in its right place on the object again.

6) Click the Buttons category at the top of the Gallery pane. Locate the orange_pill.btn file in the list of buttons.

Clicking the left-most button at the top of the Gallery pane displays the collection of very cool button files that was installed with AutoPlay. Nestled cozily near the end of this list is the `orange_pill.btn` file.

7) Click on the PLAY button, and press Ctrl+D to duplicate it. Change the new object's Text to PAUSE. While holding the Shift key down, drag the orange_pill.btn file from the Gallery pane onto this button object.

This time, instead of changing the ButtonFile setting directly, let's take advantage of AutoPlay's drag and drop assistant. In AutoPlay, if you hold down the Shift key while you drag a button file onto a button object, you'll "insert" that button file into the button object. It basically replaces the object's current button file with the one you dragged onto it, and changes the ButtonFile setting for you.

We call this kind of drag operation *shift-dragging*.

Tip: You can do the same with other objects, as well. For example, you can replace the image that is shown in an image object by dragging a new image onto it with the Shift key held down. And you can replace the text in a paragraph object by shift-dragging a text file onto it.

8) Click on the PAUSE button, and press Ctrl+D to duplicate it. Change the new object's Text to STOP, and make it use the red_pill.btn file.

Once again, you want to change the button file without affecting any of the object's other settings. This time, you should end up with a red STOP button.

Note: To make the button object use the red_pill.btn file instead of the green_pill.btn file, either change the ButtonFile setting directly, or use the shift-drag method .

You should now have a total of three little button objects: one with the word PLAY on it, another with the word PAUSE on it, and a third with the word STOP.



Note that the name of the third button—the one with the word STOP on it—is not Button3, but Button4. This is because the name “Button3” was already taken by the Back button. So, AutoPlay skipped ahead to the next available default name for a button object on this page, which in this case was “Button4.”

9) Turn off Align to Page mode.

Make sure Align to Page mode is turned off. (An easy way to check this is to right-click on an object, and choose Align. If Align to Page mode is on, the To Page item will have a box around it.)

If Align to Page mode is on, choose View > Align > To Page to turn it off.

10) Drag the STOP button to the right a bit. Select all three button objects, make the PLAY button dominant, and choose Align > Top, followed by Align > Distribute Horizontal.

The goal is to get the three objects lined up horizontally and distributed evenly, with PLAY on the left and STOP on the right, like this:



11) Group the three button objects together and center them beneath the video object.

Select all three button objects, and press Ctrl+G to group them. Then center the group horizontally beneath the video object.

You should end up with the PLAY, PAUSE and STOP buttons in a row below the video object, like this:



Notice how the button colors help convey what each button does, even if you don't read the text? Most people are used to seeing green, amber and red traffic lights, so having these three colors together borrows from that common context. (To a lesser extent, people are also used to seeing Play/Pause/Stop buttons arranged in a standard order, with play on the left, then pause, then stop.)

Tip: You aren't limited to buttons with text on them, either. If you'd rather have buttons with iconic symbols for things like play, pause and stop, you can easily create your own buttons using the AutoPlay Media Studio Button Maker, which you can access by choosing Tools > Button Maker.

12) Ungroup the button objects and pin them.

Now that you have the buttons where you want them, ungroup the objects by selecting one of them and pressing Ctrl+Shift+G, and then pin them by pressing Ctrl+P.

Ungrouping the objects will make it easier to select them independently in the next exercise. Pinning the objects will prevent them from being repositioned by accident.

Taking Control of the Video with Actions

Now that we have our PLAY, PAUSE and STOP buttons laid out on the page, let's make them actually play, pause and stop the video.

To do that, we just need to add an action to each button.

1) Click on the page surface to deselect the objects.

Clicking on the page surface deselects the objects, so that none of the objects on the page are selected. (Make sure you click somewhere on the page surface where there isn't an object. If you select an object by accident, just click again somewhere else.)

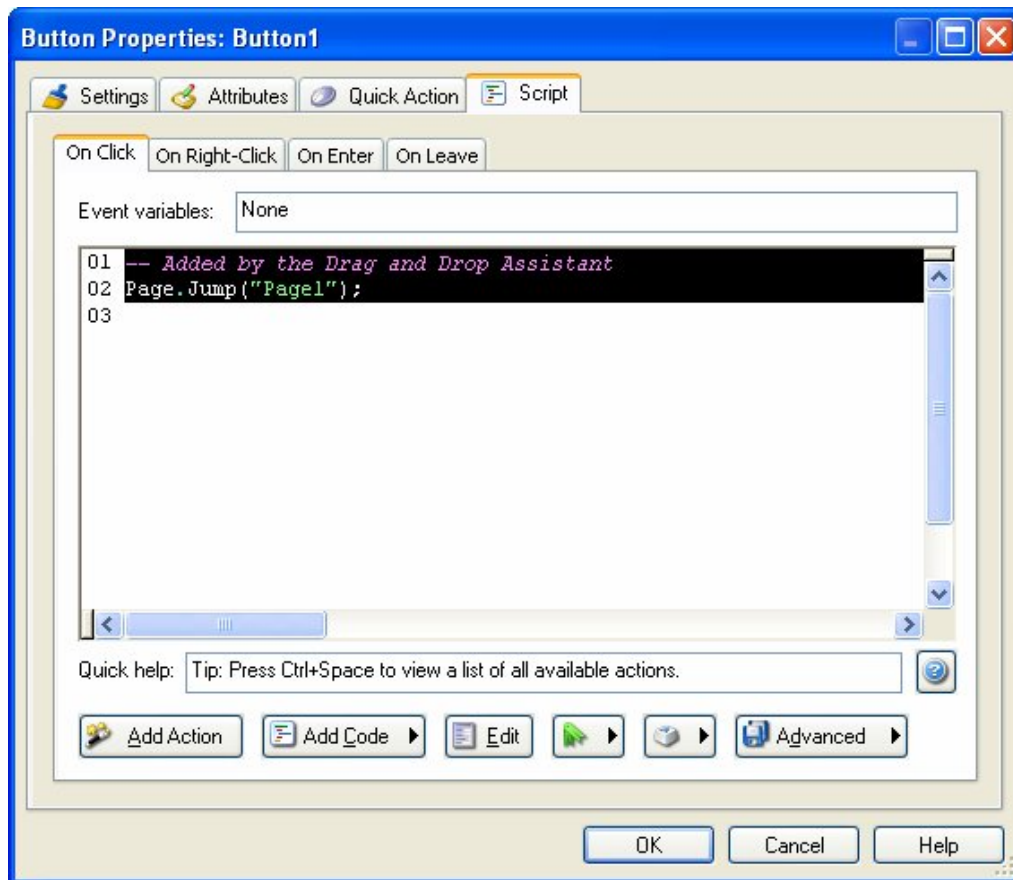
2) Double-click on the PLAY button, and click on the Script tab.

Double-clicking on an object opens up the Properties dialog for that object. Clicking on the Script tab switches to the script editor so you can edit the object's actions.

3) In the script editor, click on the On Click tab. Replace the Page.Jump action with a Video.Play action. Set the Video.Play action's ObjectName parameter to "Video1".

Note that this button object already has a Page.Jump action assigned to its On Click event. (The action was copied over from the Back button when you duplicated that object.) We're going to replace this Page.Jump action with a Video.Play action.

Replacing the action is easy enough—just remove the old action, and add another one in its place. To remove the old action, simply highlight all of its text on the script editor, and press the delete key.



Note: As you've probably noticed, actions are just text instructions that get interpreted by AutoPlay. You can edit these instructions directly, just as you would edit text in a Word document.

Once you've gotten rid of the old action, you can click the Add Action button to add a new Video.Play action in its place. (When the New Action wizard appears, select the Video category, then select the Video.Play action, and then click Next to advance to the wizard's second page where you can customize the action.)

The Video.Play action takes a single parameter, called ObjectName, which tells the action which video object to play. Since there is only one video object on the page, the ObjectName parameter should already be set to "Video1" for you.



Once you have the action configured, click Finish to add it to the script. Then click OK to close the script editor and accept these changes that you've made to the object's On Click event.

Tip: Since actions are just text, you could also add this action by typing `Video.Play("Video1");` directly into the script editor.

4) Double-click on the PAUSE button, and click on the Script tab.

AutoPlay actually remembers the tab that you were on the last time you double-clicked on an object, but it does so on a *per-object* basis. When you double-click on an object for the first time (like this PAUSE button), it always defaults to showing you the Settings tab.

Once you've visited another tab on the Properties dialog for any particular object, though, the next time you double-click on that object you'll be taken right to that tab.

Of course, that doesn't help us when we're double-clicking on a "fresh" object like the PAUSE button. Since this particular object hasn't been double-clicked before, you still need to click on the Script tab to switch to the PAUSE button's script editor.

5) In the script editor, click on the On Click tab, and replace the Page.Jump action with a Video.Pause action. Set the ObjectName parameter to "Video1".

This object also has a Page.Jump action on its On Click event. Once again, simply delete the existing action's text, and then click the Add Action button to insert a Video.Pause action in its place.

6) Double-click on the STOP button, and click on the Script tab. In the script editor, click on the On Click tab, and delete the existing action's text. Then type:

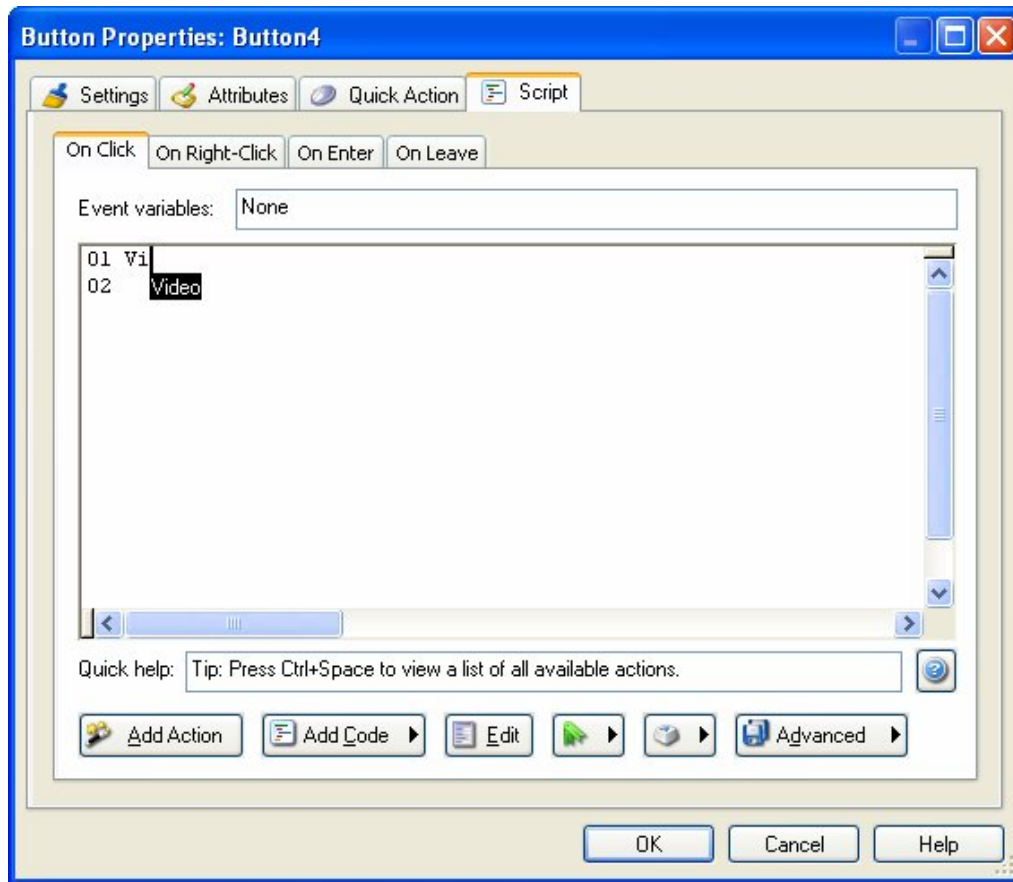
```
Video.Stop("Video1");
```

...into the script editor, right where the old action used to be.

Since actions are just text, you can also add an action by typing it directly into the script editor.

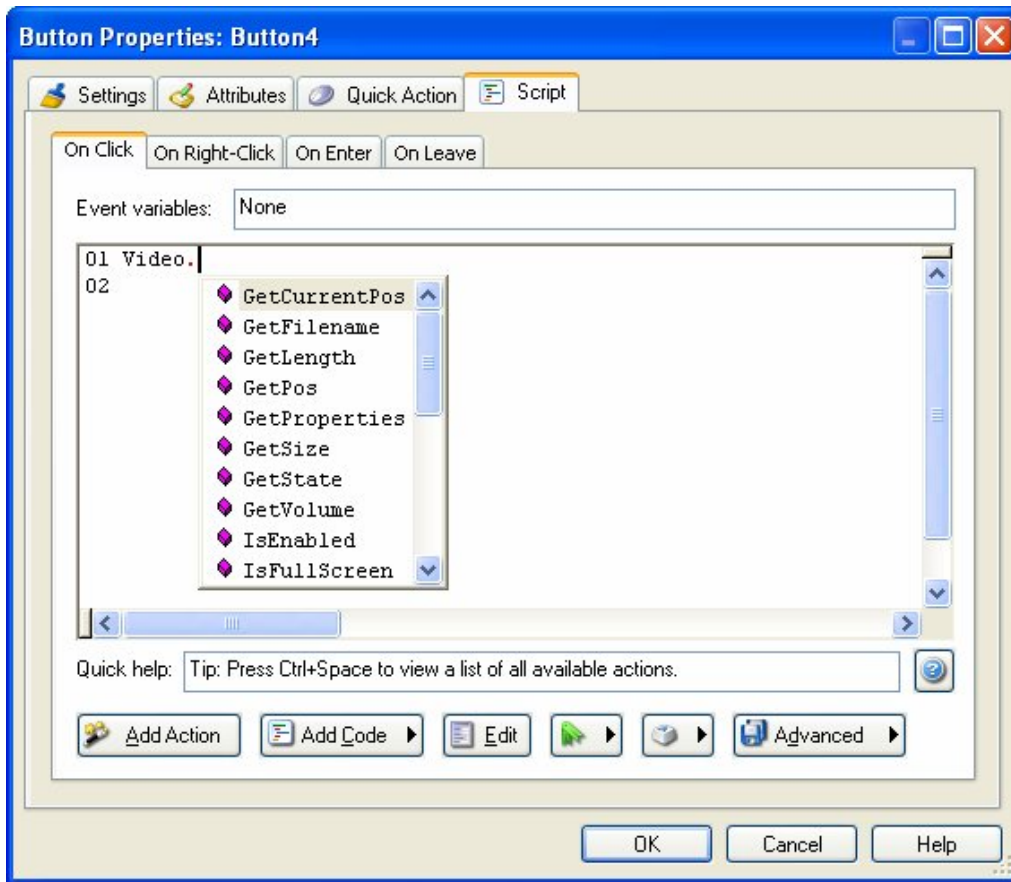
Note: This is what most people generally think of as "programming," but it really isn't that difficult. In fact, as you're about to see, AutoPlay makes programming a lot easier than you might think.

As you type the first few letters of the word "Video" into the script editor, a black tooltip will appear nearby with the word "Video" on it. This is the script editor's *autocomplete* feature at work. Whenever you type something that the script editor recognizes as a keyword, it will display its best guess at what you are typing in one of those little black tooltips.



Whenever one of those little black tooltips is visible, you can press the Tab key to automatically type the rest of the word. For example, you can type *Vi* into the script editor, press Tab, and the script editor will fill in the rest of the word “Video” for you.

There’s more editor magic, too. When you type the period after the word Video, the script editor 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 “Video.” If you like, you can choose one of the actions from that list—in this case, scrolling down to the bottom and choosing the word “Stop”—and then press either Tab or Enter to automatically type that word out for you.



You don't have to use the drop-down list, though...you can continue typing the rest of the action yourself if you want. (For short action names like Video.Stop, it's probably faster to do it that way.)

Tip: The period in an action name is either pronounced "dot," as in "Video-dot-Stop," or it isn't pronounced at all, as in "Video Stop."

Once you've typed something that the script editor recognizes as the name of an action, a little bit of Quick Help appears near the bottom of the window.



This is essentially a “blueprint” for the action, listing the names of the action’s parameters and indicating what type of value is expected for each one. In the case of our Video.Stop action, the Quick Help looks like this:

```
Video.Stop(string ObjectName)
```

...which indicates that the action takes a single parameter called ObjectName, and that this parameter needs to be a string.

Strings need to be quoted, and the name of the video object that we want to stop is Video1, so the full action needs to be typed exactly like this:

```
Video.Stop("Video1");
```

Note: The semi-colon at the end of the line tells AutoPlay where the end of the statement is. It acts as a *terminator*. (No relation to Arnold, though.) Although technically it’s optional—AutoPlay can usually figure out where the end of the statement is on its own—it’s a good idea to get in the habit of including it, to avoid any potential confusion.

Once you’ve typed that text onto the script editor, you can click OK to confirm the changes that you’ve made to the object’s On Click event.

And that’s it! You’ve just added an action, programmer-style.

6) Save the project.

Always save the project! As much fun as it was to add these video controls, there’s no point having to do it all over again if something unexpected happens. (Is that the rumble of thunder I hear off in the distance...?)

7) Choose Page > Preview Page and try out the custom video controls. When you’re done trying it out, exit the preview.

That’s it for this lesson.

Lesson 7 Summary

In this lesson, you learned how to:

- Add a panel image to frame a video
- Add an attractive text banner to the page
- Add a video object
- Customize the video object's built-in control panel
- Make your own video controls from scratch
- Control the video with a few simple actions

Lesson 8:

Audio

Audio is an important part of any multimedia application. Sound effects help make interactive objects like buttons “come alive,” and provide useful cues to the user to help them recognize which objects are interactive. Background music can help set the right mood or enhance the emotional impact of your work. And in many cases, audio is an integral part of the application, whether it’s in the form of music that you’re distributing with an AutoPlay CD, or training materials presented in the form of spoken instructions.

In this lesson, you’ll learn the basics of working with audio files, so you can make your applications sing...or talk...or maybe even inspire your users to get up and dance.

What You'll Learn

In this lesson, you'll learn how to:

- Change the default object sounds for your project
- Set object-specific sound effects
- Add background music
- Pause the background audio while a page is shown
- Load and play an audio file

How Long Will It Take?

This lesson takes approximately 35 minutes to do.

Starting the Lesson

If you're continuing from Lesson 7, you should still have AutoPlay running with the Tutorial project open. If so, you're ready to move on to the next exercise: Changing the Default Object Sounds.

Otherwise, you'll need to open the project file that you saved at the end of Lesson 7.

1) Open the Tutorial.am7 file that you saved in Lesson 7.

When you save a project, AutoPlay automatically creates a project folder for it inside your "My Documents\AutoPlay Media Studio 7.0\Projects" folder. This project folder is where everything that belongs to the project is stored—including the project file, which contains all of the settings used in the project.

The project folder and the project file always have the same name that you gave to the project when it was created. The project file's name ends with a ".am7" file extension.

Since you named this project "Tutorial" in Lesson 1, the name of the project folder will be Tutorial, and the name of the project file will be Tutorial.am7. So, the path to the project file should be something like:

```
...\My Documents\AutoPlay Media Studio 7.0\Projects\Tutorial\Tutorial.am7
```

To open the project, you just need to open that project file.

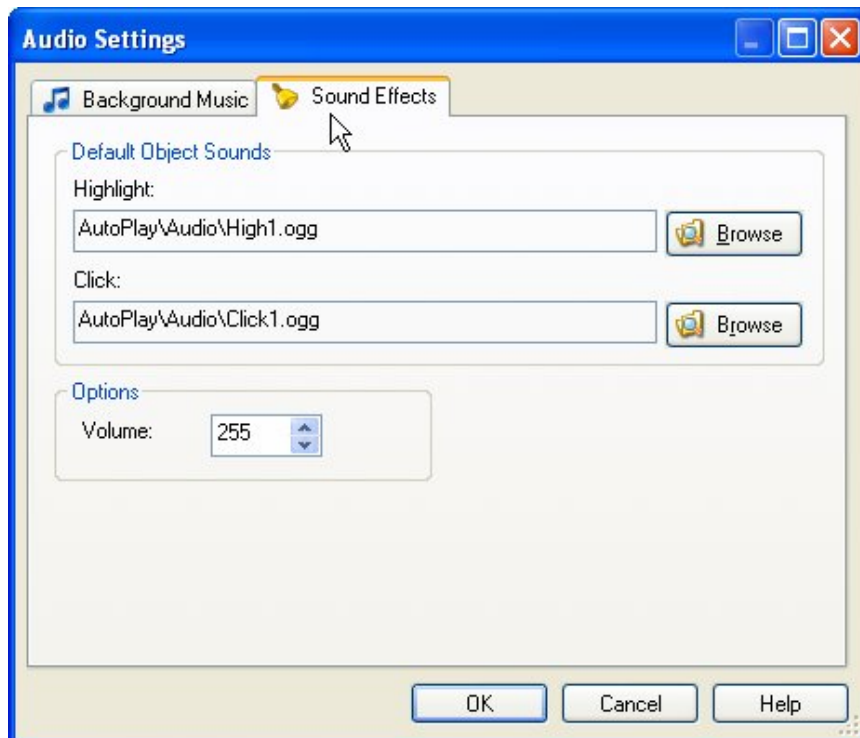
Changing the Default Object Sounds

With the exception of the Flash, RichText, input, listbox, web, progress, combobox, and tree objects, all of the objects in AutoPlay can trigger two kinds of sound effects: one for when the user moves the mouse over them (a “highlight” sound), and another for when the user clicks on them (a “click” sound). For each object’s sound effect, you have the option of playing either no sound at all, a custom sound specific to that object, or a standard sound defined in the project settings.

The standard sound makes it easy to change the sound effects for all of the objects in your project at once, without having to go to each object and change its sound settings individually.

In order to demonstrate how the default object sounds work, let’s temporarily change the standard highlight sound for all objects in the project.

1) Choose Project > Audio. When the Audio Settings dialog opens, click on the Sound Effects tab.



The Sound Effects tab is where you can choose the default highlight and click sound for all objects in the project—or at least, for all of the objects in the project that have their highlight or click sound set to “Standard.” Any changes that you make on this tab will immediately affect the whole project.

2) Click the browse button next to the Highlight setting. When the Select File dialog opens, click the Gallery button, and navigate into the Sound Effects folder.

AutoPlay comes with a number of sample sound effects that you can use in your projects. These sound effects are all in Ogg Vorbis format. Ogg Vorbis is an open-source audio compression format similar to MP3, but capable of greater sound quality and even smaller file sizes. Unlike other audio formats, Ogg Vorbis is patent and license free, making it an excellent choice for the distribution of digital audio.

Note: The MP3 codec is licensed by Fraunhofer IIS-A. If you generate revenue using music in MP3 format, you are responsible for paying them a percentage of each sale.

Ogg Vorbis files end in .ogg.

3) Click on some of the sound files to preview them. When you find one that you like, click OK to select it as the new standard highlight sound.

There’s a “Play audio while browsing” option at the bottom of the Select File dialog that appears when you’re browsing for audio files. It’s turned on by default, so when you click on a sound file, it will automatically begin playing in the background. (You can turn the option off if you prefer to browse in silence.)

Note: Make sure you have your computer speakers turned on. If you don’t have computer speakers, just hum something catchy, close your eyes and pick a file at random.

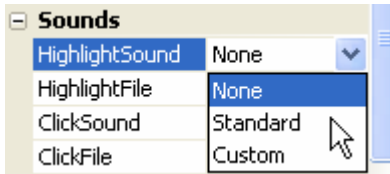
4) Click OK on the Audio Settings dialog to accept the change. Preview the project and move your mouse over the buttons to hear the new highlight sounds. Exit the preview when you’re done.

To preview the project, press F5. When the application opens, move your mouse over the buttons. The sound you picked as the default highlight sound should play when you move the mouse over each of the button objects.

All buttons have their highlight sound set to “Standard” by default. When you changed the standard highlight sound for the project, you changed the highlight sounds for all of the buttons in the project at once.

When you're ready to exit the preview, just click on the Exit button.

5) Switch to Page1 and click on the Email Address label object. In the Sounds category of the properties pane, set HighlightSound to “Standard” instead of “None.”



This will make the Email Address object play the standard highlight sound whenever the mouse moves over it. Since we're using this label as an interactive object, like the buttons on the page, making it play the same highlight sound will help the user identify the label as something they can click on.

Note: The Email Address label object is the one with “ted@sellersrealty.com” on it.

6) Choose Project > Audio, and change the standard highlight sound back to High1.ogg.

To change the highlight sound back to High1.ogg, choose Project > Audio and click on the Sound Effects tab. Click the highlight setting's browse button, and use the Select File dialog to select the High1.ogg file. Click OK on the Select File dialog, then click OK on the Audio Settings dialog, and you're done.

7) Preview the project and move your mouse over the objects on the page to hear the highlight sound. Exit the preview when you're done.

The highlight sounds have all been changed back to the original highlight sound. And since you set the Email Address label object's highlight sound to “Standard,” it now has the same highlight sound as well.

Aren't you glad you didn't have to change the sound for all objects in the project one object at a time? And this tutorial is fairly small...imagine working on a hundred-page project with thousands of buttons in it. This feature may not seem that exciting at first glance, but in the long run it can be a real time saver.

Tip: It's a good idea to use the default object sounds wherever possible, because it makes it much easier to change the sounds throughout your project later.

Setting Object-Specific Sound Effects

You aren't always going to want every object to use the same sound. Some objects were just made to sound different. Luckily, AutoPlay lets you specify a custom sound for each object's sound setting. In fact, if you wanted, every object in your project could have a unique sound.

Let's give the Email Address label object a custom click sound.

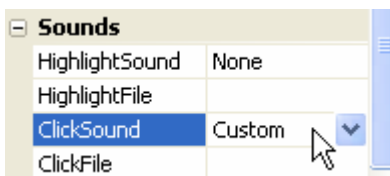
1) Click on the Email Address label object.

The Email Address label object still needs a sound effect for when the user clicks on it. We could use the standard click sound if we wanted, but since clicking on this object does something a bit different, let's make the click sound a bit different as well.

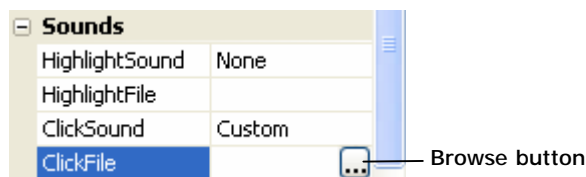
(When the user clicks on this object, the File.OpenEmail action starts a new message in the user's regular email program. Since this happens "outside" of the application—like clicking on a web link that opens a pop-up page in another window—we'll reflect the difference by giving the object a different kind of click sound.)

2) In the Sounds category of the properties pane, change ClickSound from "None" to "Custom."

By default, label objects are configured to make no sound at all. When you set the ClickSound setting to "Custom," the object will use whatever file you specify in the ClickFile setting.



3) Click on the ClickFile setting, and then click the browse button. When the Select File dialog appears, click the Gallery button, and navigate into the Sound Effects folder.



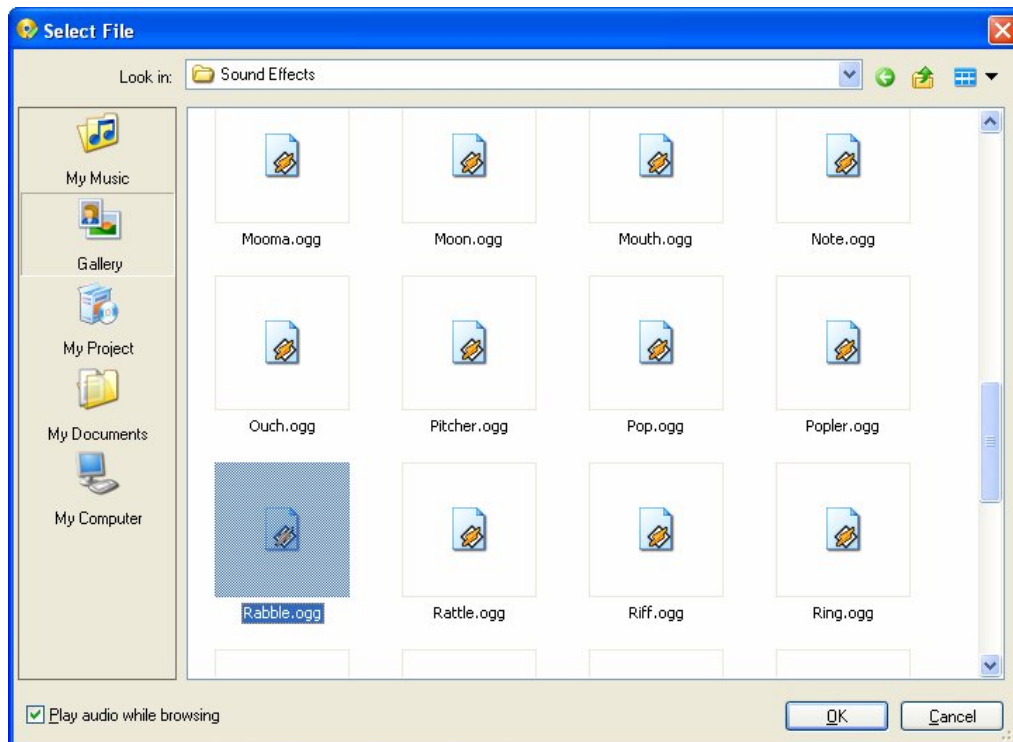
Clicking on the browse button opens the Select File dialog so you can select the audio file that you want to use for the object's click sound.

The Sound Effects folder should be where you were browsing the last time you selected an audio file, so you shouldn't have to do anything to navigate into it. If you followed the past few steps, you're already there.

Note: The Select File dialog always opens up at the same place you navigated to the last time you selected that kind of file.

4) Select the Rabble.ogg file, and click OK.

This will give the Email Address label object a “pop” sound when you click on it.

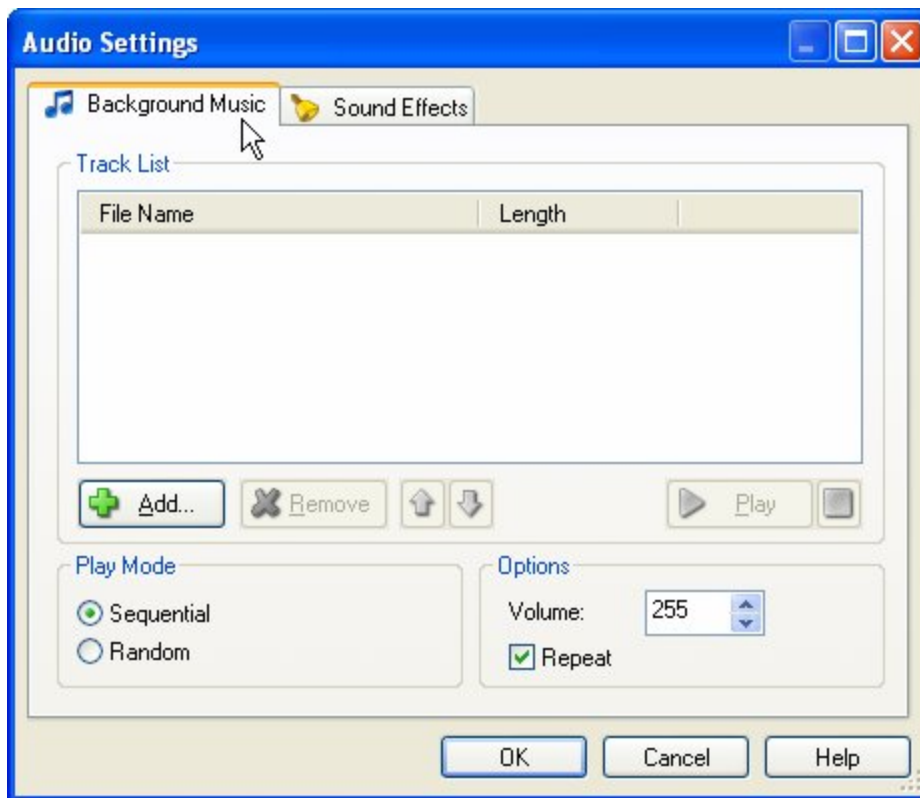


The files in the Sound Effects folder have such cool names

Adding Background Music

AutoPlay makes it really easy to add a list of songs and have them play in the background. Like a soundtrack in a movie, background music can enhance the mood and add atmosphere and emotion to your application.

1) Choose Project > Audio. When the Audio Settings dialog opens, click on the Background Music tab.



The Background Music tab is where you can put together a list of songs that will be played in sequence while the user is browsing your application.

The songs can be played back sequentially, in the order that they are listed, or you can choose to have them play back at random.

You can also specify the volume at which the background audio will play, and whether the whole list should repeat—which is to say, whether the background music should start over from the beginning when the last track has been played.

2) Click the Add button. When the Select File dialog appears, click the Gallery button, and navigate into the “Music” folder.

The first time you click the Add button on the Background Music tab, the Select File dialog automatically takes you to the Music folder. (From that point on, it remembers what folder you were in the last time you selected a background music audio file.)

3) Select the file named “Acoustic Folk (short).ogg” and click OK.

As you add songs to the list, they are automatically copied into your project’s Audio folder, so that they will be included when you publish the project to a folder or burn it to a CD. This song will end up in the project folder as “CD_ROOT\AutoPlay\Audio\Acoustic Folk (short).ogg.”

4) Add some more songs to the list.

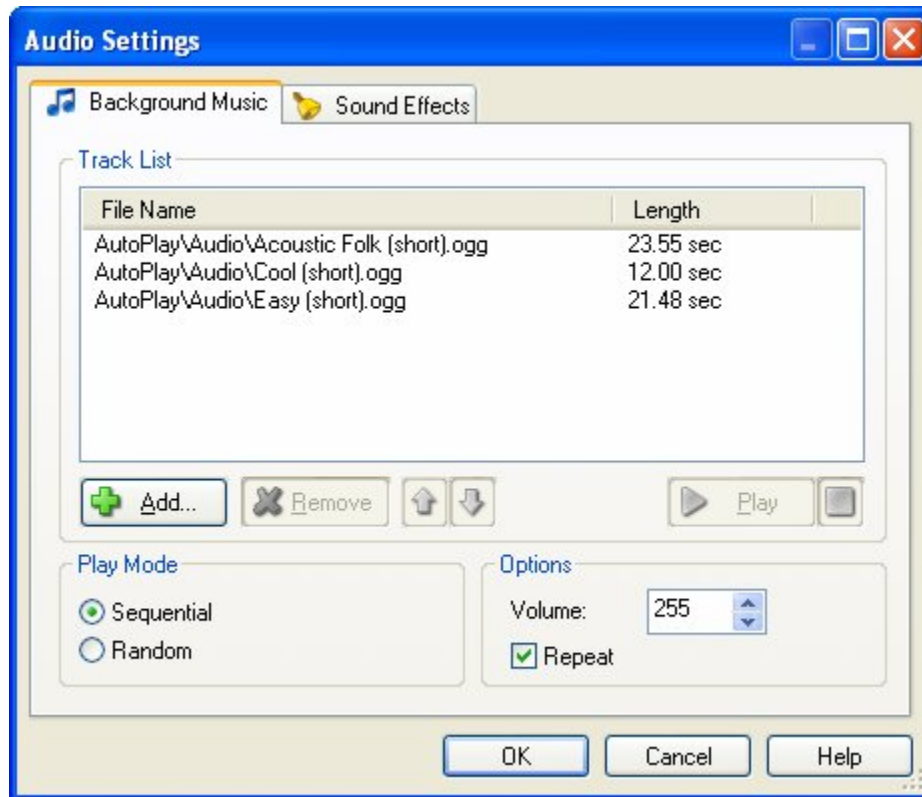
You can add all of the sample songs if you want, or just pick and choose the ones you like.

Feel free to add some favorite songs to the mix from your own collection, too.

Note: The songs will be played back in the same order that they appear in on the list.

Once a song has been added, you can change its position or remove it by using the buttons beneath the list.

Tip: You can also rearrange the songs by dragging them up or down with the mouse.



Three sample tunes from the gallery

5) Press F5 to preview the project. Try jumping back and forth between the pages.

The background music starts up as soon as the application starts, and keeps playing until it closes. Note that jumping to another page doesn't interrupt the music at all...it just keeps on playing, smooth as always.

Even jumping to the video page doesn't stop the background music. When the video starts playing, its audio plays over top of the background music. The video has its own soundtrack, but it just gets mixed together with the background music of the application.

It sounds pretty bad, because the video's audio track and the background music were definitely not meant to be played at the same time. But we can fix that by pausing the background music while the user is on the video page.

6) Exit the preview.

Just click the Exit button or press Alt+F5 to close the application and return to the AutoPlay design environment.

Pausing the Background Audio

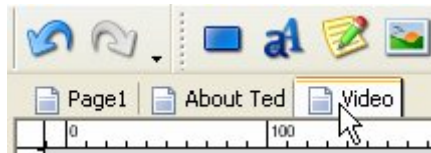
There are times when you'll want the background music to stop playing. For instance, if you have a video object that has its own soundtrack, you probably don't want it to clash with the background music, so you'll want to pause the background music while the video is being shown.

In our case, the video is shown while the user is on the Video page, so we need to turn the music off when they reach that page, and turn it back on when they leave it. This is easy to do with a pair of Audio.TogglePlay actions.

Tip: You can use this same technique to add a toggle button so the user can turn the music on or off—just in case their taste in music is a bit *too* different from yours.

1) Switch to the Video page.

Before you can add any actions to the Video page's events, you need to switch to the page. To do that, just click on the page's tab.



2) Add an Audio.TogglePlay action to the page's On Show event. In the action's parameters, set Channel to CHANNEL_BACKGROUND. Leave the script editor open.

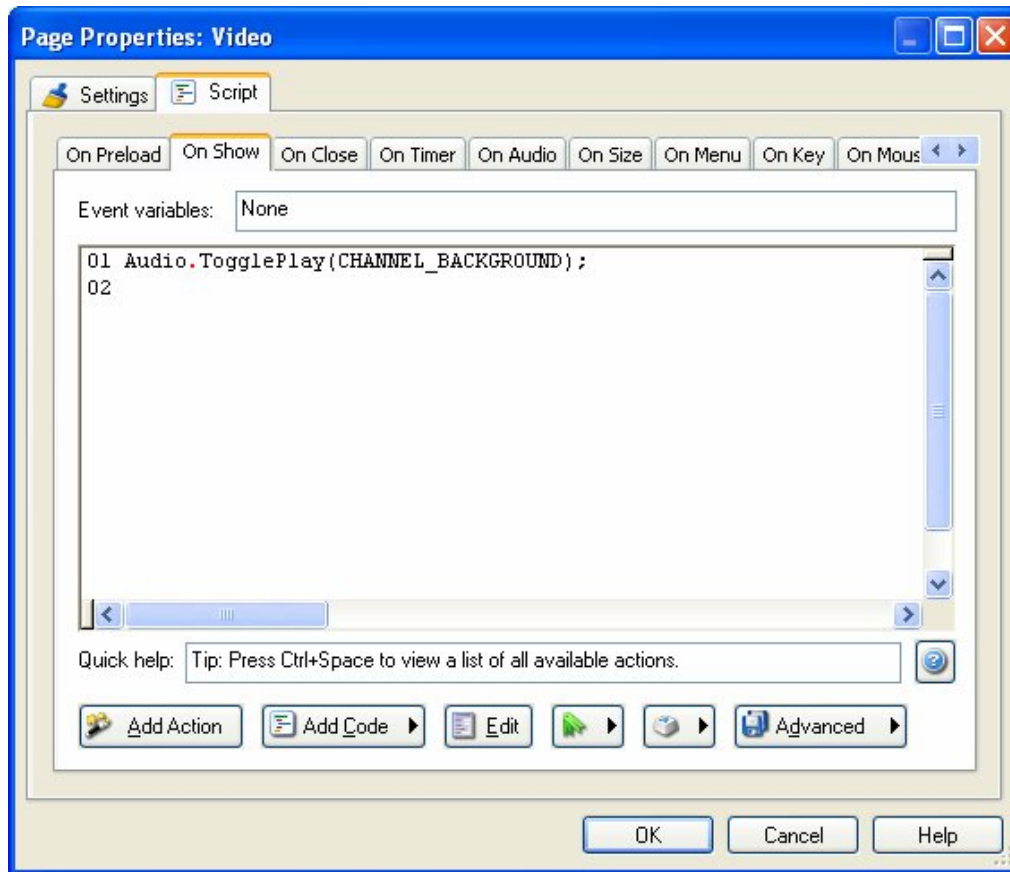
The Audio.TogglePlay action toggles an audio channel's playback between playing and paused. If there is currently audio playing in that channel, the action will pause it; if the audio in that channel is currently paused, the action will un-pause it.

To add the action, double-click on the page surface, and click on the Script tab. In the script editor, click on the On Show tab, and then click the Add Action button to bring up the New Action wizard. Change the category to "Audio," select Audio.TogglePlay from the list, and click Next to edit the action's parameters.



In the action parameters, you can specify the channel that you want the action to toggle. In this case, we want to toggle the playback of the background audio channel, so you need to change the Channel setting to CHANNEL_BACKGROUND.

Once you have the action configured, click Finish to close the wizard. The Audio.TogglePlay action will appear in the script editor.



Note: Don't close the script editor just yet; you need to add an action to the On Close event as well.

The On Show event is triggered whenever the page opens. The background music will still be playing when the user leaves Page1, so this action should pause the music when the user arrives at the Video page.

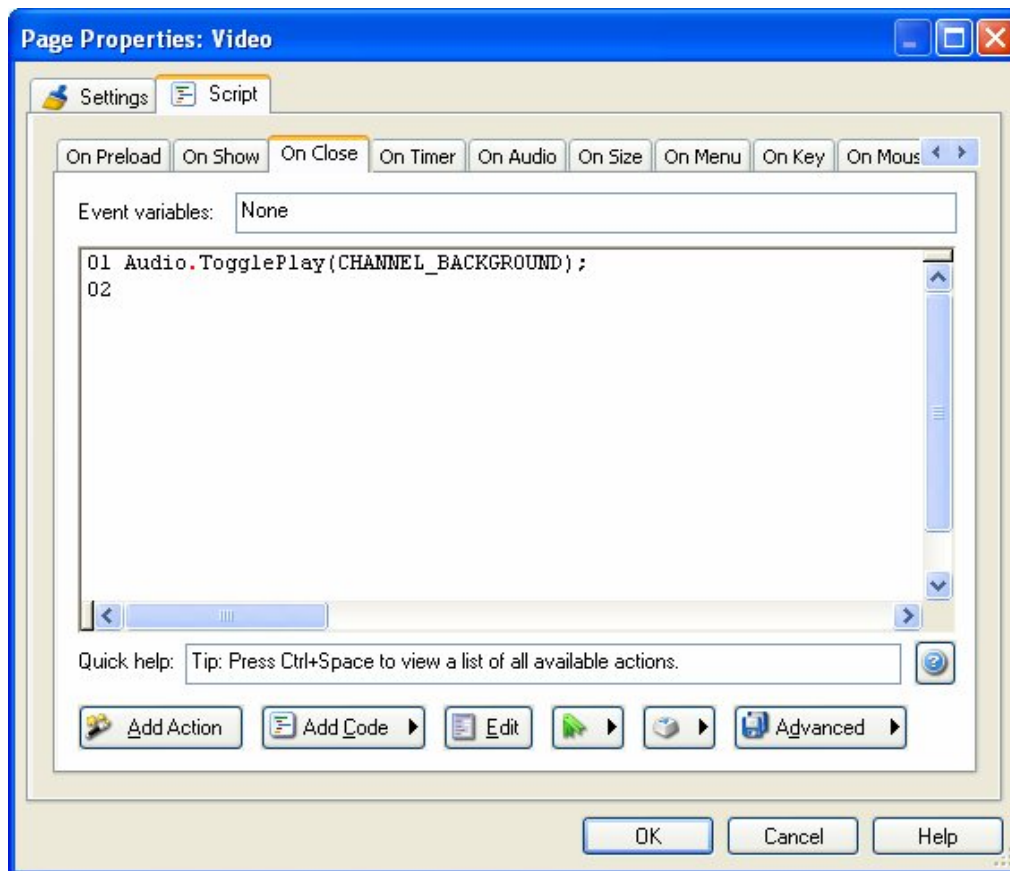
3) Copy the Audio.TogglePlay action to the page's On Close event.

We want to add another Audio.TogglePlay action to the On Close event for this page, to toggle the background music back on when the user jumps back to the main page. It needs to be just like the action you added to the On Show event, because we want the two actions to balance each other out—one to turn the audio off, and one to turn it back on again.

You *could* click on the On Close tab and then add another action the same way you did in step 2. But since we want this action to have the same parameters as the first one, you might as well save yourself a bit of time and just copy the text from one tab to the other.

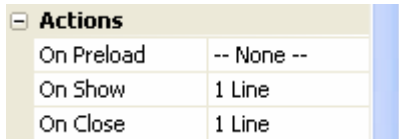
To do so, just select the whole line of text and press Ctrl+C to copy it into the clipboard. Then, click on the On Close tab, and press Ctrl+V to paste the text onto it.

Note that this won't affect the action that you added to the On Show tab at all—the actions you add to one tab stay the same when you switch to another one. You're just making a change to both events at the same time.



Once you've configured the action for the On Close event, click OK to close the script editor and confirm your changes.

In the properties pane, both events should now show “1 Line” next to them.



| Actions | |
|------------|------------|
| On Preload | -- None -- |
| On Show | 1 Line |
| On Close | 1 Line |

4) Preview the project, and click on the “Video Presentation” button to jump to the Video page.

Now the background music stops when you arrive at the Video page, and the video’s audio plays all by itself.

This is because the `Audio.TogglePlay` action is pausing the background channel on the page’s On Show event.

5) Click on the Back button to return to the main page.

When you click on the Back button, the background music starts playing again, right where it left off. This time it’s the `Audio.TogglePlay` action on the page’s On Close event that is toggling the background audio channel back on.

6) Exit the preview.

That’s it for this preview, so click on the Exit button (or press `Alt+F4`) to exit the application and return to the `AutoPlay` design environment.

Loading and Playing an Audio File

Now that you know how to pause the background music, let’s tackle playing an audio file in response to an event. For example, you might want to start playing an audio file when the user clicks on a button, or when the user jumps to a specific page.

For this exercise, we’ll play a different audio file while the user is on the About Ted page.

1) Switch to the About Ted page. Double-click on the page surface, and click on the Script tab. In the script editor, click on the On Show tab, and add an `Audio.Pause` action to pause the background audio channel.

Before we start playing a different song, we need to pause the background music. Although we could use an `Audio.TogglePlay` action to do it, like we did on the Video page, let’s use an `Audio.Pause` action this time.

To add the action, just click on the Add Action button, select the Audio.Pause action from the list, and click Next to get to the action's parameters. Set the Channel parameter to CHANNEL_BACKGROUND, and click Finish.

That will pause the background music. Now we need an action to play a different audio file.

2) Click the Add Action button to add a second action to the On Show tab. When the New Action wizard appears, choose the "Audio" category. Select the Audio.Play action from the list, and click Next.



If you were looking through the list of audio actions, and you wanted to play an audio file, you might head straight for Audio.Play. But when you clicked Next to customize the action, you wouldn't see any way to specify an audio file. All you would find is a single parameter letting you select the audio channel that you want to play.



This is because the Audio.Play action is used to start playing a file that has already been loaded into an audio channel.

In fact, the Audio.Play action assumes that you've already loaded an audio file into a channel using an Audio.Load action.

Luckily, the Audio.Load action gives you a way to do both at the same time.

3) Click Back to go back to the first page of the New Action wizard. Select the Audio.Load action instead, and click Next. Leave the Channel parameter set to CHANNEL_NARRATION, and set the PlayAutomatic and Loop parameters both to *true*.

The Audio.Load action certainly has more parameters to play with.



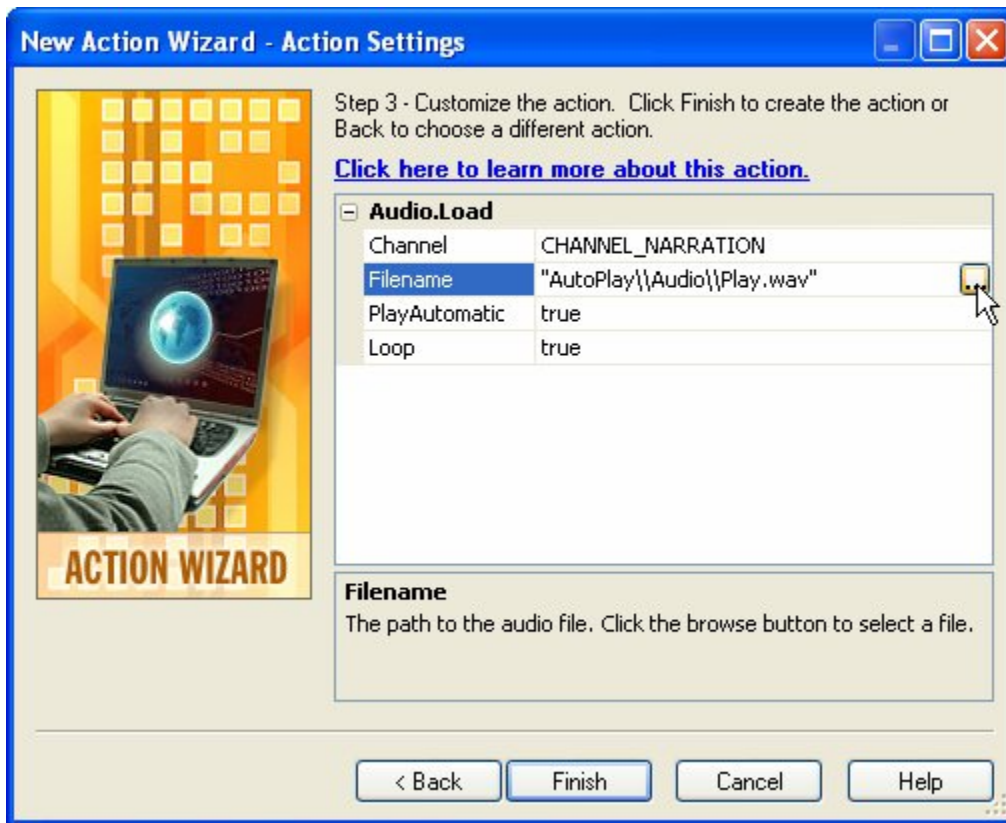
We want the audio file to play in a different channel than the background music, so leave the Channel set to CHANNEL_NARRATION. The name implies that this channel is for “spoken” audio, but that’s only one use for it. You’re free to use the channel for other forms of audio, which is exactly what we’re going to do.

Note: We don’t want to use the background audio channel for this, because it’s already occupied by a song from the Background Music tab...and we only want to replace that music “locally” on the About Ted page—not everywhere. Since we’re only after a temporary change, we need to leave the background audio channel alone.

Setting PlayAutomatic to true tells the action to begin playing the file automatically as soon as it’s loaded. Since we don’t know how long the user will remain on this page, we want the file to keep playing in the background until the page closes. This means that we need the file to loop when it reaches the end, so make sure Loop is set to true as well.

Note: Don't close the wizard yet—you still need to tell the action which audio file to load.

4) Click on the Filename setting, and then click the browse button. When the Select File dialog appears, click the Gallery button, and select a song from the Music folder.



When you click on the browse button, the Select File dialog appears so you can select an audio file. This is the file that the action will load.

Tip: If you'd rather use your own music, click one of the other buttons and look for an audio file somewhere on your system.

Once you select an audio file, AutoPlay will copy it to your project's Audio folder and put the appropriate path into the Filename parameter for you. It also automatically escapes all of the backslashes in the path.



Escaped Backslash

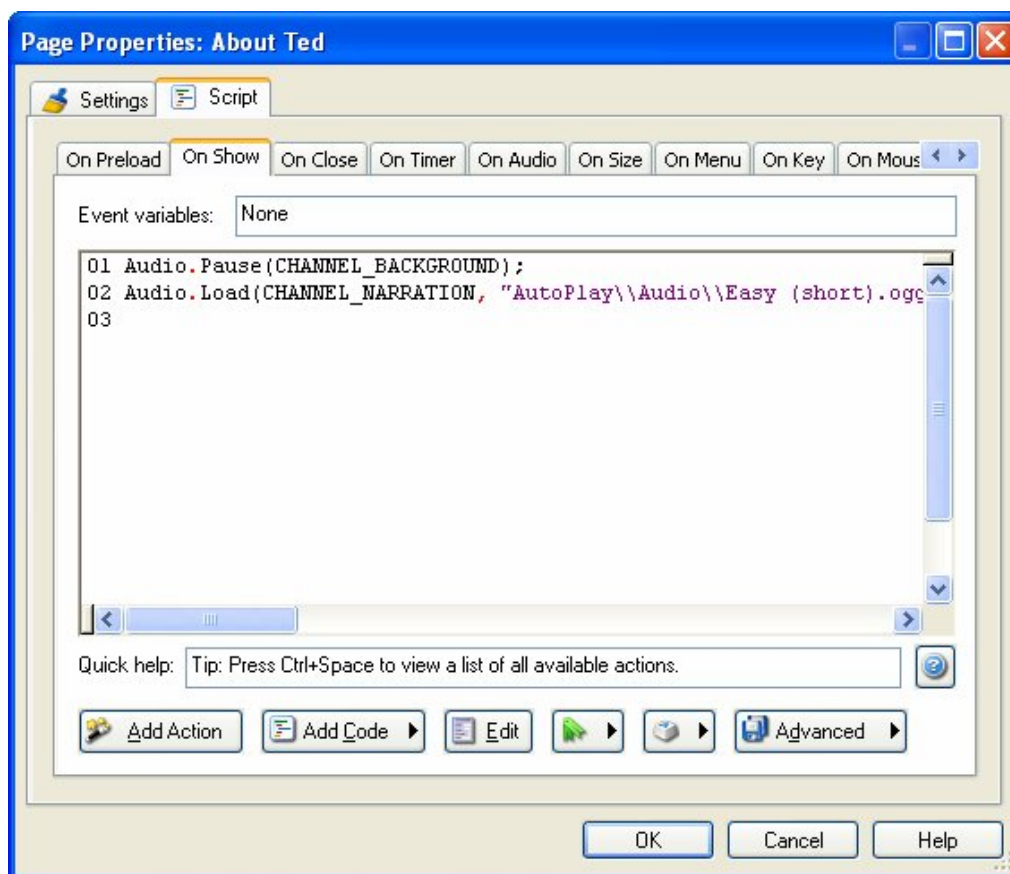
Inside a string like the Filename parameter, a backslash character is used to indicate an escape sequence. For instance, in strings a newline is represented by the sequence `\n`. Those two characters are the escape sequence that represents the invisible character you type by pressing the Enter key.

Given that a backslash indicates the beginning of an escape sequence, we need some way to represent a normal, actual backslash in a string. As you can see, this is done by using a pair of backslashes. In fact, `\\` is an escape sequence that represents a single (normal) backslash.

This may seem a little bit complicated, but don't worry...you can always use the browse button to select a file, and AutoPlay will handle all the double-backslashes for you. But if you ever want to edit a string directly, just remember that you need two backslashes (\\) to represent a regular backslash in a string.

5) Click Finish to close the New Action wizard.

When you close the wizard, the second action will appear on the script editor, just below the first one. The end result should look like this:



{Of course, the filename may be different if you chose a different file.)

6) Switch to the On Close tab, and on the first two lines, type:

```
Audio.Stop(CHANNEL_NARRATION);  
Audio.Play(CHANNEL_BACKGROUND);
```

Remember to press Enter at the end of each line.

This will add two actions to the On Close event: an Audio.Stop action, to stop the song that is playing in the “narration” channel; and an Audio.Play action, to start playing the background music again.

Note: You could also add these actions by clicking the Add Action button and using the New Action wizard. If you feel more comfortable doing it that way, you can...there’s nothing wrong with using the wizard. One way isn’t any “better” than the other. In fact, I tend to use both methods myself...typing in the actions that I’ve become familiar with, and using the wizard to add actions that I’m not so sure about.

We need the Audio.Stop action to stop playing the song that we loaded with the Audio.Load action on the On Show event. We need the Audio.Play action to start playing the “normal” background music that we paused with the Audio.Pause action on the On Show event. The whole progression will go something like this:

Step 1: *Page opens*

Step 2: *Audio.Pause -- pauses the background music*

Step 3: *Audio.Load -- loads our page-specific song*

Step 4: *Page closes*

Step 5: *Audio.Stop -- stops our page-specific song*

Step 6: *Audio.Play -- unpauses the background music*

7) Click OK to confirm your changes.

Clicking OK on the script editor will finish adding the four actions to the two page events, confirming all of the changes that you’ve made. The properties pane will show that there are “2 Lines” of action script for both the On Show and On Close events.

You now have two actions on the On Show event (Audio.Pause and Audio.Load), and two actions on the On Close event (Audio.Stop and Audio.Play).

8) Preview the project, and go to the About Ted page.

As soon as you jump to the About Ted page, the background music will pause, and the audio file that you selected in step 3 will begin playing in its place. This is the result of the two actions on the page's On Show event, firing one after the other in sequence.

9) Click on the Back button to return to the main page.

As you leave the About Ted page, the audio file that you selected in step 3 will stop playing, and the original background music will continue where it left off. This time, it's the two actions you added to the On Close event at work.

10) Exit the preview and save the project.

When you're done previewing the project, exit the application and return to the AutoPlay design environment.

That's it for this lesson! Be sure to save your changes before moving on.

Lesson 8 Summary

In this lesson, you learned how to:

- Change the default object sounds for your project
- Set object-specific sound effects
- Add background music
- Pause the background audio while a page is shown
- Load and play an audio file



Lesson 9:

Publishing

This lesson will walk you through the final stage of AutoPlay project development: building and publishing your AutoPlay application.

What You'll Learn

In this lesson, you'll learn how to:

- Build to a folder on your hard drive
- Build a compressed executable
- Burn a CD-R, CD-RW, DVD±R or DVD±RW

How Long Will It Take?

This lesson takes approximately 10 minutes to do.

Starting the Lesson

If you're continuing from Lesson 8, you should still have AutoPlay running with the Tutorial project open. If so, you're ready to move on to the next exercise: Building to a Folder.

Otherwise, you'll need to open the project file that you saved at the end of Lesson 8.

1) Open the Tutorial.am7 file that you saved in Lesson 8.

When you save a project, AutoPlay automatically creates a project folder for it inside your "My Documents\AutoPlay Media Studio 7.0\Projects" folder. This project folder is where everything that belongs to the project is stored—including the project file, which contains all of the settings used in the project.

The project folder and the project file always have the same name that you gave to the project when it was created. The project file's name ends with a ".am7" file extension.

Since you named this project "Tutorial" in Lesson 1, the name of the project folder will be Tutorial, and the name of the project file will be Tutorial.am7. So, the path to the project file should be something like:

```
...\My Documents\AutoPlay Media Studio 7.0\Projects\Tutorial\Tutorial.am7
```

To open the project, you just need to open that project file.

Building to a Folder

The fastest publishing method is to build the project to a folder on your hard drive.

When you publish your project to a folder, AutoPlay uses the settings in the project file to generate an executable for your application. It then copies this executable—along with the entire contents of your project’s CD_ROOT folder—to the output folder of your choice.

The only difference between building to a folder, and burning a CD, is *where* your application ends up. Building to a folder is essentially like preparing the contents of a CD in a folder before burning it.

1) Choose Publish > Build to open the Publish wizard.



The first page of the Publish wizard lets you choose how you want to publish your AutoPlay application. The option you select on this page determines (a) what options

are available when you click Next, and (b) what the end result of the build process will be.

2) Select the Hard Drive Folder option, and click Next.

You're going to publish the project to a folder on your hard drive, so select the Hard Drive Folder option.



Clicking Next will bring you to the second page of the wizard where you can specify the output folder.

Tip: Even if you're planning to burn your project to a CD, it's a good idea to build it to a folder at least once first. That way, you can test your application without wasting a CD-R, or having to wait for a slow CD-RW burn. Once you've made sure that everything works the way you intended, you can either re-publish the project using a different method, or burn the contents of the output folder using an external CD burning program.

3) In the Output Folder field, type C:\AutoPlay Output.

The output folder is simply the location where you want the application to be built. You can use any folder for this purpose, but it's best to create a new folder, since any existing files will be deleted before the application is built.

Warning! Any files in the output folder will be deleted during the build process! Be very careful not to use a folder that contains any files or subfolders you want to keep.

You can also select an output folder by clicking the browse button.



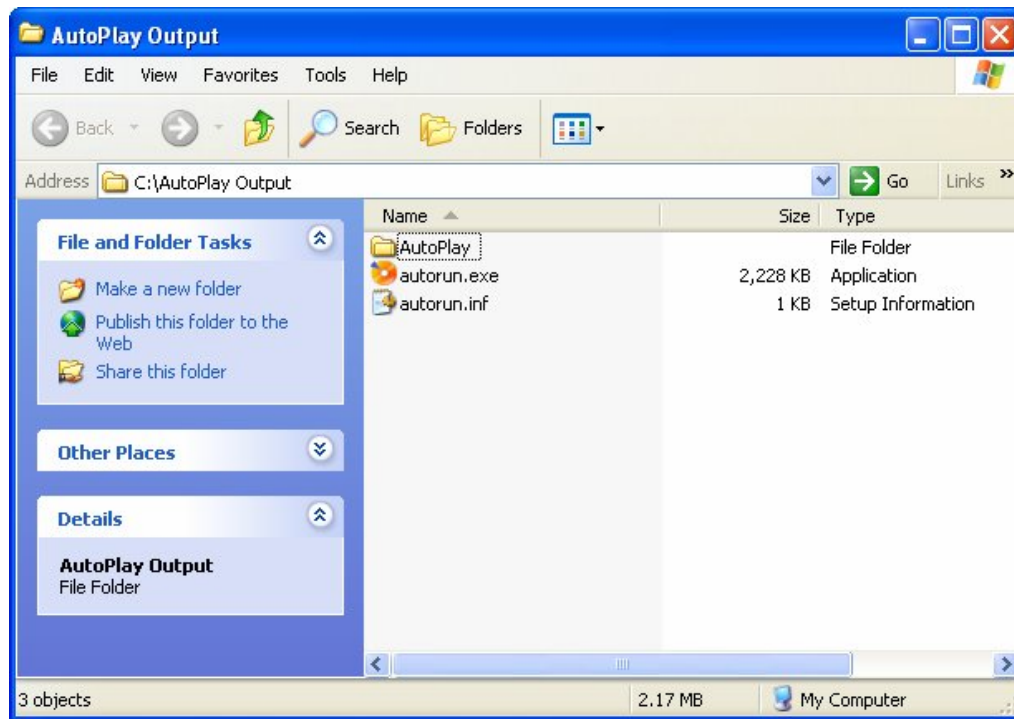
Tip: The output folder you choose doesn't *have* to be on a local hard drive; you could easily direct the output to a folder on a shared network drive, or even to some other form of storage, such as a USB memory key. The only requirement is that there is enough room on the device for your AutoPlay application.

4) Click Build and wait for the build process to end.

AutoPlay will build the project and inform you when your application is ready. Since the project is fairly small, and the build process is very fast, you shouldn't have to wait too long. (On a fast system it shouldn't take more than five or six seconds.)

5) Click Close to exit the Publish wizard. When the output folder opens, double-click on the autorun.exe file to launch your application.

When you close the Publish wizard, AutoPlay will automatically open the output folder for you.



The contents of this folder are exactly what you would see on the root of the CD or DVD if you had burned your project to a disc instead of building it to a folder.

In fact, everything here is built and ready to go. You could select everything in the output folder, drag it into your CD burning software, and burn it onto a CD. Or you could use a tool like Setup Factory to package it up into an installer, so it can be installed on the user's hard drive like any other application. What you do with the project at this point is entirely up to you!

Of course, before you distribute your project, you should always test it thoroughly. To launch it, just double-click on the Autorun.exe file.

6) Exit from the application and return to AutoPlay Media Studio.

Once you've had a chance to try your application out, close the application and return to the AutoPlay design environment.

Now let's try building the project to a compressed executable.

Building a Compressed Executable

For smaller projects, building a compressed executable is a great way to make the application portable. The entire project will be compressed into a single executable file, with all of the project's contents inside it. When you double-click on this file to launch it, the contents will be extracted to a temporary location on the local hard drive, and the application will run from there. The whole process is very seamless and automatic.

You can even choose to encrypt the contents, to prevent anyone from accessing them without launching the application.

Although you can technically build a compressed executable for a project of any size, it's better to reserve this method for smaller projects. The larger the project is, the longer it takes for the data to be extracted when the user launches your application. (The contents must be extracted each time the application is started.)

Tip: If the application is going to be used repeatedly, or is relatively large, you may want to use a professional development tool like Setup Factory to create a traditional installer for it. That way, the contents can be unpacked once (at install time) and then accessed like any other program, reducing launch times significantly.

1) Press F7 to open the Publish wizard.

Pressing F7 opens the Publish wizard, just like choosing Publish > Build from the program menu.

2) Select the Web/Email Executable option, and click Next.

This time, you're going to publish the project as a single executable file.



Clicking Next will bring you to the second page of the wizard where you can specify a path and filename for the executable file.

3) In the Filename field, type *C:\Temp\TedSellers.exe*. Leave the other settings at their default values.

The Filename field lets you specify the full path and filename of the executable file that AutoPlay will build for you. By default, it will be set to build the application in your My Documents\AutoPlay Media Studio 7.0\Output folder, with the same name as the project file (but ending with .exe).

To change this setting, you can either type a new path into the field, or you can select a folder and filename by clicking the browse button.



Setting a path and filename for the compressed executable

The Progress Window settings on this dialog let you configure the small popup “Loading” message that appears while the application is extracting the project data onto the local hard drive. For example, you could change the message from “Loading” to “Please wait...” or make it so it doesn’t appear at all.

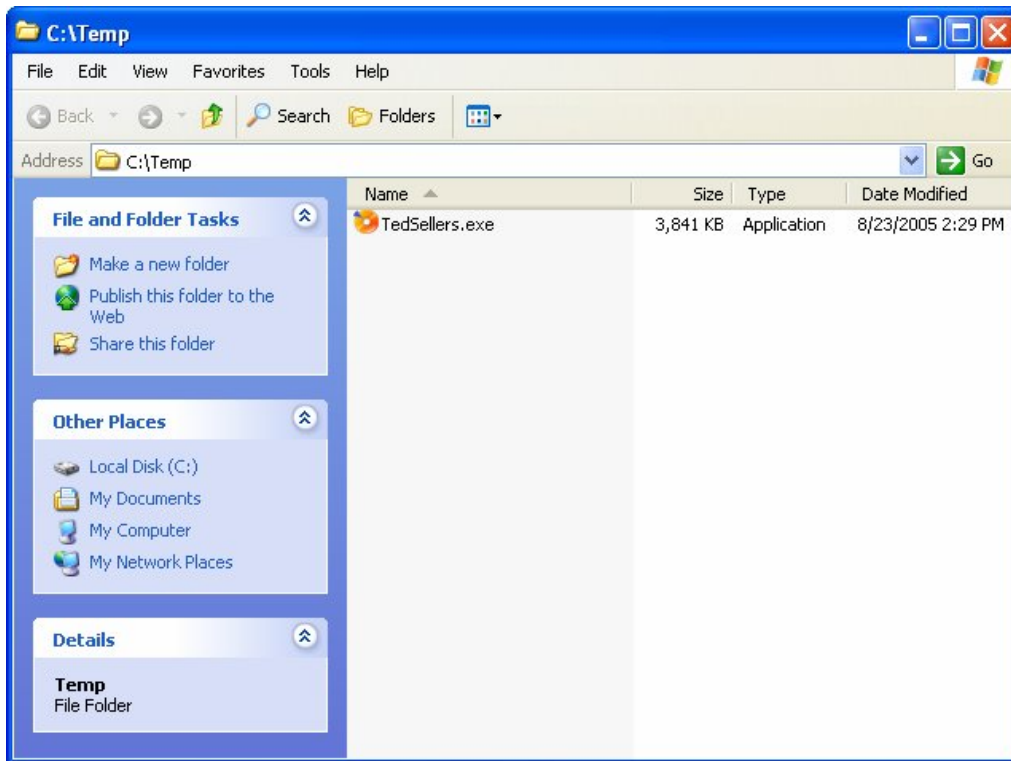
The “Encrypt data segment” option tells AutoPlay to encrypt all of the project data that it stores in the compressed executable. This adds a small layer of security to the application, so that users can’t extract the project files directly from the executable using an external zip utility like WinZip.

4) Click Build and wait for the build process to end.

AutoPlay will build the project and inform you when your application is ready. This build process is almost as fast as the hard drive folder option, so you shouldn’t have to wait too long.

5) Click Close to exit the Publish wizard. When the C:\Temp folder opens, double-click on the TedSellers.exe file to launch your application.

When you close the Publish wizard, AutoPlay will automatically open the C:\Temp folder for you. Inside that folder, you should see an executable file called TedSellers.exe.



The entire application is contained within that file. When you double-click on it, the “Loading” message will appear as the contents are automatically extracted to a temporary folder on the hard drive, and then the application will run the same as before.

Note: When you build a compressed executable, there is only one file to distribute.

6) Exit from the application and return to AutoPlay Media Studio.

Once you've had a chance to try your application out, close the application and return to the AutoPlay design environment.

Note: A little bit of magic happens behind the scenes after you click the Exit button. As your application exits, it removes the files that it extracted to the temporary folder when you launched it. You don't have to worry about the files being left "exposed" in the user's temp folder, or about taking up space on the user's hard drive. Like any well-behaved program, your AutoPlay application cleans up after itself.

Burning a CD or DVD

With its built-in burning engine, AutoPlay Media Studio makes publishing your project to a CD-R, CD-RW, DVD±R or DVD±RW a piece of cake.

For this exercise, I'll describe the steps for burning a compact disc, so you don't have to keep reading "CD-R, CD-RW, DVD±R or DVD±RW" over and over. But the same principles apply for burning to DVD±R or DVD±RW.

Note: You will need a CD-R or CD-RW drive and a blank CD-R or CD-RW disc in order to complete this exercise.



1) Insert a blank recordable CD or CD-RW into your CD writer, and press F7 to open the Publish wizard.

Pressing F7 opens the Publish wizard, just like choosing Publish > Build from the program menu.

Note: We recommend using a CD-RW disc so you can reuse it afterwards.

2) Select the “Burn data CD/DVD” option, and click Next.

This time, you’re going to burn the project to a recordable CD.

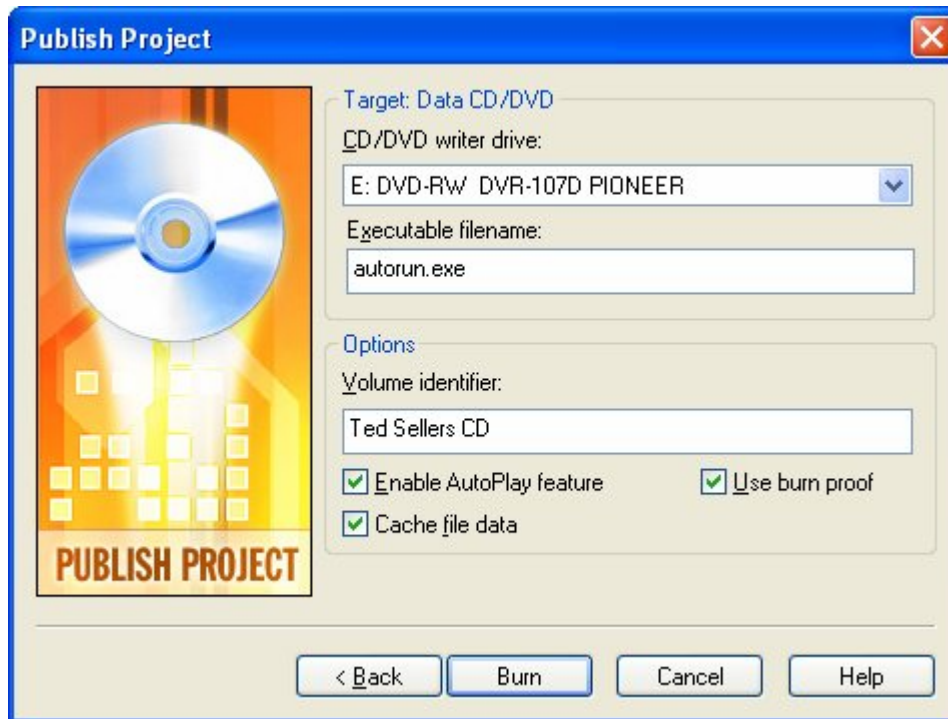


Clicking Next will bring you to the second page of the wizard where you can specify the burn options.

3) Select your CD writer drive from the drop-down list, and change the volume identifier to something appropriate, like *Ted Sellers CD*.

The “CD/DVD writer drive” drop-down list is provided in case you have multiple CD or DVD drives in your system. Select the drive that you will be using to burn the CD.

The “Volume identifier” field lets you specify the name of the CD as it will appear in the My Computer folder in Windows.



4) Click Burn and wait for the burn process to end.

If you're using a CD-RW disc, a warning dialog will appear to ask if you're sure you want to do so. You aren't burning a "master" disc to send to a duplication service, so just click Yes.

If the CD-RW isn't empty, another dialog will appear asking if you want to erase the disc. If you're sure that there isn't anything on this disc that you wanted to keep, then click OK and AutoPlay will erase the disc for you before burning the project. (If you put the wrong disc in by mistake, just eject it and replace it with a blank one.)

Once the burn is completed, the drive should automatically eject the CD.

5) Click Close to exit the Publish wizard. Close the CD tray to reinsert the disc back in the drive.

After you insert the CD in the drive, you should hear it spin up as the drive begins to read the contents. The drive will look in the root folder of the CD for a file named autorun.inf which tells it the name of a program to load when the CD is inserted.

Note: AutoPlay automatically creates the autorun.inf file for you when the “Enable AutoPlay feature” option is turned on.

Your AutoPlay application should start up automatically. (If it doesn't, then the autorun feature is disabled on your computer. You will need to either re-enable it, or open the CD in My Computer and then double-click on the autorun.exe file yourself.)

6) Exit from the application and return to AutoPlay Media Studio.

Once you've had a chance to try your application out, close the application and return to the AutoPlay design environment.

That's it! You've just published your AutoPlay project to a CD.

You now know everything you need to know to build and publish a project just like this one.

Tip: In the next lesson, we'll start a new project so you can try out some of AutoPlay's scripting abilities.

Lesson 9 Summary

In this lesson, you learned how to:

- Build to a folder on your hard drive
- Build a compressed executable
- Burn a CD-R, CD-RW, DVD±R or DVD±RW



Lesson 10:

Scripting Basics

This lesson will teach you some of the basics of scripting in AutoPlay. Although you can accomplish a lot in AutoPlay without any knowledge of scripting at all, even a little bit of scripting practice can make a big difference. You can accomplish far more in a project with a little bit of scripting than you ever could without it. Scripting opens the door to all sorts of advanced techniques, from actions that are only performed when specific conditions are met, to functions that you can define, name and then call from anywhere else.

10

What You'll Learn

In this lesson, you'll learn how to:

- Display a message
- Use a variable
- Add an if statement
- Test numeric values
- Set a button object's text
- Concatenate strings
- Compare strings
- Use a for loop
- Create functions

How Long Will It Take?

This lesson takes approximately 30 minutes to do.

Starting the Lesson

For this lesson, we're going to create a new project from scratch. If you're continuing from Lesson 9, you should still have AutoPlay running with the Tutorial project open.

If not, you'll need to open AutoPlay Media Studio first.

1) If you have AutoPlay running with the Tutorial project open, choose File > New.

Choosing File > New will open the New Project Options dialog.

Once it opens, move on to step 3.

2) Or, if you don't have AutoPlay running, use the Start menu to launch the AutoPlay Media Studio program. When the Welcome dialog appears, click on "Create a new project."

You'll find AutoPlay Media Studio under:

Start > Programs > Indigo Rose Corporation > AutoPlay Media Studio 7.0

Once the program launches and the Welcome dialog appears, click on the "Create a new project" option. This will open the New Project Options dialog.

3) Change the Project Name to Lesson 10.

You want to replace the default text with a unique name for this project, so highlight all of the text in the Project Name field and type *Lesson 10*.

This name will be used for the project folder and the project file.

4) Select the Blank Project and click Create Project Now.

When you click Create Project Now, the New Project Options dialog closes, AutoPlay sets up the Lesson 10 project folder and project file, and the new blank project is loaded into the design environment.

You're now ready to learn some basic scripting.

Displaying a Message

Okay, first things first. Before we get to any of the fancy stuff, let's make a script that does something really simple, like displaying a message to the user.

First, though, we need a place to put our script. For convenience, let's use a button object's On Click event.

1) Add a button object to the page.

One quick way to add a button object is to right-click on the page and choose Button from the right-click menu. Then just select the button you want to use from the list of button files on the Select File dialog, and click OK.

Note: It doesn't matter what button you choose, so pick whichever one you like.

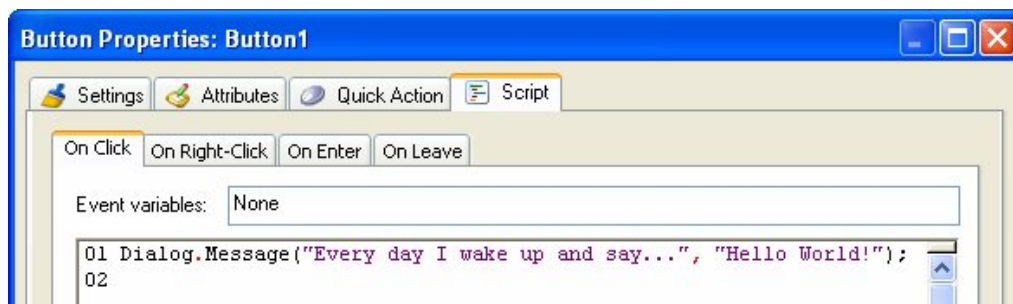
2) Double-click on the button object, click on the Script tab, and click on the On Click event tab.

You want to add an action to the object's On Click event, so open the object's Properties dialog, switch to the Script tab, and make sure you're editing the script for the On Click event.

3) On the first line, type:

Dialog.Message("Every day I wake up and say...", "Hello World!");

This script consists of a single Dialog.Message action which passes "Every day I wake up and say..." as the string to display in the title bar, and "Hello World!" as the string to display on the dialog. The script should look like this when you're done:



Note: Although this script only has a single action, scripts can be much longer. In fact, there's no limit to how long scripts can be.

4) Click OK to close the Properties dialog. Press F5 to preview the project. When the preview application appears, click on the button to see the message.

Clicking on the button triggers its On Click event and performs any script that has been added to it. In this case, the script consists of a single call to the Dialog.Message action, which displays the following message:



5) Click OK to close the dialog message box, and exit from the preview.

Click OK on the message to close it, and then exit from the preview and return to the design environment.

Tip: You can exit from the preview by pressing Alt-F4.

Using a Variable

One of the most powerful features of scripting is the ability to make use of variables. Variables are essentially just “nicknames” or “placeholders” for values that may need to be modified or re-used in the future. Each variable is given a name that you can use to access its current value in your script.

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 place a value into a variable by assigning the value to it with an equals sign. For example, the following script assigns the value 10 to a variable called “amount.”

```
amount = 10;
```

You can change this value at any time by assigning a different value to the variable. (The new value simply replaces the old one.) For example, the following script assigns 45 to the amount variable, replacing the number 10:

```
amount = 45;
```


...and the following script assigns the string “Woohoo!” to the variable, replacing the number 45:

```
amount = "Woohoo!";
```

Note that you can easily replace a numeric value with a string in a variable. Having a number or a string in a variable doesn't “lock” it into only accepting that type of value—variables don't care what kind of data they hold.

This ability to hold changeable information is what makes variables so useful.

1) Double-click on the button object. In the On Click script, replace the "Hello World!" string with a variable named strMsg.

Just edit the script on the button object's On Click event so that it looks like this instead:

```
Dialog.Message("Every day I wake up and say...", strMsg);
```

This will make the Dialog.Message action display the current value of the strMsg variable when it is performed. Before we try it out, though, we need to assign a value to that variable somewhere.

Note: A common practice among programmers is to give their variables names with prefixes that help them remember what the variables are supposed to contain. One such prefix is “str,” which is used to indicate a variable that contains a string. Other common prefixes are “n” for a numeric value (e.g. nCount, nTotal) and “b” for a Boolean true/false value (e.g. bLoaded, bReadyToStart).

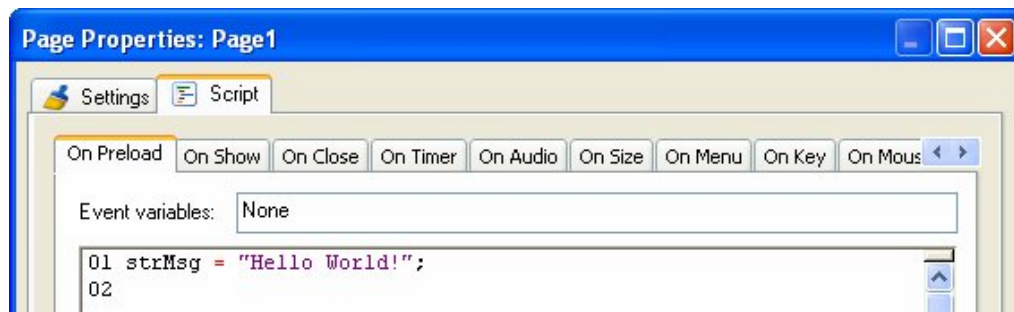
2) Click OK to close the Properties dialog.

Once you've modified the On Click script, click OK to accept your changes and close the Properties dialog.

3) Double-click on the page and click on the Script tab. On the first line of the On Preload script, type:

strMsg = "Hello World!";

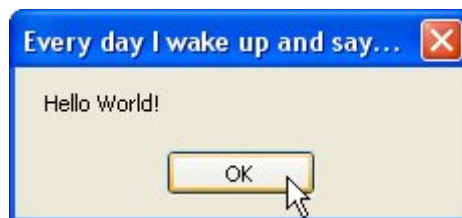
The script should look like this when you're done:



This will assign the string “Hello World!” to the variable named strMsg during the page’s On Preload event. The On Preload event is triggered as soon as the page has been created in memory, just before it actually appears on the screen. This makes it a good place to put any initialization script, such as setting up default values or preparing variables that will be used once the page is shown.

4) Press F5 to preview the project. When the preview opens, click on the button to see the message.

The Dialog.Message action displays the message box, just like before.



Note that the variable’s name, strMsg, is nowhere to be found...instead, the value that is currently in the variable is displayed. In this case, it’s the value that was assigned to the variable in the page’s On Preload event.

5) Exit the preview. In the page's On Preload script, change the value that is assigned to the strMsg variable to "Good Morning!".

Double-click on the page, and edit the script in the On Preload event so it looks like this instead:

```
strMsg = "Good Morning!";
```

6) Press F5 to preview the project. When the preview opens, click on the button to see the message.

This time, the message looks like this:



As you can see, you've just changed the message without even touching the Dialog.Message action.

Now imagine if you had such Dialog.Message actions on fifty pages throughout your project, and you decided you wanted to change the text. With variables and a little planning, such changes are a piece of cake.

Adding an If Statement

The if statement gives your scripts the ability to make decisions, and do one thing or another in different circumstances.

Each if statement consists of a *condition* (or “test”), followed by a *block* of script that will only be performed if the condition is found to be true.

The basic syntax is:

```
if condition then  
    do something here  
end
```

For example:

```
if age < 18 then
    Dialog.Message("Sorry!", "You must be 18 to access this CD.");
    Application.Exit();
end
```

The above script checks to see if the value in a variable named “age” is less than 18. If it is, it puts up a message saying “You must be 18 to access this CD,” and then immediately exits from the application.

For example, if we set age to 17 first:

```
age = 17;
if age < 18 then
    Dialog.Message("Sorry!", "You must be 18 to access this CD.");
    Application.Exit();
end
```

...the block of script between the “then” and “end” keywords will be performed, because 17 is less than 18. In this case, we say that the if statement’s condition “passed.”

However, if we set age to 20:

```
age = 20;
if age < 18 then
    Dialog.Message("Sorry!", "You must be 18 to access this CD.");
    Application.Exit();
end
```

...the block of script between the “then” and the “end” isn’t performed, because 20 isn’t less than 18. This time, we say that the if statement’s condition “failed.”

Note: An if statement's condition can be any expression that evaluates to true or false.

Let's modify the script on our button object's On Click event to only display the message if it is “Hello world!”.

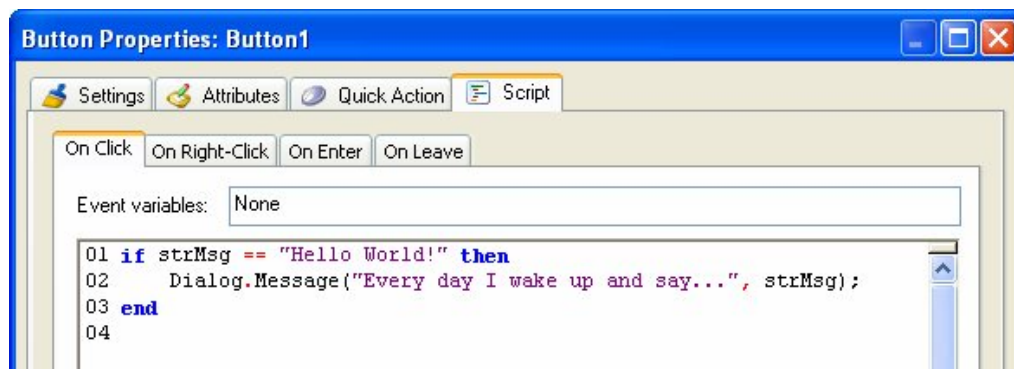
1) Double-click on the button object. Edit the On Click script to add an if statement, like this:

```
if strMsg == "Hello World!" then  
    Dialog.Message("Every day I wake up and say...", strMsg);  
end
```

Use a tab character to indent the original “Dialog.Message...” line to the right a little. (It’s good practice to indent any lines inside an if statement in order to help make the logic easier to follow.) To insert a tab character at the start of the line, place the cursor to the left of the “D” and press the Tab key.

Tip: You can indent one or more lines at once by highlighting them and pressing Tab to increase the indent (to the right), or Shift+Tab to decrease the indent (to the left).

The script should look like this when it’s done:



The double equals compares for equality, so this if statement’s condition will only be true if strMsg contains “Hello World!” with the exact same capitalization and spelling.

Tip: An easy way to add an if statement on the script editor is to highlight the line of text that you want to put “inside” the if, click the Add Code button, and then choose “if statement” from the menu. An if statement “template” will be added around the line that you highlighted. You can then edit the template to fit your needs.

2) Press F5 to preview the project. When the preview opens, click on the button to trigger the script.

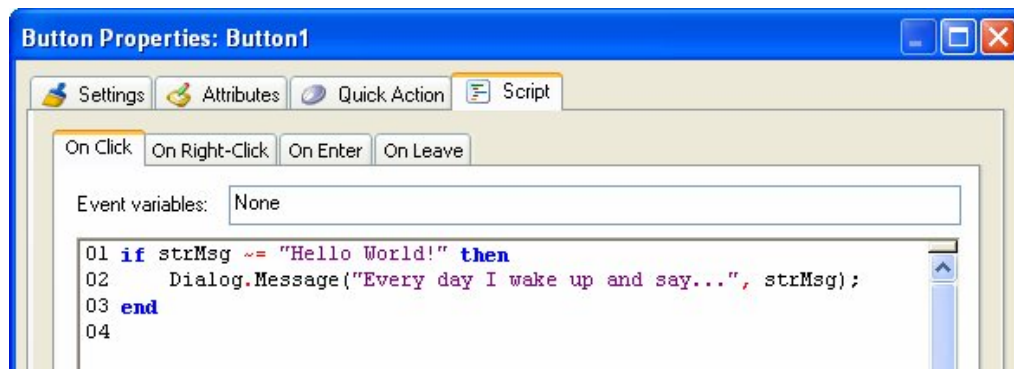
This time, nothing happens, because strMsg is still set to "Good Morning!" in the On Preload event. “Good Morning!” doesn’t equal “Hello World!” so the if condition

fails, and the block of code between the “then” and “end” keywords is skipped entirely.

3) Exit from the preview. Edit the button object’s On Click script to change the == to ~=, like this:

```
if strMsg ~= "Hello World!" then  
    Dialog.Message("Every day I wake up and say...", strMsg);  
end
```

The script should look like this when it’s done:



The tilde equals (~=) compares for inequality, so this if statement’s condition will only be true if strMsg contains anything *but* “Hello World!” with the exact same capitalization and spelling.

4) Preview the project. When the preview opens, click on the button.

This time, because strMsg contains “Good Morning!”, which is definitely not equal to “Hello World!”, the message will appear.

The == and ~= operators are fine when you want to check strings for an exact match. But what if you’re not sure of the capitalization? What if the variable contains a string that the user typed in, and you don’t care if they capitalized everything correctly?

One solution is to use an old programmer’s trick: just convert the contents of the unknown string to all lowercase (or all uppercase), and do the same to the string you want to match.

Let's modify our script to ask the user for a message, and then display what they typed, but only if they typed the words "hello world" followed by an exclamation mark.

5) Exit from the preview. Edit the button object's On Click script to look like this:

```
strMsg = Dialog.Input("", "Enter your message:");  
if String.Upper(strMsg) == "HELLO WORLD!" then  
    Dialog.Message("Every day I wake up and say...", strMsg);  
else  
    Dialog.Message("Um...", "You didn't type Hello World!");  
end
```

The first line uses a Dialog.Input action to pop up a message dialog with an input field that the user can type into. Whatever the user types is then assigned to the strMsg variable.

Note: This new value replaces the value that was assigned to strMsg in the page's On Preload event.

In the if statement's condition, we use a String.Upper action to convert the contents of strMsg to all uppercase characters, and then compare that to "HELLO WORLD!".

We've added an "else" keyword after the first Dialog.Message action. It basically divides the if statement into two parts: the "then" part, that only happens if the condition is true; and the "else" part, that only happens if the condition is false. In this case, the else part uses a Dialog.Message action to tell the user that they didn't type the right message.

6) Preview the project. Try clicking on the button and typing different messages into the input field.

Depending on what you type, you'll either see the "hello world!" message, or "You didn't type Hello World!".

Note that if you type some variation of "hello world!", such as "hello world!", or "hELLO World!", the capitalization that you type will be preserved in the message that appears. Even though we used a String.Upper action to convert the message string to all uppercase letters in the if statement's condition, the actual contents of the variable remain unchanged.

When the `String.Upper` action converts a value to all uppercase letters, it doesn't change the value in the variable...it just retrieves it, converts it, and passes it along.

7) Exit the preview.

Now you know how to compare two strings without being picky about capitalization. Programmers call this a *case-insensitive* comparison, whereas a comparison that only matches perfect capitalization is considered *case-sensitive*.

Tip: You can also use a `String.CompareNoCase` action to perform a case-insensitive comparison.

Testing a Numeric Value

Another common use of the `if` statement is to test a numeric value to see if it has reached a certain amount. To demonstrate this, let's create another button that displays a message after you click on it 5 times.

1) Add another button object to the page.

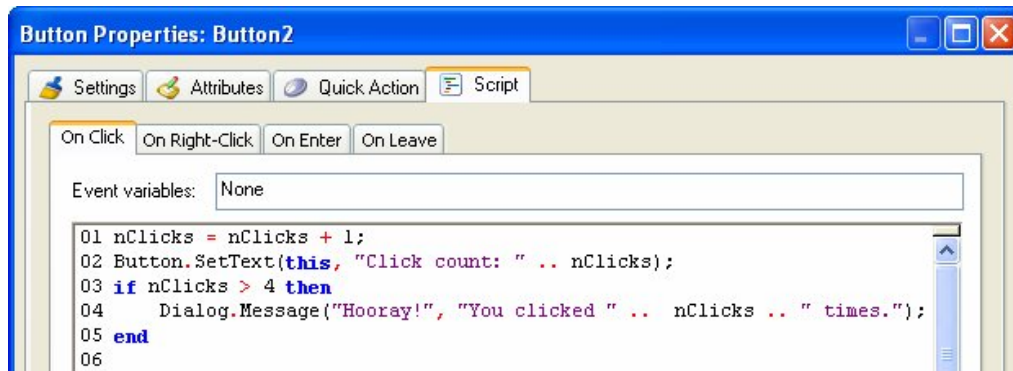
It doesn't matter what button you choose, or where you put the button on the page. We just need something to click on, so we can trigger an `On Click` event and perform some actions.

2) Add the following script to the button object's `On Click` event:

```
nClicks = nClicks + 1;  
Button.SetText(this, "Click count: " .. nClicks);  
if nClicks > 4 then  
    Dialog.Message("Hooray!", "You clicked " .. nClicks .. " times.");  
end
```

Double-click on the new button you added, then click on the `Script` tab and type the above script into the `On Click` event.

It should look like this when you're done:



The first line adds 1 to the current value contained in a variable named `nClicks`. (We'll add some script to the page's On Preload event in the next step, to set this variable to 0 when the page loads.)

The second line uses a `Button.SetText` action to change the button object's text to "Click count: *n*", where *n* is the current value of `nClicks`.

The `Button.SetText` action takes two parameters. The first parameter is a string containing the name of the object that you want it to operate on. In this case, we used the special variable `this`, which contains the name of the object that the On Click event was triggered on. This is essentially the same as using the name of the object, like so:

```
Button.SetText("Button2", "Click count: " .. nClicks);
```

The `Button.SetText` action's second parameter is a string containing the text that you want to display in the button object. We used the concatenation operator to join the current value of `nClicks` to the end of the string "Click count: ". Note that this string has a space at the end of it, so there will be a space between the colon and the value of `nClicks`. (The resulting string just looks better that way.)

The concatenation operator consists of two periods, or dots (`..`). In fact, it's often referred to as the "dot-dot" operator. It is used to join two strings together to form a new, combined string. It's kind of like string glue.

Note: Technically, the value inside `nClicks` isn't a string—it's a number. That doesn't really matter, though. When you're doing concatenation, AutoPlay will automatically convert a number into the equivalent string, as required.

The rest of the script is just an if statement that tests whether the value of `nClicks` is greater than 4, and displays a message when that's true. (A couple more concatenation

operators are used in the Dialog.Message action, to create the string for its second parameter. Note that you can easily include the contents of a variable in the “middle” of a string as well.)

Once you’re done editing the button’s On Click script, click OK to close the script editor.

3) Add the following line to the page's On Preload event:

nClicks = 0;

We need to initialize nClicks before the button is clicked, so that our script in the button’s On Click event will start counting up from 0. Note that we can’t just set nClicks to 0 in the button’s On Click event, or it would be reset to zero every time the button was clicked.

To add the line to the page’s On Preload event, just double-click on the page surface, then click on the On Preload tab of the Script editor, and add the nClicks line to the existing script. Once you’re done, it should look something like this:



It doesn’t matter if you put the nClicks = 0; line before or after the existing line...the order of the individual lines in this script isn’t important.

4) Preview the project, and click on the button object 5 times.

Each time you click on the button, its text will change to show how many times you’ve clicked on it so far. After you’ve clicked on the button 5 times, it will also display a message telling you how many times you’ve clicked.

5) Exit the preview.

There you go. You've just created a pretty sophisticated little program.

Tip: If you feel adventurous, see if you can modify the script to stop announcing the number of clicks once you've clicked 10 times.

Here's a hint: you'll need to use a logical operator like "or" and "and" to test more than one thing at once in the if statement's condition. For example, you could make the test only be true between 3 and 7 by using a condition like this:

```
if (nCount > 2) and (nCount < 8) then
    -- only true if greater than 2 and less than 8
end
```

The second line in the above script is a *comment*. (It doesn't actually do anything.) Everything on a line after two dashes (--) is completely ignored by AutoPlay. You can use such comments wherever you'd like to add "internal" notes to your scripts.

Using a For Loop

Sometimes it's helpful to be able to repeat a bunch of actions several times, without having to type them over and over and over again. One way to accomplish this is by using a *for* loop.

The basic syntax of the for loop is:

```
for variable = start, end, step do
    do something here
end
```

For example:

```
for n = 10, 100, 10 do
    Dialog.Message(" ", n);
end
```

The above script simply counts from 10 to 100, going up by 10 at a time, displaying the current value of the variable *n* in a dialog message box. This accomplishes the same thing as typing:

```
Dialog.Message(" ", 10);
Dialog.Message(" ", 20);
Dialog.Message(" ", 30);
```

```
Dialog.Message(" ", 40);  
Dialog.Message(" ", 50);  
Dialog.Message(" ", 60);  
Dialog.Message(" ", 70);  
Dialog.Message(" ", 80);  
Dialog.Message(" ", 90);  
Dialog.Message(" ", 100);
```

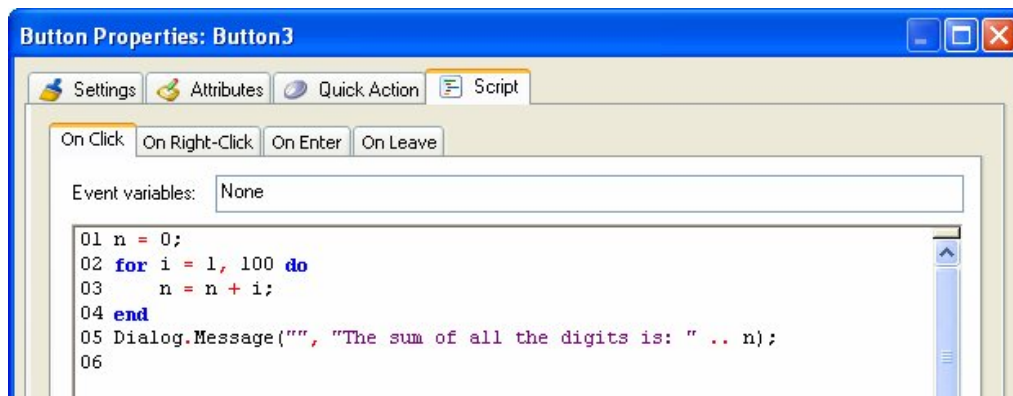
Obviously, the for loop makes repeating similar actions much easier.

Let's use a simple for loop to add all of the digits between 1 and 100, and display the result.

1) Add a button object to the page. Add the following script to this button object's On Click event:

```
n = 0;  
for i = 1, 100 do  
    n = n + i;  
end  
Dialog.Message("", "The sum of all the digits is: " .. n);
```

The script should look like this when you're done:



The first line creates a variable called *n* and sets it to 0. The next line tells the for loop to count from 1 to 100, storing the current “count” at each step in a variable named *i*.

During each “pass” through the loop, the script between the “do” and the “end” will be performed. In this case, it consists of a single line that adds the current values of *n*

and *i* together, and then stores the result back into *n*. In other words, it adds *i* to the current value of *n*.

This for loop is the same as typing out:

```
n = n + 1;  
n = n + 2;  
n = n + 3;  
n = n + 4;
```

...all the way up to 100.

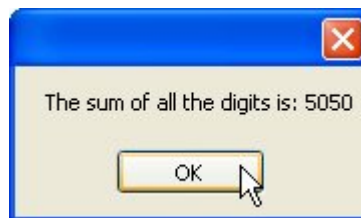
The last line of our script just displays the result of the calculation to the user in a dialog message box.

Tip: You can use the Add Code button to insert an example for loop, complete with comments explaining the syntax. You can then edit the example to fit your own needs.

2) Preview the project, and click on the button object.

When you click on the button object, the for loop will blaze through the calculation 100 times, and then the `Dialog.Message` action will display the result.

It all happens *very* fast.



3) Exit the preview.

You probably won't find yourself using for loops as often as you'll use if statements, but it's definitely worth knowing how to use them. When you need to repeat steps, they can save you a lot of effort.

Of course, when it comes to saving you effort, the real champions are functions.

Creating Functions

A function is just a portion of script that you can define, name and then call from somewhere else.

You can use functions to avoid repeating the same script in different places, essentially creating your own custom “actions”—complete with parameters and return values if you want.

In general, functions are defined like this:

```
function function_name (arguments)  
    function script here  
    return return_value;  
end
```

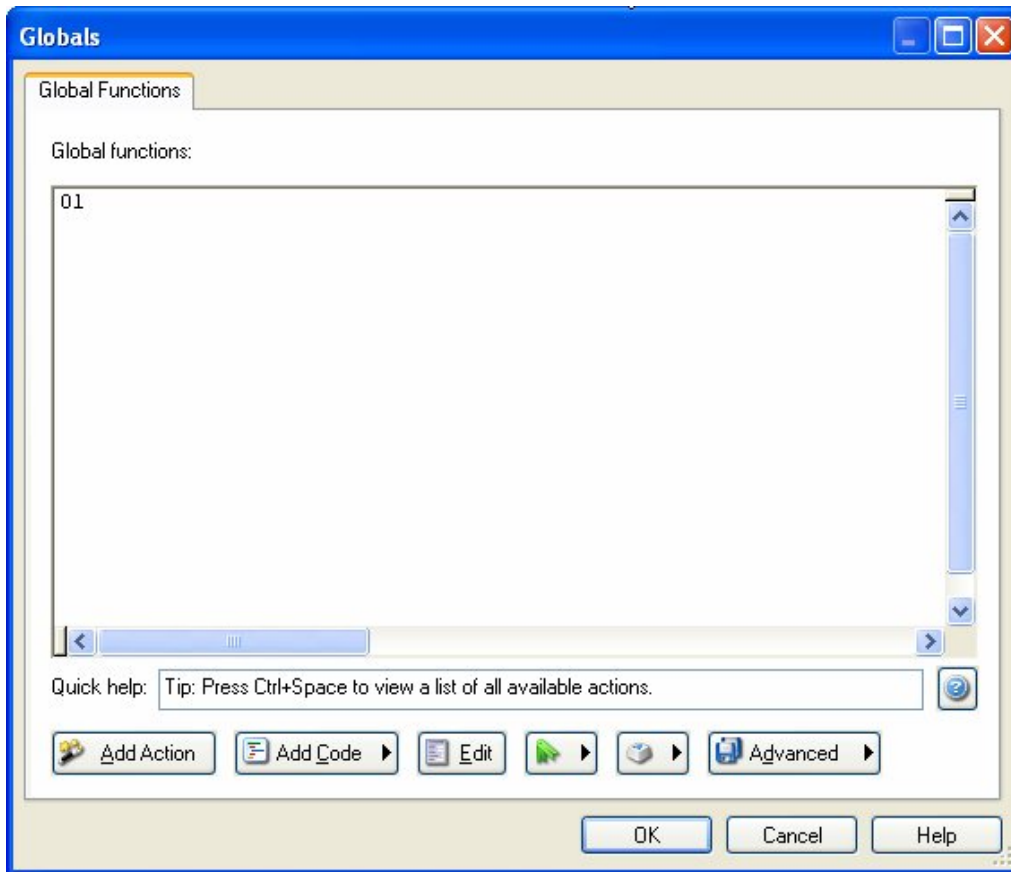
The “function” keyword tells AutoPlay that what follows is a function definition. The *function_name* is simply a unique name for the function. The *arguments* part is the list of parameters that can be passed to the function when it is called. It’s essentially a list of variable names that will receive the values that are passed. (The resulting variables are local to the function, and only have meaning within it.) A function can as many arguments as you want (even none at all).

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 try some examples, so let’s dive right in.

1) Choose Project > Global Functions.

This opens the Global Functions dialog.



The Global Functions dialog is a convenient place to put any functions or variables that you want to make available throughout your project. Any script that you add on this dialog will be performed when your application is launched, right before the project's On Startup event is triggered.

2) Add the following script:

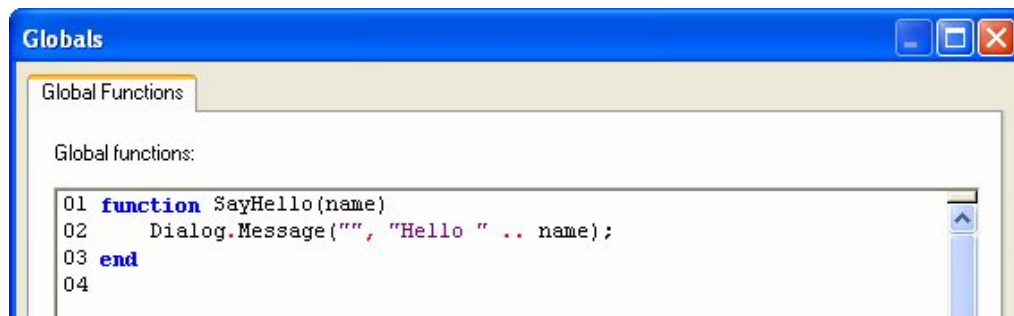
```
function SayHello(name)  
    Dialog.Message("", "Hello " .. name);  
end
```

When you're done, click OK to close the dialog.

This script defines a function named SayHello that takes a single argument (which we've called "name") and displays a simple message.

Note that this only *defines* the function. When this script is performed, it will "load" the function into memory, but it won't actually display the message until the function is called.

It should look like this when you're done:



Once you've entered the function definition, click OK to close the Global Functions dialog.

3) Add a button to the page, and add this script to its On Click event:

```
SayHello("Mr. Anderson");
```

This script *calls* the SayHello function that we defined on the Global Functions dialog, passing the string "Mr. Anderson" as the value for the function's "name" parameter.

4) Preview the project and click on the button object.

When you click on the button object, the script on the object's On Click event calls the SayHello function, which then displays its message.



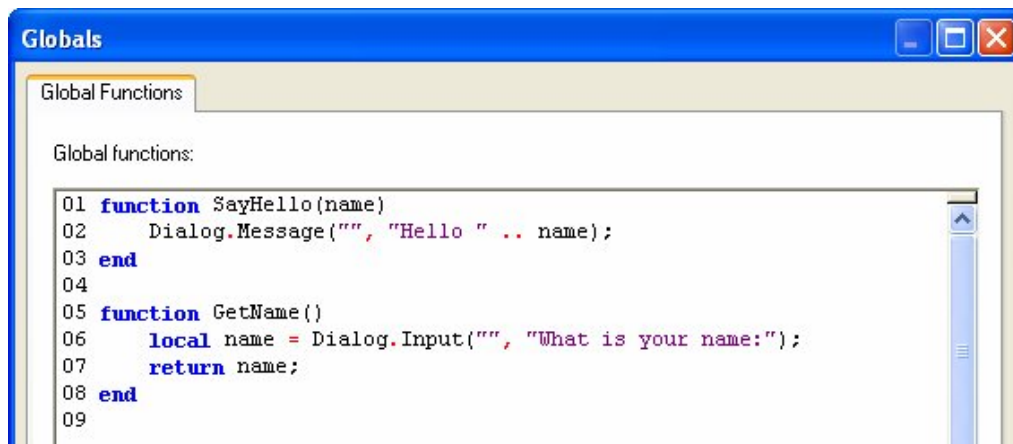
Note that the SayHello function was able to use the string that we passed to it in the message it displayed.

5) Exit the preview. Choose Project > Global Functions, and add the following script below the SayHello function:

```
function GetName()  
    local name = Dialog.Input("", "What is your name:");  
    return name;  
end
```

When you're done, click OK to close the dialog.

The end result should look like this:



This script defines a function called GetName that does not take any parameters. The first line inside the function uses a Dialog.Input action to display a message dialog with an input field on it asking the user to type in their name. The value returned from this action (i.e. the text that the user entered) is then stored in a local variable called name.

The “local” keyword makes the variable only exist inside this function. It’s essentially like saying, “for the rest of this function, whenever I use ‘name’ I’m referring to a temporary local variable, even if there’s a global ‘name’ variable which may happen to exist.” Using local variables inside functions is a good idea—it prevents you from changing the value of a global variable without meaning to. Of course, there are times when you *want* to change the value of a global variable, in which case you just won’t use the “local” keyword the first time you assign anything to the variable.

The second line inside the function returns the current value of the local “name” variable to the script that called the function.

Tip: We could actually make this function’s script fit on a single line, by getting rid of the variable completely. Instead of storing the return value from the Dialog.Input action in a variable, and then returning the contents of that variable, we could just put those two statements together, like so:

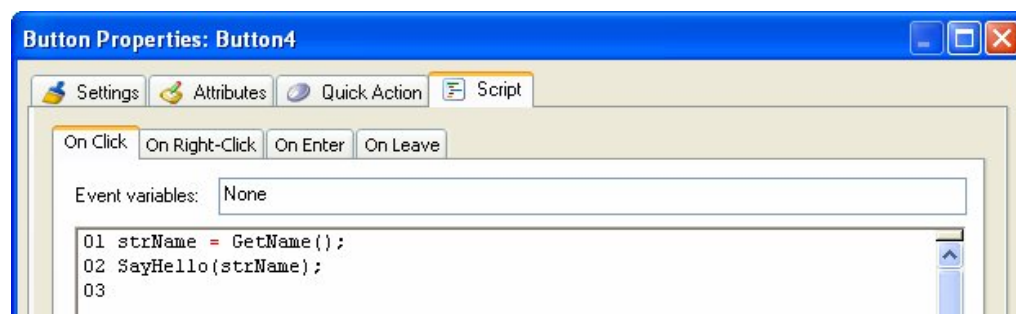
```
function GetName()  
    return Dialog.Input("", "What is your name:");  
end
```

This would make the GetName function return the value that was returned from the Dialog.Input action, without storing it in a variable first.

6) Edit the script in the button object’s On Click event so it looks like this instead:

```
strName = GetName();  
SayHello(strName);
```

It should look like this when you’re done:



The first line calls our `GetName` function to ask the user for their name, and then stores the value returned from `GetName` in a variable called `strName`.

The second line passes the value in `strName` to our `SayHello` function.

7) Preview the project. Try out the script by clicking on the button object.

When you click on the button object, an input dialog will appear, asking you to enter your name.



After you type in your name and click OK (or press Enter), a second dialog box will appear, greeting you by the name you entered.



Pretty neat, huh?

8) Exit the preview. Edit the script in that same button object's On Click event so it looks like this:

SayHello(GetName());

This version of the script does away with the `strName` variable altogether. Instead, it uses the return value from our `GetName` function as the argument for the `SayHello` function.

In other words, it passes the GetName function's return value directly to the SayHello function.

Whenever a function returns a value, you can use a call to the function in the same way you would use the value, or a variable containing the value. This allows you to use the return value from a function without having to come up with a unique name for a temporary variable.

9) Preview the project, and try out the script again. When you're done, exit the preview.

The script should work exactly the same as before: you'll be asked for your name, and then greeted with it.

This is just a simple example, but it should give you an idea of what an incredibly powerful feature functions are. With them, you can condense large pieces of script into simple function calls that are much easier to type and give you a single, central location to make changes to that script. They also let you create flexible "subroutines" that accept different parameters and return results, just like the built-in AutoPlay actions.

And despite all that power, they are really quite simple to use.

Where to Go from Here

Well, that's the last lesson. I hope you've enjoyed learning about AutoPlay, and have found this user's guide both useful and helpful.

The next chapter contains a more detailed guide to the AutoPlay scripting language.

Feel free to join your fellow AutoPlay Media Studio users in our online forums, where assistance and camaraderie abound. You can join in the fun by choosing Help > Online Forums right from the AutoPlay program menu.

While you're at it, be sure to check out the online program reference and the helpful list of "How do I...?" questions as well. (You can access these resources right from the Help menu.)

There is a lot more that you can learn about this product, if you ever find the need or desire to. Chances are, if there's something you want to do with AutoPlay, there's at least one way to do it, and probably more.

As one of our wisest users once said, AutoPlay is easy to learn, but difficult to master.

Lesson 10 Summary

In this lesson, you learned how to:

- Display a message
- Use a variable
- Add an if statement
- Test numeric values
- Set a button object's text
- Concatenate strings
- Compare strings
- Use a for loop
- Create functions

Scripting Guide

One of the most powerful features of AutoPlay Media Studio is its scripting engine. This chapter will delve a bit further into the scripting environment and language.

AutoPlay 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
```

The above example assigns a value to a variable, tests the contents of that variable, and if it meets the test (in this case, if the value is less than 10), uses an AutoPlay action named “Dialog.Message” to display a message to the user.

New programmers and experienced coders alike will find that AutoPlay 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

Before You Begin

In order to try out the example scripts in this chapter, you will need a basic project that you can build and run (or preview) in order to see scripts in action. Although any simple project will do, I recommend starting a new project using the “Blank” project template, which won’t have any pre-existing scripts inside it.

Tip: In addition to trying out the scripts in this chapter, you might want to explore some of the other project templates which already have working action scripts in them. These scripts are some practical examples of real-world solutions that can be very good examples to learn from.

AutoPlay scripts are always performed in response to an event, whether it’s a specific page being shown (an On Show event) or an object being clicked (an On Click event).

In order to try your hand at scripting, then, all you need is an event to put the action script “in” so it will get performed when you build and run your project. Three really good places to put script in order to try it out are:

On Startup

The project’s On Startup event is a good place to try any scripts that you want performed automatically, without having to click on a button or anything. Of course, since this event occurs as soon as the application starts up—even before the first page is shown—it’s not a good place to put any actions that operate on objects. (If you do, it either won’t work, or you’ll generate some kind of error.)

You can access the On Startup event by choosing Project > Actions.

Tip: If you just want to test actions in On Startup without even showing a single page, add an `Application.Exit()` action to the end of your script for the event. This action immediately exits from the application as soon as it is encountered in a script.

On Show

Each page has an On Show event which occurs as soon as the page is displayed. This can be a good place to try scripts that you want performed automatically but that need to interact with one or more objects on the page.

You can access a page’s On Show event by double-clicking on the page surface and clicking on the Script tab.

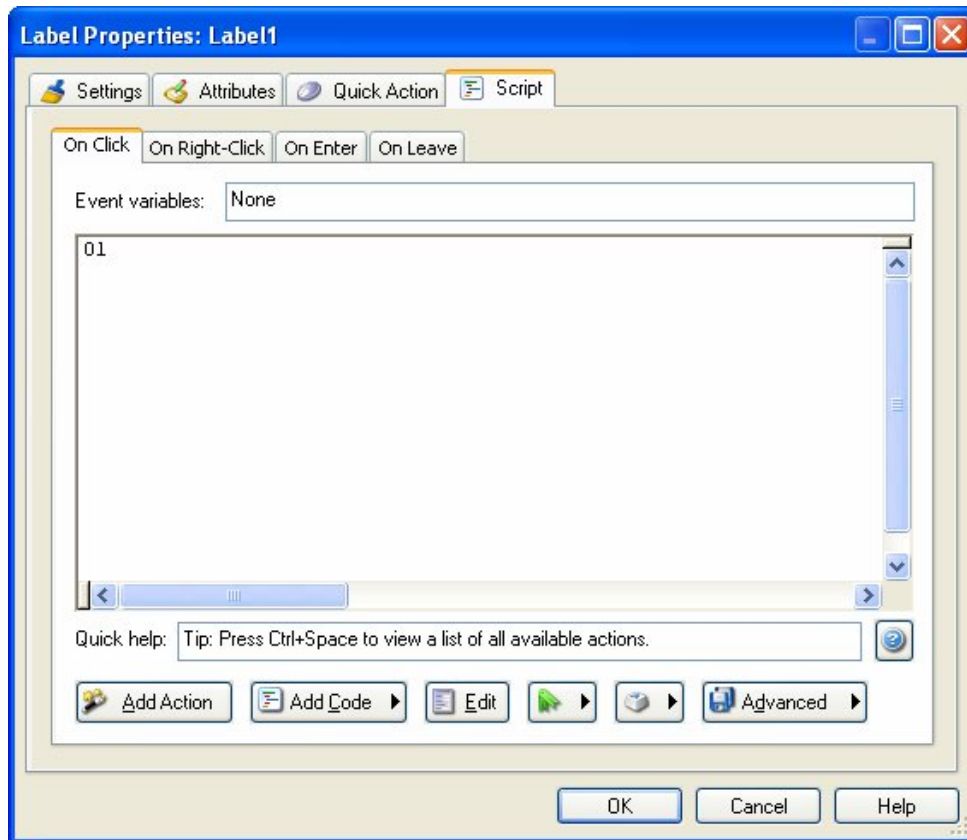
On Click

Probably the best place to try out scripts is in an object's On Click event. As its name suggests, the On Click event occurs whenever you click on the object.

You can access an object's On Click event by double-clicking on the object and clicking on the Script tab.

Label objects are especially good candidates for script tests, since they're easy to add to the page, and they don't take up much room (so you can fit lots of them on a single page). A good practice is to change each label object's text to describe the script that will be performed when you click on it, so you can easily tell them apart.

Button objects also work well (and have the added benefit of being really cool).



A label object's On Click event

Important Scripting Concepts

There are a few important things that you should know about the AutoPlay 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 “know” about other variables and functions declared elsewhere in the project. For example, if you assign “myvar = 10;” in the project’s On Startup event, myvar will still equal 10 when the next event is triggered. There are ways around this global nature (see *Variable Scope* on page 309), but it is generally true of the scripting engine that variables and functions are global by default.

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 AutoPlay is concerned.

The same principle applies to function names as well. For example:

```
Dialog.Message("Hi", "Hello World");
```

...refers to a built-in AutoPlay 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 those 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

Variables are very important to scripting in AutoPlay. Variables are essentially just “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 named “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 is a simple example 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 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 statement “b = a;” assigns the current value of “a” (which is 10) to “b.”

Variable Scope

As mentioned earlier in this chapter, all variables in AutoPlay are *global* by default. This just means that they exist project-wide, and hold their values from one event 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 the project's On Startup event, and then enter:

```
Dialog.Message("The value is:", foo);
```

...into a page's On Show event, the second script will use the value that was assigned to "foo" in the first script. As a result, when the page is shown, 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 On Startup event is triggered *before* the page's On Show event, so the value 10 is already assigned to foo when the On Show event's 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 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 in separate events, 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

AutoPlay'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 AutoPlay 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 AutoPlay: 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”). AutoPlay 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 = "Fox Mulder";
Dialog.Message("Title", "Hello, how are you?");
LastName = 'Mulder';
```

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 only double quotes for the second line, it would have to 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-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.

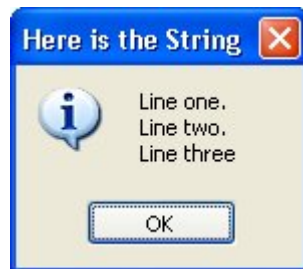
The escape sequences that you can use include:

- \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 across multiple lines by using double square brackets, i.e. `[[` and `]]`. A string defined 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 340.

nil

nil is a special value type. It basically represents the absence of any other kind of value.

In fact, it's often used when testing to see whether a variable is empty. (If a variable is equal to nil, it doesn't contain a value...it's essentially "undefined.")

You can assign nil to a variable, just like any other value. Note that this is *not* 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 = "Forgive and...";
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 “subroutines”), which are essentially small pieces of script that can be executed on demand. Each function has a name which is used to identify the function. You can actually use the function name (or even an entire function definition) as a special kind of value, which you can store in a variable as a sort of function “reference.” This kind of reference is of the *function* type.

For more information on functions, see page 335.

Table

Tables are a very powerful way to store lists of indexed values. Tables are actually associative arrays—that is, they are arrays which can be indexed not only with numbers, but with any kind of value (including strings).

Here are a few quick examples (we’ll cover tables in more detail on page 327):

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 = "Elisa";  
t.LastName = "Toffoli";  
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 = "Bruce";  
table_one.LastName = "Springsteen";  
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 332.

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

The concatenation operator is two periods with no spaces between them (..). It is used to combine two or more strings together. You don't have to put spaces before and after the two 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 330.

Tables (Arrays)

Tables are very useful. They can be used to store any type of value, even including functions or 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:

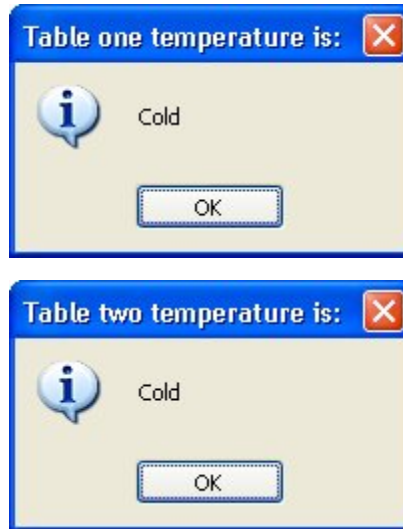
```
table_one = { mood="Happy", temperature="Warm" };

-- 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 that 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:

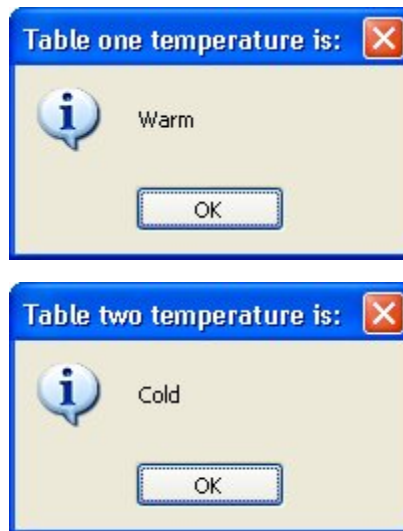


Table Actions

There are a number of table-related actions 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 actions, 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. Functions are simply portions of script that you can define, name and then execute (or “call”) from anywhere else.

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 then try out your application, nothing 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 any other script in the rest of the application.

This brings up an important point about scripting in AutoPlay. When making a function, the function does not get “into” the engine until the function definition is executed. That means that if you define HelloWorld() in a button’s On Click event, but that event never gets triggered (because the button is never clicked), the HelloWorld() function will not exist. That is, you will not be able to call it from anywhere else.

That is why, in general, it is best to define functions in the Global Functions script of the project. (To access the Global Functions script, choose Project > Global Functions from the menu.)

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 a parameter to the function.

A *parameter* is a special variable in the function definition that will be used to “receive” a value passed into the function. The value that actually gets passed into the function is known as an *argument*.

Tip: Programmers often confuse the terms “parameter” and “argument,” so you will often find parameters referred to as arguments, and vice-versa. This is understandable, since they both refer to almost the same thing: a value being passed to a function (an argument), and a local variable that represents the value in the function (a parameter).

```
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 local 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.

Note: Parameter variables are always local to the 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 has two parameter variables (one for each argument it can receive), and each function call passes two string arguments to the function.

Note that by changing the content of those strings, you can send different values 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 (i.e. has two parameters) 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 specify two variables to receive the return values 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 preceding script 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. (In fact, this is how all of the AutoPlay actions are implemented: each action category is actually a table containing the category’s actions.) 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 actions available to you in AutoPlay, 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;
```

However, you cannot put spaces between the two dots:

```
foo = "Hello " . .user_name; -- syntax error!
```

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 a little 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 action, to ensure that both strings have the same case before you compare them. The String.Upper action returns an all-uppercase version of the string it is given, and the String.Lower action returns an all-lowercase version. Note that it doesn't matter which action you use in your comparison, so long as you use the same action on both sides of the == operator in your if statement.

For example:

```
strOne = "Mooohahahaha";
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 action 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.

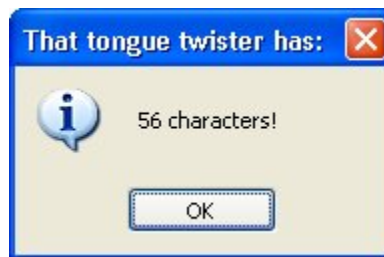
Tip: You could also use a String.CompareNoCase action to perform a case-insensitive comparison on the original strings.

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` action, 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 preceding example would display the following two messages:



Extracting Strings

There are three string actions that allow you to “extract” a portion of a string, rather than copying the entire string itself. These actions 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 actions 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.

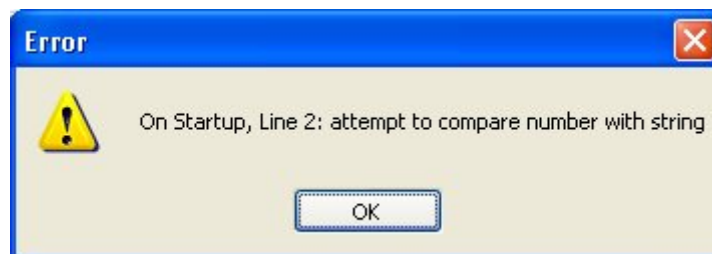
For example, if you have an input field where the user can enter their age, and you read in the text that they typed, you might get a value like “31”. Because they typed it in, though, this value is actually a string consisting of the characters “3” and “1”.

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 (when placed in the On Startup event):

```
age = "31";
if age > 18 then
    Dialog.Message("", "You're older than 18.");
end
```

...would produce the following error message:



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` action.

The `String.ToNumber` action 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.

Special Built-in Functions

There are three special 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" );
```

...and we included the file in our AutoPlay project, in the Scripts folder. Wherever the following line of script is added:

```
dofile( "AutoPlay\\Scripts\\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 and runs a script file into the scripting engine. 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( "AutoPlay\\Scripts\\foo.lua" );  
require( "AutoPlay\\Scripts\\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 the `require` function will only load a given script file once per session, it is best suited for loading scripts that contain only variables and function definitions. Since variables and function definitions are global by default, you only need to load them once for them to take effect; 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 from the various events in a project. 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 for the type of value in the variable. 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 to work with certain data types. 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

AutoPlay 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 AutoPlay as smartly and effectively as possible!

This section will explain AutoPlay’s error handling methods as well as cover a number of debugging techniques.

Error Handling

All of the built-in AutoPlay actions use the same basic error handling techniques. However, this is not necessarily true of any third-party functions 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 AutoPlay’s error handling system, they may not necessarily do so. Therefore, you should always consult a script’s author or documentation in order to find out how the error handling is, well, handled.

There are two kinds of errors that you can have in your scripts when calling AutoPlay 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 AutoPlay when you build the project or close the script editor.

For example, consider the following script:

```
foo =
```

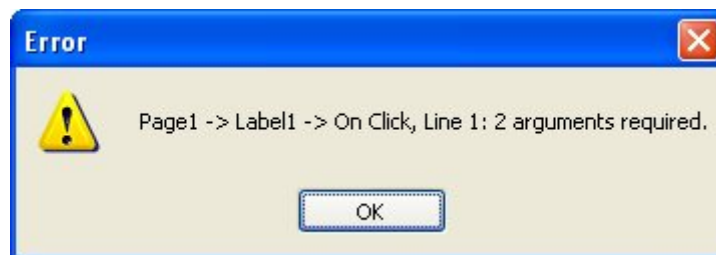
The preceding script is incorrect because we have not assigned anything to the variable `foo`—the script is incomplete. This is a pretty obvious syntax error, and would be caught by the scripting engine when you build your project.

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 script 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_does_not_exist.txt");
```

If you put that script into an event right now and try it, you will see that nothing appears to happen. This is because `AutoPlay`'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");
```

In this case you probably 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 it turns out that 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. As you'll see in the next section, this is where the `Debug` actions come in handy.

Debug Actions

AutoPlay comes with some very useful actions for debugging your scripts. 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:

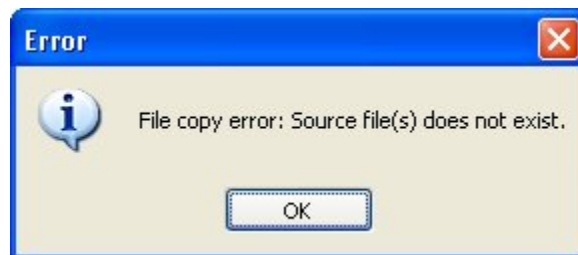


It would be much nicer to actually tell them some information about the exact problem. Well, you are in luck! There is a built-in table called `_tblErrorMessage`s that contains all possible error messages, indexed by their error codes. This means you can use the last error number to get an actual error message that will make more sense to the user than a number like “1021.”

For example, here is a modified script that displays 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: "  
        .. _tblErrorMessage[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

Every AutoPlay application has the ability to show a debug window that can be used to display debug messages. This window is available throughout the execution of your application, 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 run that script, the debug window will appear on top of your application, but nothing else will happen—yet. That’s where the following Debug actions come in.

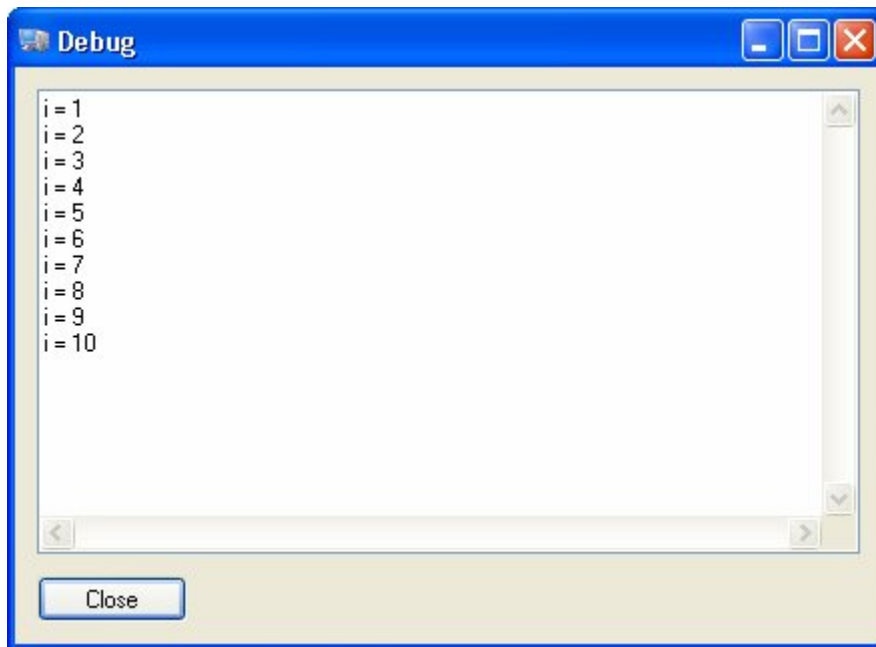
Debug.Print

The Debug.Print action outputs the text of your choosing to 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 of bread crumbs that you can trace in order to understand what’s happening behind the scenes in your project.

Debug.SetTraceMode

AutoPlay 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);
```

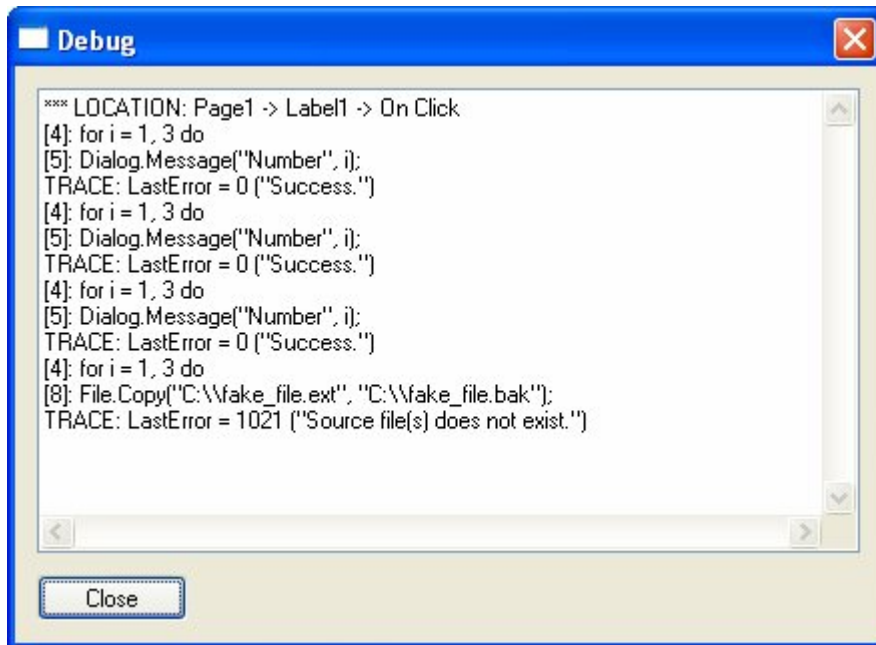
...where *turn_on* is a Boolean value that tells the program whether to turn the trace mode on or off. For example:

```
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 the preceding script produced the following output in the debug window:



Turning trace mode on is something that you will not likely want to do in your final, distributable application, but it can really help find problems during development.

Tip: Always test your application before you ship it to your users! For example, it's far less expensive (and embarrassing) to re-burn a master CD because you forgot to comment out a `Debug.ShowWindow()` or `Debug.SetTraceMode()` action than it is to re-deploy thousands of CDs to your users.

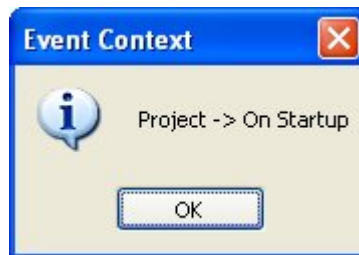
Debug.GetEventContext

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 the following script from the On Startup event:

```
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 scripting guide, and for good reason. This is a great action to use in 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 in order to "break" it into arbitrary stages. This can be helpful when you're not sure where an error is occurring in a script.

For example, the following script uses four dialogs to let you know how far you have progressed into a function:

```
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
```

If the error occurred after you saw temporary debug message “2”, you’d know the problem was somewhere between the lines where temporary debug messages “1” and “2” are shown.

Final Thoughts

Hopefully this chapter has helped you to understand scripting in AutoPlay Media Studio. Once you get the hang of it, it is a really fun and powerful way to get things done.

Other Resources

Here is a list of other places that you can go for help with scripting in AutoPlay.

Help File

The AutoPlay help file is packed with good reference material for all of the actions and events supported by AutoPlay, and for the design environment itself. You can access the help file at any time by choosing Help > AutoPlay Help from the menu.

Tip: If you are in the script editor and you want to learn more about an action, simply click on the action and press the F1 key on your keyboard.

AutoPlay Web Site

The AutoPlay web site is located at <http://www.indigorose.com/ams>. 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

AutoPlay'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 AutoPlay that are not officially supported by Indigo Rose. These functions, if any, are documented in the Lua 5.0 Reference Manual.

Only the actions 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

%

%TempLaunchFolder%, 347

|

\\r\\n, 355

—

_tblErrorMessage, 353

A

accessing table elements, 328

action category, 149, 183

Action Properties dialog, 166

actions, 131, 145, 158, 186, 227, 247, 258, 279, 289, 349

Actions category, 146

actions, cutting and pasting, 188

Add Action button, 149, 152, 183

Add Code button, 285, 293

add operator (+), 288

adding a panel image, 195, 211

adding a text banner, 213

adding a video object, 216

adding actions, 145

adding an if statement, 283

adding background music, 244

adding buttons, 131, 158

adding custom video controls, 222

adding page actions, 187

adding pages, 155

Align to Page mode, 102, 103, 117, 120, 198, 212

aligning objects, 98, 102, 144

Alignment setting, 65, 84, 180, 215

Alignment toolbar, 102, 103

alignment tools, 101, 103, 117, 166, 179, 198, 212

alpha transparency, 131

application, 155, 263, 267, 270

Application.Exit, 150, 151, 284, 352, 353, 354

Application.GetLastError, 352, 356

arguments, 294, 296, 335, 337

arithmetic operators, 319

arranging objects, 104

arrays. *See* tables

aspect ratio, 58, 220

assignment operator (=), 280

associative arrays, 329

Attributes category, 56, 118

Attributes tab, 159

audio channel, 248, 254

Audio folder, 256

Audio Settings dialog, 239, 241, 244

Audio.Load, 253

Audio.Pause, 251

Audio.Play, 258

Audio.Stop, 258

Audio.TogglePlay, 247

Auto Start, 219

autocomplete, 230

AutoPlay program window, 18

autorun.inf, 273

B

background, 37, 41

background color, 38, 177

background image, 41–47, 156

background music, 244, 247

Background Music tab, 244, 254

background style, 40

BackgroundColor, 215

BackgroundStyle, 215
 backslashes, 256
 begin playing automatically, 219
 black bars, eliminating, 220
 block, 283
 Bold, 70
 Boolean variables, 316, 348
 Border, 215
 Border Color, 215
 bounding box, 50, 58, 75, 92, 110, 116, 119, 125, 167, 177
 browse button, 223, 242
 building a compressed executable, 267
 building to a folder, 263
 burning a CD, 271
 button objects, 131, 165
 Button Properties dialog, 132, 159, 162
 Button.SetText, 289
 Buttons folder, 135

C

capitalization, 286
 category, 149
 category heading, 20, 73
 CD, 271
 CD_ROOT, 51, 53, 263
 CD-R/CD-RW option, 272
 centering objects, 99, 101, 103, 117
 changing text settings, 61, 138
 changing the default object sounds, 239
 CHANNEL_BACKGROUND, 248
 CHANNEL_NARRATION, 254
 cleaning unused resources. *See* resource optimizer
 Click color, 76, 86, 140, 181, 200, 214, 223
 click sound, 239, 242, 243
 Click state, 72
 clipboard, 164
 Close button, 221
 color chooser, 38, 78, 80, 82
 color square, 78, 80
 colors

- design environment, 27

 Colors dialog, 74, 78, 181
 comment operator (--), 291
 comments, 307
 comparing strings, 340
 compressed executable, 267
 Compressed Executable option, 268
 concatenating strings, 340
 concatenation operator (.), 289, 321, 340
 condition, 283, 287
 constants, 154
 control panel, 217, 218
 control structures, 323–27
 converting numeric strings into numbers, 345
 copying colors, 78, 181
 copying objects, 164
 copying tables, 332
 counting, 288, 292
 counting characters, 342
 creating functions, 294
 creating tables, 327
 ctrl-clicking, 91
 cursive fonts, 78
 cursor, 170
 cursor keys, 220
 Cursor setting, 170
 custom actions, 294
 custom color, 75
 custom sound effects, 242
 Custom tab, 75, 79
 custom video controls, 222

D

debug actions, 352
 debug window, 354, 355
 Debug.GetEventContext, 357
 Debug.Print, 355
 Debug.SetTraceMode, 356
 Debug.ShowWindow, 354, 356
 debugging your scripts, 349
 default object sounds, 239
 default values, 282

- default web browser, 154
- deleting objects, 52
- Desert Sunrise*, 41, 44
- design environment, 17, 18, 19, 27
- Dialog.Input, 287, 297, 298
- Dialog.Message, 279, 281, 282, 284, 287, 291, 302, 309, 328, 329, 339, 350, 351, 352, 353, 354, 356, 357
- displaying a message, 279
- distributing objects, 120
- Docs folder, 114, 154
- dofile, 347
- dominant object, 92, 93, 99, 179
- double quotes, 312
- Down state, 140
- drag and drop assistant, 162, 203, 224
- drag selecting, 94
- draw order, 106
- duplicating objects, 60, 141
- duplicating pages, 157
- dynamic text, 182

E

- edit button, 63, 201
- Edit Text dialog, 63, 84, 202
- editing multiple objects, 98
- Effects folder, 240, 242
- else, 287, 323
- elseif, 323
- email, 169, 172
- empty string (""), 185, 186, 187
- Encrypt Data Segment option, 269
- encryption, 267
- enumerating tables, 330
- equality (==), 285
- error handling, 349
- escape sequence, 184, 256
- escape sequences, 313, 355
- escaping backslashes, 314
- escaping strings, 313
- event tabs, 148, 160
- events*, 145, 151, 158, 182, 183, 186, 258, 282, 289

- executable, 263, 267, 268
- expressions, 319
- extracting strings, 344

F

- F1 help, 24, 358
- Family setting, 70, 138
- File setting, 165, 223, 224
- File.Copy, 351, 352, 353, 354, 356
- File.Open, 216
- File.OpenEmail, 242
- File.OpenURL, 152, 155
- File.Run, 216
- finding strings, 342
- Folder.Delete, 351
- Font dialog, 67
- font family, 70, 138
- font gotchas, 78
- font settings, changing, 67
- font size, 58, 138
- for, 291, 292, 326, 330
- forums, 24, 300
- frame effects, 195
- function arguments, 336
- function definition, 336
- function prototype, 350
- function variable, 294
- functional errors, 351
- functions, 294, 300, 317, 335, 348

G

- Gallery pane, 45, 195, 211, 224
- Gallery tab, 41, 49
- GetName function, 297, 299
- global functions, 336
- Global Functions dialog, 294
- global variables, 309
- Gradient Color, 41
- grid, 24
- grouping objects, 116, 226

H

- Hard Drive Folder option, 264

Height, 56, 165, 178
HelloWorld function, 335, 337, 339
help, 24
help file, 358
hexadecimal, 39, 40, 73, 81, 181
Highlight color, 73, 75, 86, 139, 181, 200, 214, 223
highlight sound, 239, 240, 241
Highlight state, 72, 139
hotkeys, 61
HTML files, 154
hyperlinks, 72, 74

I

Icons folder, 106
if, 283, 286, 287, 289, 323
image object, 48
Image Properties dialog, 48
Images folder, 54, 156
Images subfolder, 49
inequality (~=), 286
initialization, 282
input dialog, 299
input field, 287, 297
interactive buttons, 131
interactive labels, 72, 86
Italic, 180

J

joining strings, 289

K

Keep Aspect, 55
Keep Scaling Mode, 220
keyboard shortcuts, 11
knowledge base, 25

L

label objects, 56
Label Properties dialog, 57, 82
Left, 56, 84, 165
loading an audio file, 251

local variables, 294, 298, 309
locking objects, 125, 190
logical operator, 291
logical operators, 321
Loop setting, 219, 254
loops, 291
Lua variables, 308
Lua web site, 359

M

mailto: link, 170
Make Same Height, 138
Make Same Size, 138
Make Same Width, 138
Match Normal button, 82, 181, 200, 214
matching colors, 82
Maximize button, 17
menu commands, 11
More Colors..., 73
mouse, 72
mouse pointer, 170
mouse position, 51
moving actions, 188
moving objects, 51, 111
MP3, 240
multi-line comments, 307
multiple arguments, 337
multiple lines, 62
multiple objects, 88

N

Name, 156
Name setting, 92, 93
naming objects, 66, 164, 178, 223, 225
naming variables, 281, 310
navigation buttons, 158
New Action wizard, 149, 152, 183
New Button Object button, 132
New Label Object button, 59
New Project dialog, 15
newline (\n), 63, 184
nil, 315, 348
Normal color, 73, 86, 181, 200, 214, 223

Normal state, 72
not equal (~=), 286
notes, 291
null characters, 312
numbers, 312, 348
numeric arrays, 328
numeric tests, 288

O

object browser, 109, 118, 122
object position, 51, 56
object size, 51, 56
objects
 aligning, 98, 102, 144
 arranging, 104
 centering, 99, 101, 103, 117
 copying, 164
 deleting, 52
 deselecting, 51, 94
 distributing, 120
 drag selecting, 94
 duplicating, 60, 141
 editing, 98
 grouping, 116, 226
 locking, 125, 190
 moving, 51, 96, 111
 naming, 66, 164, 178, 223, 225
 pinning, 119, 167, 198, 213
 resizing, 58, 111
 selecting, 51, 91, 100
 ungrouping, 227
 unlocking, 126
Ogg Vorbis, 240
On Click, 145, 158, 160, 162, 166, 186,
 227, 230, 279, 280, 281, 285, 289, 292,
 296
On Close, 187, 249, 258
On Enter, 182, 183, 186
On Leave, 182, 185, 186
On Preload, 187, 188, 282, 290
On Show, 187, 249, 251
On Startup, 295, 306, 357
online forums, 24, 300

online help, 24
opacity, 112
operator associativity, 322
operator precedence, 322
operators, 319
order of table elements, 332
Orientation setting, 69
output folder, 263, 265, 266
overlapping objects. *See* arranging objects
overriding functions, 338

P

page, 37
 background, 37
 settings, 37
 shadow, 27
 size, 31
page actions, 187
page background, 157, 195
page events, 187, 247, 249, 282
page manager, 158, 162
page name, 156, 157
Page Properties dialog, 47
page surface, 19, 37, 38, 41, 47, 157
page tab, 155, 156, 164, 195, 211
Page.Jump, 158, 160, 162, 166, 186
Page.Navigate, 167
pages
 adding, 155
 duplicating, 157
 previewing, 221
 renaming, 157
panel images, 195, 211
Panels folder, 196, 211
panes, 19, 22
 pinning and unpinning, 23
 tabbed, 22
paragraph objects, 177, 192, 199, 203, 213
Paragraph Properties dialog, 182, 214
Paragraph.SetText, 183, 184, 186, 187
parameters, 152, 154, 160, 166, 184, 185,
 229, 233, 248, 253, 289, 294, 335
pausing the background audio, 247

- pinning objects, 119, 167, 198, 213
- PlayAutomatic, 254
- playing an audio file, 251
- preferences, 25
 - design environment, 27
 - project size, 28
- prefixes, 281
- Preview pane, 45, 137
- previewing, 31, 85
- previewing a single page, 221
- program menu, 18
- program window, 18
- programming. *See* scripting
- Progress Window settings, 269
- project, 54
 - naming, 16
 - previewing, 85
 - saving, 32, 85
 - starting a new, 14
- project file, 14, 16
- project folder, 14, 16, 53, 113
- Project pane, 53, 113, 134, 156
- project settings, 29
- project size, 28
- project template, 16
- properties pane, 20, 38, 50, 94, 134, 146, 151
 - closing, 20
 - opening, 20
- Publish wizard, 263, 267, 271
- publishing a CD, 271
- publishing a compressed executable, 267
- publishing to a folder, 263
- putting functions in tables, 339

Q

- quick action, 169
- Quick Help, 232

R

- red squiggles, 64
- redefining functions, 338
- redo, 97

- relational operators, 320
- relative paths, 51
- Remove Unused Resources dialog, 114
- removing objects, 52
- renaming pages, 157
- repeat, 325
- repeating actions, 291
- replacing strings, 343
- require, 347
- reserved keywords, 311
- resize handles, 54, 58, 92, 119, 167
- resizing objects, 111
- resource optimizer, 113
- Restore button, 17
- Restore Size, 56
- return, 294, 297, 335, 337
- return values, 294, 300
- returning multiple values, 338
- returning values, 337
- revert, 115
- RGB, 40, 75

S

- saving, 85, 106
- SayHello function, 296, 299
- script editor, 147, 151, 159, 166, 186, 188
- Script editor, 290
- Script tab, 159, 229
- scripting, 276, 279, 302
 - comments, 307
 - important scripting concepts, 306
 - variables, 308
- scripting resources, 358–59
- scroll bar images, 206
- scroll bars, 179, 199, 216
- Scroll Style, 204
- scrolling text, 192
- security, 269
- select a line of text, 188
- Select File dialog, 41, 43
- selecting multiple objects, 91
- selecting objects, 51, 100
- semi-colon, 233

- sending email, 169
- setting object-specific sound effects, 242
- Settings tab, 229
- Setup Factory, 266, 267
- shadow effects, 131
- shift-dragging, 225
- sideways text, 69
- single executable file, 267
- single quotes, 313
- Size setting, 70, 180
- size threshold, 47
- skin files, 206, 218
- sound effects, 239, 242
- Sound Effects tab, 239, 241
- soundtrack, 244
- spacing objects. *See* distributing objects
- spell check, 64, 202
- standard highlight sound, 240, 241
- Standard tab, 74
- Standard toolbar, 168
- State Colors category, 72
- states, 82
 - Click, 72
 - Disabled, 72, 131
 - Down, 131
 - Highlight, 72, 131
 - Normal, 72
 - Up, 131
- status bar, 19, 51
- status text, 174, 177
- storing functions in tables, 339
- String.CompareNoCase, 288, 341
- String.Find, 342
- String.Left, 344
- String.Length, 342
- String.Lower, 341
- String.Mid, 344
- String.Replace, 343
- String.Right, 344
- String.ToNumber, 346
- String.Upper, 287, 288, 341
- strings, 160, 184, 233, 256, 279, 281, 282, 289, 312, 348

- comparing, 340
- concatenating, 340
- converting to numbers, 345
- counting characters, 342
- extracting, 344
- finding, 342
- manipulating, 346
- replacing, 343
- subroutines, 300
- SW_SHOWNORMAL, 154
- syntax errors, 349

T

- table functions, 334
- tables, 317, 327–34, 348
 - accessing elements, 328
 - associative arrays, 329
 - copying, 332
 - creating, 327
 - enumerating, 330
 - numeric arrays, 328
 - order of elements, 332
- target media size, 28
- technical support, 25, 359
- TedSellers.txt, 203
- terminator, 233
- test, 283
- testing a numeric value, 288
- Text, 165
- text colors, 72, 138
- text offset, 224
- Text setting, 201, 214
- TextFile.ReadToString, 351
- then, 287
- this, 289
- title bar, 30, 31, 221
- toolbars, 18
- Top, 56, 84, 179
- transparency, 112
- transports, 218
- TrueUpdate, 18
- types, 311, 348

U

- undo, 97, 106
- undocumented functions, 359
- ungrouping objects, 227
- unlocking objects, 126
- unused resources, 113
- up folder, 107, 156
- updating AutoPlay, 18
- user forums, 24
- using a for loop, 291
- using a variable, 280

V

- V. Scroll, 216
- values, 311
- variable assignment, 318
- variable names, 281
- variable scope, 309
- variables, 280, 281, 287, 289, 291, 292, 308
 - naming, 310
- video actions, 227
- video controls, 217, 222
- video objects, 216

- Video Properties dialog, 216
- Video.Pause, 230
- Video.Play, 227
- Video.Stop, 230
- volume, 245

W

- web browser, 153, 154
- web site, 359
- Welcome dialog, 15
- while, 324
- Width, 56, 165, 178
- width and height, matching the, 137
- window title, 30
- windowed objects, 106
- work area, 20, 100
- workspace layouts, 23

Y

- y offset, 224

Z

- z-order, 104, 106, 108, 216

