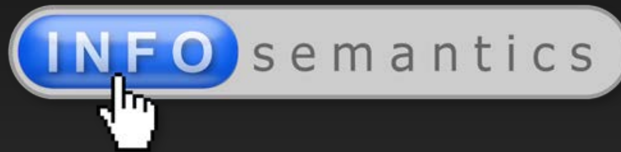


Guide to Adobe Captivate Advanced Actions

How to take your Captivate e-learning to the next level

www.infosemantics.com.au

www.infosemantics.com.au

Foreword

Copyright statement

© 2013-2014 by Rod Ward, Infosemantics Pty Ltd. ALL RIGHTS RESERVED

info@infosemantics.com.au

This book contains material protected under **International and Federal Copyright Laws and Treaties**. Any unauthorized reprint or use of this material is prohibited. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system without express written permission from the author / publisher, or as part of a multi-user license.

Multi-user licensing for this e-book

Most likely you have purchased this e-book for your own use. As stipulated in the copyright statement above, the licensed purchaser of this e-book is not permitted to distribute this book to other parties.

However, if you have purchased this publication as part of a **multi-user license** deal, then you are allowed to provide one copy of this document to each of the licensed end-users, but you must not exceed the maximum allowed number of licenses.

For example, if you purchase a three-user (**3x-user**) license, then you are allowed to distribute three copies of the document to end users, but no more. (Licensing options are available for **single**, **3x**, **5x**, **10x**, **25x**, and **unlimited** users.)

About the author – Rod Ward

I work as a contract **Technical Author** and **E-Learning Developer** for medium-large companies in Australia. In the course of my career I've used many different e-learning authoring tools, including every version of Captivate since its initial release by **Macromedia** around 2004. Over many years of professional use, I developed an intimate knowledge of how Captivate work…and more importantly for you…why it sometimes *doesn't* work as expected.

When Cp4 introduced the ability to build custom 'widgets', my company Infosemantics Pty Ltd started creating and selling our own range of ActionScript 3 widgets to extend the capabilities of Captivate as an e-learning authoring tool. Developing widgets allowed us even greater insight into the inner workings of Captivate. At our website (www.infosemantics.com.au) the information is almost entirely dedicated to assisting e-learning authors using Adobe Captivate and other related e-learning authoring tools.

Table of Contents

Foreword 2
Copyright statement ... 2
About the author – Rod Ward ... 2
Is this book for you? 6
Prerequisites ... 6
What areas does this book cover? ... 6
Other questions you may have about this book ... 6
About 'Programming' E-learning with Captivate 8
Why you need a production process ... 8
About Advanced Actions ... 8
Can advanced actions really be called 'programming'? ... 9
What you CAN do with advanced actions ... 10
What you CANNOT do with advanced actions ... 10
What if I want to go beyond the limitations? ... 11
Introduction to Captivate Variables 13
Captivate variable types ... 13
Create user variables ... 14
Naming your user variables ... 15
About variable data typing ... 17
Editing user variables ... 17
Removing user variables ... 17
Tracing variable usage in Cp7.1 ... 18
Displaying variables on slides at run-time ... 19
How to display project information ... 19
Overcoming dynamic text formatting issues ... 22
Boolean variables ... 23
Migrating variables from one project to another ... 23
Understanding Captivate's Run-time Events 24
Slide events ... 24
Interactive object events ... 27
Run-time events offered by interactive objects ... 29
Rollover slidelet events ... 30
Drag and drop events ... 30
How to decide which objects and events to use ... 31
Event scenarios ... 31
What 'infinite attempts' *really* means ... 34
Using Single Actions 36
Executable actions ... 36
Why are actions sometimes disabled or missing from the actions list? ... 45
About Expressions ... 46

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

Using Standard Actions 48
Create a standard action ... 48
A key difference between single and standard actions ... 50
A closer look at the Advanced Actions dialog ... 51
Editing Variables from within the Advanced Actions dialog ... 54

Using Conditional Actions 55
Why I use conditional actions and not standard actions ... 55
Creating conditional actions ... 55
About decision blocks ... 56
Order of run-time decision block execution ... 59
About IF > THEN > ELSE ... 59
About condition statements ... 60
Stacking up multiple AND / OR condition statements ... 62
Completing a conditional action ... 64
Adding multiple decision blocks ... 65
Preview a conditional action ... 65

Shared Actions 67
Replicating actions and variables across projects ... 67
Create a new shared action ... 68
Export shared actions ... 70
Import shared actions to a different project ... 71
How to import shared actions from a shared Library ... 72
Create new advanced actions from a shared action ... 72
Executing shared actions directly ... 74
The long list of shared action gotchas ... 75
Recommendations about using shared actions ... 77
How to copy and paste actions across projects ... 77

Debugging Advanced Actions 80
Setting up a debugging environment ... 80
How to set up a DEBUG slide in Cp5 or 5.5 ... 80
How to set up a DEBUG slide for Cp6 or Cp7 ... 82
General debugging tips ... 84
Standard action does not release the playhead ... 84
Interactive objects do not work in published output ... 85
Actions revert to Continue or Go to Next Slide ... 87

Capturing User Input 89
Make something clickable ... 89
Using click boxes ... 89
Using buttons ... 91
Using Smart Shape buttons ... 92
Using Widgets ... 93
Responding to mouse rollover events ... 94
Capturing text input ... 95
Text-entry boxes (TEBs) ... 95

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

TEB validation ... 97

Practice Exercises 99

General exercises ... 99
Create a dummy condition statement ... 99
Creating toggle actions ... 101
Single toggle action (Captivate 7.0.1 or above) ... 101
Toggle action from a standard action (Captivate 7.0.1 or above) ... 102
Toggle using a standard action Expression ... 103
Toggle using a conditional action ... 104
Toggle the visibility of an image ... 105
Toggle between two different images ... 108
Cycle through an array of objects ... 111
Dynamically change the text in a button ... 114
Dynamic navigation ... 117
Create a menu slide ... 117
Set up dynamic feedback on menu slides ... 119
Hide or lock navigation until sections of a module have been viewed ... 123
Advanced dynamic feedback for menu slides ... 128
Rewind back to the beginning of a slide ... 140
Capturing and validating user data ... 144
Capture user data using TEBs ... 144
Use a NULL variable to validate for empty TEB fields ... 147
Highlight data entry fields containing invalid data ... 150
Validate data involving multiple correct answers ... 153
Standardize message text in user variables ... 158
Concatenate variable values ... 160
Getting further with dates ... 164
Usability issues with confusing short date formats ... 164
System variables used for constructing dates ... 164
Show the duration of a module in minutes ... 165
Set up your own custom date format variables ... 168

Suggested Extra Resources 175

Ask me! ... 175
Forums ... 175
Adobe Captivate User Forums ... 175
Captivate on LinkedIn ... 175
Captivate on Facebook ... 175
Adobe support ... 175
Tutorials you can view ... 176

Appendix: Reserved keywords 177

Appendix: Keystroke Shortcuts 179

Appendix: Captivate System Variables 186

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

Is this book for you?

Well that depends…Most publications about **Adobe Captivate** (**Cp** for short) are aimed at newbie users and therefore only explain the basic techniques that you can also usually find out by just reading the online **Help** files or browsing free resources on the internet. If that's what you were expecting here then I need to tell you this e-book is different.

Prerequisites

This book assumes you're not a Captivate newbie; a beginner with no knowledge of the software. You should ***already know*** most of the basic techniques required to create an Cp e-learning course.

For example, you should already be able to:

- Create and edit project files;
- Add, edit, and delete slides as well as assign master slides;
- Insert all types of standard objects (**captions**, **highlight boxes**, **images**, **rollover captions**, **rollover images**) or interactive objects (**buttons**, **click boxes**, **Smart Shapes**, **text-entry boxes**) to your slides and set their properties;
- Record screen captures to create software tutorials;
- Record and edit voiceover audio (with closed captioning if necessary);
- Create interactive quizzes using Captivate's standard quiz slide types;
- Publish your content for different output formats (**SWF**, **PDF**, **HTML5**, **EXE**); and
- Configure your output to integrate with a **Learning Management System** (**LMS**).

So what's left? Well, this book is one of a series designed by Infosemantics for Captivate authors wanting to take their skills to the ***next level*** and learn some of the **specialized techniques** that only usually come from spending many years as a professional e-learning developer.

What areas does this book cover?

Specifically, this book explains how to add interactivity, personalization or customize your e-learning content using variables, simple actions, standard actions, and conditional actions.

I approach each topic assuming you have zero knowledge of advanced actions. But although this book starts out easy, I should warn you that it gets progressively deeper and more technical. So if you're only interested in doing *simple stuff* with Adobe Captivate, stop reading now, before you get hooked on all the power that lies just under the surface of this amazing software tool. Once you enter the intoxicating world of **advanced actions**, there's no turning back!

Other questions you may have about this book

You may have one or more of the following queries, so let's get them out of the way right now.

Which versions of Captivate does this e-book apply to?

The information in this publication applies to all versions of Captivate from **Cp5** onward but example screenshots will usually be from **Cp7**. Where a technique is not possible with a particular Cp version (eg because a required feature only came with later versions) I will point this out in the text. There's often a workaround that may enable you to still achieve similar results.

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

Does this apply to PC only or can MAC users still benefit?

In principle all of the advice in this e-book should apply equally to either PC or MAC. **Cp6** was the first version able to run natively on MAC OS. However, since I only work with PCs, I'm unable to verify all solutions work flawlessly on MACs. So, if you're a MAC user, feel free to inform me about any errors or omissions, and I'll include that information in future updates of this e-book.

What's the best way to use this information?

Don't try to read this book like a novel. It's too technical (and it has no plot). I suggest you first read through the Table of Contents and get a feel for the high-level topics. The initial paragraphs of each chapter give an overview of the content. So a first pass at the book might be to read these.

After that, go back to the beginning and work your way through the chapters from beginning to end. Don't skip the early chapters that explain foundational techniques about using variables and advanced actions. I've structured the chapters so that they get progressively more difficult and complex. Many examples or projects described in later chapters depend heavily on your ability to perform tasks explained in earlier chapters. Where something you need to do would require you to have read something else first, you'll find internal hyper-links to jump back to the relevant part.

Can I use a keyword search?

Yes. Just hit **Ctrl + F** to open the search field in the **Acrobat** window. The arrows under the **Find** field will take you to the **Previous** and **Next** occurrences of your search keyword in the content.

Will there be free updates for this e-book after my initial purchase?

That's my intention. But I can't promise how often. To download the latest e-book version at any time, log into the Infosemantics website with the username and password you set up when you made the original purchase. Then go to **My Account > Files** and click the download link found there.

Can I email you about a specific Captivate question I have?

Yes! If you purchased this book, you can ask questions, but there's one condition…

...before sending email, search this e-book for the answer FIRST!

Please don't be lazy and expect me to design and create your solutions for you. If the answer would require too much of my time to work out, and I'm not able to provide the solution free of charge, you have the option of purchasing some consulting time so that I can come up with a solution and send you files to use or study. (I have to make a living too you know.)

Can I send feedback about this book?

Please do! Send email to: info@infosemantics.com.au for any of the following reasons:

- If you find this book useful, I'd like to hear **why** or **how** it helped you.
- If you find **a mistake or error**, I'd **DEFINITELY** like to know about that so that I can fix it!
- If you think there's **something I missed out** that should be included in future versions.
- If you would like to see **e-books** about other areas of developing e-learning with Adobe Captivate, I'd like to hear your ideas too.

Go for it!

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

About 'Programming' E-learning with Captivate

This chapter sets the limits of what you can reasonably expect to achieve when working with Captivate's advanced actions 'programming' environment. It lists some of the creative possibilities, but also candidly discusses some of the limitations you need to be aware of.

Let me start this discussion by pointing out something to which most Captivate authors never give much thought, and that is the need for some kind of ordered process when developing e-learning.

Why you need a production process

When you create an e-learning module you're really creating a piece of *software*. But, just as with developing any software application, your e-learning needs to go through a process that helps to avoid chaotic decisions, eliminate errors, and reduce wasted effort.

Your process might include phases such as the following well-known **ADDIE** methodology:

- **Analysis** – where you work out what it is you need to do and how you should approach it;
- **Design** – including instructional design, interaction design, and technical design to solve problems and deliver objectives;
- **Development** – where you execute the design, then perform rigorous testing and debugging until everything *reliably* works as expected;
- **Implementation** – where content is finally deployed (often to a Learning Management System or LMS for short).
- **Evaluation** – where outcomes or results of your project are examined and decisions are made for possible future iterations of the process.

This book doesn't delve much into the **analysis**, **implementation**, or **evaluation** phases of an e-learning development workflow (those will be covered in other e-books). But we do discuss a lot about technicalities encountered in the middle sections of the process where you design, develop and debug your customized interactivity.

The reason I'm pointing out the wisdom of following some kind of design and development process is that I don't want you to be like a lot of Cp users that mistakenly think developing e-learning is all about adding 'bells and whistles' to impress the audience. The kind of enhancements to interactivity that you'll be capable of including in your projects once you master variables and advanced actions (hopefully as a result of studying this book) would certainly impress many clients. However, there's already too much e-learning in the world that was designed to *impress* rather than to *teach*.

So while I certainly hope you get excited about the creative possibilities we'll be demonstrating in this e-book, I also want to encourage restraint in how you implement them. There should always be an instructional design objective underpinning any feature you include in your e-learning.

Rule #1: Just because you **CAN** do something, doesn't mean you **SHOULD**.

About Advanced Actions

Wanting to open up more creative possibilities for e-learning developers, the architects of Captivate kindly provided a mechanism allowing *non-programmers* to customize interactivity of their e-learning content. This all comes under an umbrella term of "**Advanced Actions**". It's really just a simplified

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

method of scripting to customize how Cp e-learning output works at runtime.

For me personally, advanced actions are ***THE*** killer feature of Captivate that make it my go-to tool for creating custom interactivity in e-learning courses. Of course Captivate is by no means the *first* or *only* rapid e-learning authoring tool to have this capability. In fact most (if not all) other tools now offer some form of support for scripting. As a professional e-learning developer, I also have other rapid e-learning authoring applications in my tool-kit that enable some level of scripting. But, for my money, Captivate just seems to take things that little bit further.

Captivate offers four different ways to execute actions and uses the broad term **Advanced Actions** to encompass them as a whole. Here's a quick summary of what they are and how they differ from one another:

Single actions

Adobe just calls these 'actions' in the help files, but I prefer to use the term '*single* actions' because it better describes how they differ from the other types we discuss below. These are the actions you can assign directly to any run-time event via the **Properties tab > Actions** accordion. Their limitation is that they only allow you to execute *one action per event* (hence the reason for the name). If you need to execute *multiple* actions from a *single* event, then you need one of the next action types.

Another limitation of single actions is that you have to set them up each and every time. You cannot save and reuse them as you can with the ones we discuss next.

Standard actions

These are re-usable actions you create via the **Project > Advanced Actions** dialog (or **SHIFT + F9**). They allow you to execute one or more **single actions** in a sequence. After creating a standard action you need to trigger (execute) it using any one of Captivate's dozen or more run-time events.

Conditional actions

I love conditional actions! They are the unsung misunderstood work-horses of advanced actions. Like **standard actions** they also allow you to execute any number of **single actions** from a single run-time event, but with the bonus of being able to specify multiple **IF>THEN>ELSE** conditions that determine whether or not groups of actions get executed at all. This then allows your content to respond to user interaction and make decisions on-the-fly. And this is where Captivate gets really exciting!

Shared actions

Cp7 introduced a new type of action called Shared Actions. In theory they allow standard and conditional actions to be saved in a stripped-down 'parameterized' format that can then be exported from one project file and imported into another. In practice they tend to be more trouble than they are worth (in my opinion). Unless the actions are very complex, in the vast majority of cases it's going to be quicker to set them up again from scratch in the target project.

Other than making actions portable across projects, shared actions don't really allow you to do anything more than the other action types. They're something to watch for the future but I wouldn't get too excited about them in their present form. There's a special chapter dedicated to Shared Actions later in this e-book.

Can advanced actions really be called 'programming'?

Strictly speaking, Cp's advanced actions would not qualify as 'programming' if compared to languages such as **JavaScript**, **ActionScript**, **Java** or **C++**, etc. Many of the established concepts that are regarded as essential to modern object-oriented programming are missing completely from advanced

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

actions. So it's probably more accurate to refer to what we're doing here as 'scripting'.

When you publish your project, Captivate converts your variables and advanced actions into programming code that is executed at runtime. For **SWF** output, the code will be **ActionScript**. If the output is **HTML5**, then your code will be **JavaScript** and **CSS**. Either way, it's likely to be far more complex code than you could have written by yourself (unless you're already a talented programmer).

Cp's simplified (and somewhat clunky) advanced actions interface shields you from the dizzying complexity that hides just 'under the hood' of the project CPTX file. And, although there are lots of limitations in what you can do, some very sophisticated results are still achievable within those limits.

What you CAN do with advanced actions

So what level of sophistication are we talking about here? Well consider the following examples:

- Create your own custom User Variables and display them in your content at runtime;
- Use dozens of System Variables that Captivate exposes for you;
- Use run-time events to execute Simple Actions chosen from a menu;
- Define a series of actions to be performed in sequence, and save this as a Standard Action that can be reused and executed by events on any number of slides;
- Create **IF – THEN – ELSE** conditional actions that determine whether or not certain actions (or sets of actions) are executed;
- Create dynamic content that responds to user input;
- Manipulate numeric variable values to calculate results or scores;
- Hide or show objects at runtime;
- Change the appearance of objects to indicate state;
- Concatenate strings to assemble sentences or create your own custom date formats from component system variables;
- Create dynamic navigation menus that indicate which sections a user has completed;
- Simulate navigation components such as tabs, drop-down or fly-out menus;
- Store user details in custom variables and then use these later in the content;
- Create custom quiz interactions;
- Share actions between projects (only in Cp7 and later versions).

The list of what you **can** do is long and growing as clever Cp authors come up with new ways to bend Captivate to their will. We will be demonstrating how to do all of the above later in this e-book.

What you CANNOT do with advanced actions

So what are some of the limitations mentioned above? Here are just a few examples:

Arrays

Each Cp variable can only store one value. This means you cannot create a variable as an indexed array containing multiple values. Additionally, you cannot assign an object as the value of a variable.

Functions

As mentioned above, you can define a series of actions to be performed in sequence, and save this as a Standard Action or Conditional Action to be triggered by screen events or user interaction. But

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

this is not really the same as being able to program with reusable functions. For example, you cannot call one advanced action from another advanced action.

Loops

If you need to do something repetitively in software, you will usually end up using a loop. This is a logic structure where the program asks a question, and if the answer requires an action, then it is performed and the original question is asked again and again until the assigned action is no longer required. Captivate has Conditional Actions that allow you to ask questions (**IF** conditions) and **THEN** execute actions, **ELSE** do something else. You can fake a simple loop by create multiple conditional clauses that do essentially the same thing over and over again to roughly approximate effect of a loop. But there are no looping action types per se.

Nesting of advanced actions

You cannot build an advanced action inside another advanced action. If you need to have the same code in more than one action, you have to manually recreate the code in other clauses of the action, or copy and paste lines one at a time. It gets tedious fast.

No inheritance

This is an important programming concept that allows you to save code when creating new types of objects by appending or extending existing objects. This just doesn't exist in advanced actions. So for example, you cannot use advanced actions to define a new type of text caption or highlight box. You cannot extend an existing advanced action. Each must be standalone.

Actions are not reusable across projects

Until **Cp7** there was no way to copy actions from one project to another. If you copied and pasted slides from one project to another, all actions and variables were stripped out. (It was actually a useful way to fix a project that had become unusable due to corrupted advanced actions.) The only way to reuse any variables or advanced actions was to build them into your project template and use this template to spawn all new projects.

Now at last with **Cp7** there is a new feature called Shared Actions which we will explore in a later chapter of this e-book. It's by no means a perfect solution because in many cases it's actually quicker to rebuild the action in the new project than to use a shared action. But at least it does offer some hope for the future of advanced action reusability.

What if I want to go beyond the limitations?

With such a long list of limitations it's easy to see why hard-core programmers are unimpressed with advanced actions. They're more accustomed to unlimited creative vistas.

On balance, most Captivate users are **NOT** programmers. They've usually been thrust into the world of e-learning authoring because they needed to create or deliver training materials. Advanced actions are really designed for *non-programmers*.

But that doesn't mean serious programmers are left out in the cold. For the hard-core types, there are two main options, **ActionScript 3** and **JavaScript**.

ActionScript 3

This is the full-on object-oriented programming language from Captivate's heavy-duty cousin, Flash. You can use AS3 to create static, interactive, or question widgets that can extend the capabilities of Cp SWF output to do almost anything Flash offers. If you are already an ActionScript developer

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

interested in learning how to develop AS3 widgets for Captivate, I recommend you check out **Tristan Ward's** WidgetFactory API and his WidgetKing blog.

But remember that HTML5 devices (e.g. iPads and phones) do not usually support Flash content. So, since HTML5 is gaining ground in e-learning, you may want to consider the other alternative programming language that Cp will also accept…JavaScript.

JavaScript

Sometimes derided by programmers as "just a scripting language", JavaScript is actually based on the same international ECMAScript standard as ActionScript and many other languages. It's a loosely typed object-oriented language that is now the most popular programming language in the world, thanks in part due to the recent rise in popularity of **HTML5**.

If by some chance you've decided to create widgets for Captivate that will work in both SWF as well as **HTML5**, then you will need to become adept at both **ActionScript** and **JavaScript**. Both of these languages are beyond the scope of this e-book, however, there are many books and online resources available to learn the languages.

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

Introduction to Captivate Variables

This chapter explains Captivate variables, including what they are, the different types of variables, how to insert them, name them, and use them to display information in a project at run-time. You need to have a good understanding of variables because they are essential components of the more complex interactions you'll be creating later in this e-book.

All programming languages use variables to temporarily store values that can then be used by other parts of an application. Some programmers like to think of variables as being 'containers' or 'buckets' into which you place values such as numbers or strings of letters. However, technically variables work more like pointers or identifiers that *link* to or *reference* their values.

You don't really need to worry too much about the technical aspects. All you need to know is that variables store values and you can usually change those values using advanced actions. (The exceptions are **READ-ONLY** system variables.) In Captivate's terminology, you are said to '**assign**' the value of a variable when you use an action to change it.

Captivate variable types

Captivate has a two main types of variables and you need to become familiar with both of them in order to create some of the advanced interactivity we describe later in this book.

System variables

These variables come pre-defined with Captivate. You don't need to create them. They allow you to monitor or modify what the system (i.e. your e-learning module) is doing at run-time. The Appendix at the end of this e-book provides a comprehensive list of all system variables available with each Captivate version. (Product icons are used to indicate which Cp versions offer which system variables.) I recommend you study this list of system variables and think about the creative possibilities they offer for creating more dynamic navigation, quizzing, and interactivity.

Of the 70 or so system variables available in Cp7, only a dozen or so can be assigned values via advanced actions. The rest are **READ-ONLY** at run-time (as indicated in the appendix list mentioned above). Additionally, you cannot change any names of system variables, and since all variable names must be unique, you cannot use the name assigned for a system variable as the name of a user variable.

User variables

These are variables that you, the Cp author, can create and name as you choose...as long as you don't contravene certain variable naming rules. Furthermore, any name you assign to a user variable must not conflict with Captivate's internal functions, exposed system variables, or other hidden variables.

Hidden variables

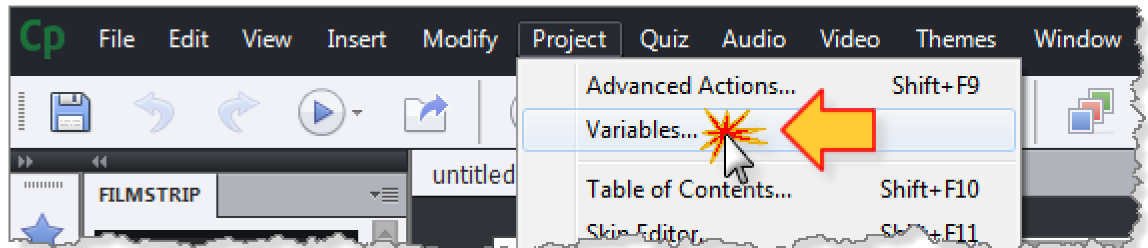
There are of course hundreds of other hidden variables that Captivate uses internally, but you might as well forget about trying to use these in your e-learning. From time to time some Cp developer will stumble across one of these by accident, usually as a result of accidentally naming a user variable with one of the 150 or so reserved keywords. The reason Adobe doesn't publish lists of these hidden variables is that it might encourage technically savvy e-learning developers to try using them and then inevitably break some essential run-time function, or even corrupt their project file entirely.

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

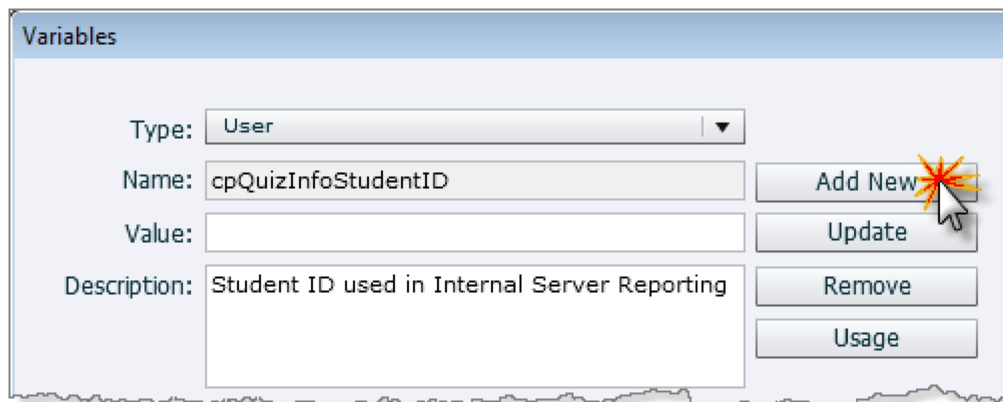
Create user variables

Fortunately, one of the easiest tasks you'll ever perform in Captivate is to create a new user variable, and if you intend to be creating complex interactivity, you'll be creating lots of variables!

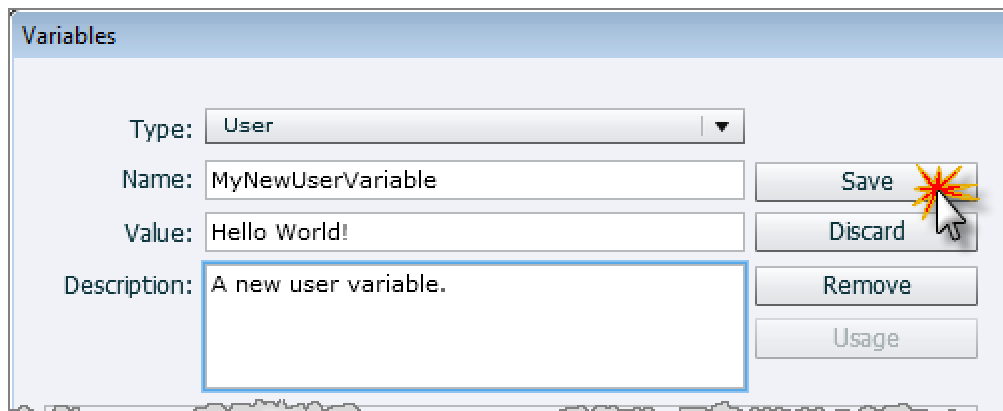
1. From the **Project** main menu, select the **Variables** option.



2. When the **Variables** dialog opens, click the **Add New** button. (The screen below is from **Cp7.1**, but except for the extra Usage button all versions since **Cp5** are much the same.)



3. Type the name of your user variable, enter a default value (if applicable) and provide a description of its purpose, then click the **Save** button.



Your newly-created variable will now be listed in the lower part of the **Variables** dialog. Variables are listed in alphabetical order, with only two exceptions. **Cp7** users will find two new user variables called **cpQuizInfoStudentID** (used for **Internal Server Reporting**) and **cpQuizInfoStudentName** (used for capturing the student name variable from a learning management system). For some reason, Adobe has opted to have these variables always sort to the bottom of the list of variables.

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

Naming your user variables

Naming Cp user variables is a bit like naming your children. It's worth taking some time to consider the ramifications of the moniker you assign, because once it goes on the birth certificate and the ink is dry it can be a major issue to change at a later date.

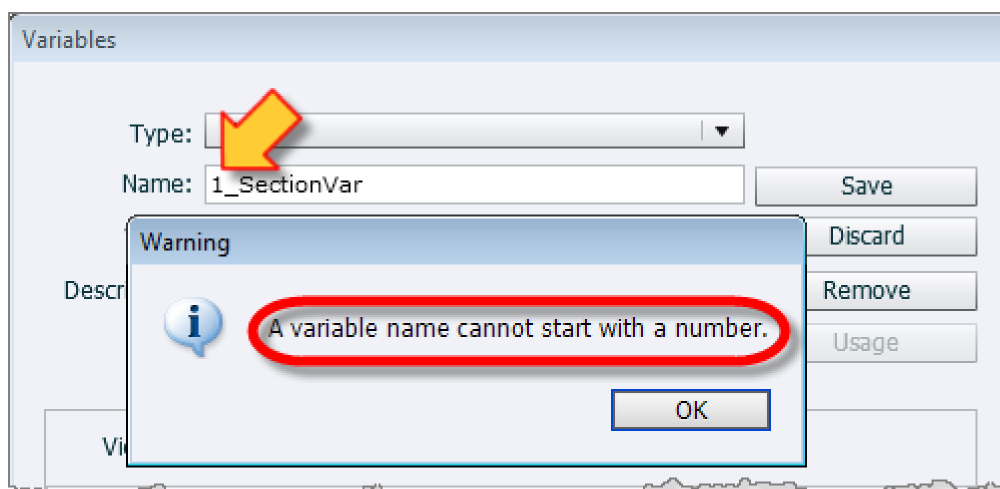
Similarly, whenever you create a new user variable you should pause and think carefully about the name you give it because once you assign a name to a user variable and save the new variable from then on ***you cannot change the name***. You're stuck with it.

If you later decide the name was a mistake, your only option is to create another variable with the correct name and then go through all the places (captions, advanced actions, etc) where you used the previous variable and reassign them to the new one. That's more trouble than you want, believe me.

So here are some basic rules to follow when naming user variables:

You cannot begin a variable name with a number or underscore character

If you try to do so, Captivate will warn you that this is not allowed (as shown below).



Some Cp authors like to begin all user variables with the letters **V** or **v_** so that variables can more easily be identified in lists. Just remember that variables are sorted alphabetically in the dialog list.

NEVER use reserved keywords

There are around **150 ActionScript keywords** that Captivate reserves for its own internal use and you must **never** assign any of these as the name of a user variable.

If you don't believe me, check out this page on the Captivate help system. To save you looking it up, I've copied the list of reserved keywords here in a special appendix at the end of this book.

I recommend you print out this list and keep it pinned up somewhere near your desk (if your office clean-desk policy doesn't prohibit such practicalities) to avoid the angst that often bedevils Cp authors that have inadvertently transgressed into reserved keyword territory.

Although in theory Cp *should* detect and *prevent* you from using reserved keywords as variable names, it doesn't always seem to do this flawlessly. If you should inadvertently happen to fall into this trap, it could prevent you from being able to publish your project, throw error messages, or in a worst-case scenario, corrupt your project and render it unusable.

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

Names must be unique

All variable names must be unique across any single CPTX project file. You can however use the same variable names in *different* modules of a course. In fact, you may need to do this when persisting variable values from one module to another. (More about this later…)

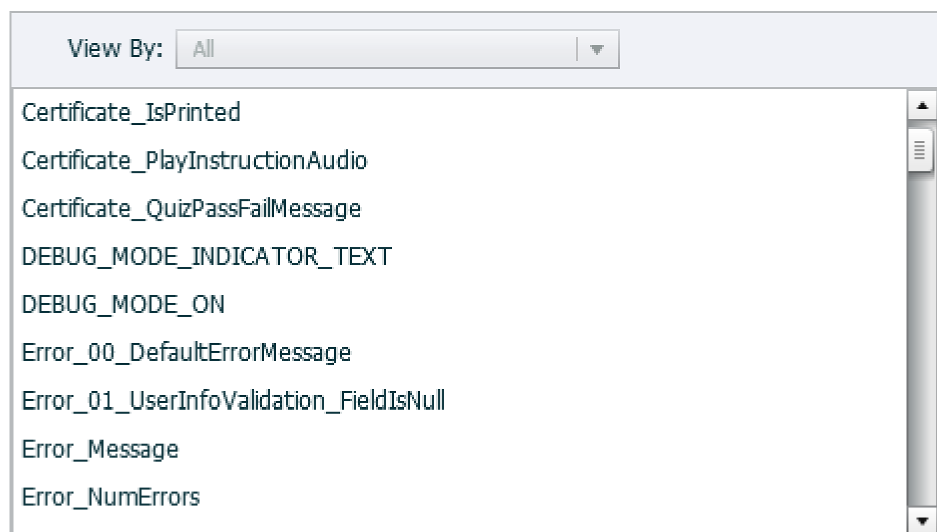
UseCamelCaseNames

CamelCase is a naming convention often used by software programmers. Words in the name all start with a capital letter, thus making the capital letters stand out like the humps on a camel. It's not mandatory, but it is popular with professional programmers because it makes the words in a variable name easier to read at a glance and understand. Look at the example below and ask yourself which of the names is easier to read quickly:

- **myreallylonginvolvedvariablename**
- **MyReallyLongInvolvedVariableName**

Names should be intuitive

Once you really get going with Cp interactivity it's quite likely you'll be creating dozens, scores, or even hundreds of variables in a single CPTX project file. (My personal record is just under 350 variables in a single module.) So it's wise to give some careful thought to how you might structure your variable names so that they will still make sense several years from now when you (or someone else) might need to come back and edit or update your course. It's wise to avoid names that will only make sense to you and nobody else.



Another suggestion you might follow is to use a prefix on your names that tells you what overall part of the interaction each variable belongs to.

For example, if you have a number of variables that are all used in the course navigation, you might begin all of their names with the word **Nav_** as follows:

- **Nav_NextButtonState**
- **Nav_BackButtonState**
- **Nav_TOCItem**

If you have a number of variables involved in a specific quiz interaction, then you may use a prefix such as **Quiz_** or similar. Take a look at the screenshot below of part of the **Variables** dialog in one of my own projects. Can you tell from the names what each variable might be used for?

Whatever naming convention you follow, the point is that each variable name should be intuitive enough to make its purpose obvious without the next developer being required to laboriously trawl through the **Variables** dialog looking for descriptions. And while we're on the subject…

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

Always add descriptions

Even though you go to great lengths to make your variable names self-explanatory, it's still good to include text in the **Description** field to document what the variable will be doing or where it might be used. For example, if a particular variable will be supplying essential data to one or more advanced actions or conditional advanced actions, then you might list them in the description.

About variable data typing

In most programming languages variables are 'typed', meaning that they are deliberately restricted to accept and store only certain types of values such as Booleans (0 or 1, true or false, yes or no), integers (numbers), or strings (text).

By contrast, Cp user variables are not typed. They'll accept almost any numeric or text value you want to throw at them. (As mentioned before, system variables are mostly READ-ONLY so their data is usually also of a fixed type.)

Although on the surface this freedom to store any type of value in a variable might seem like a good thing, it can also sometimes get you into trouble. For example, you may end up inadvertently trying to perform math functions on a string of text, or concatenating a number, which of course won't work.

So this means the onus is on you, the Cp developer, to keep track of what type of data your variables should hold. And that's a lot easier to do if you keep your documentation and descriptions up to date.

Editing user variables

Once you've created a number of user variables from time to time you'll need to change something. Since you cannot change the variable's name, your editing options are limited to altering the default value, description text, or removing/deleting the variable altogether.

1. Open the **Variables** dialog and select the variable you want to change. (The **Update** button is initially disabled until you make some detectable change to the variable properties.)
2. Edit the **Value** or **Description** fields as needed and then click the **Update** button.

(After making the change, the **Update** button again becomes inactive)

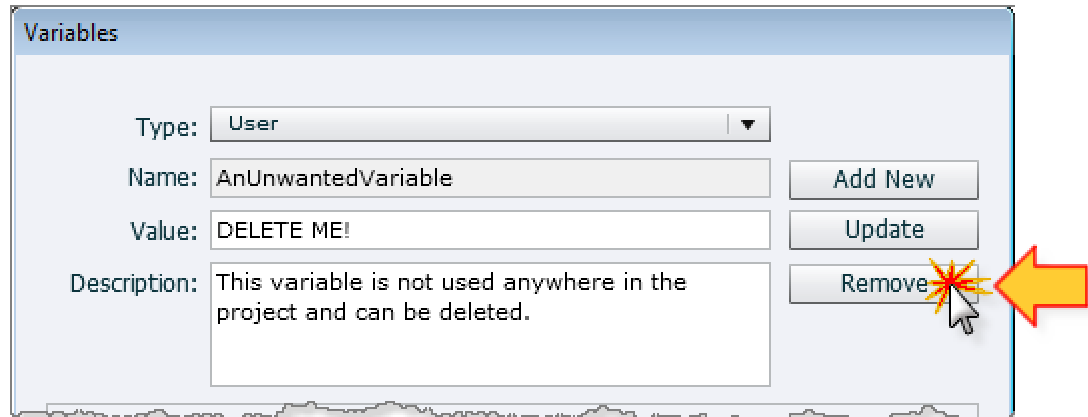
Removing user variables

As you develop a project you may find that some of the variables you create become redundant and need to be removed (perhaps because you had to create a duplicate variable with a different name). It's a good idea to take out the trash by deleting these unwanted variables because they can waste browser memory and slow down run-time performance.

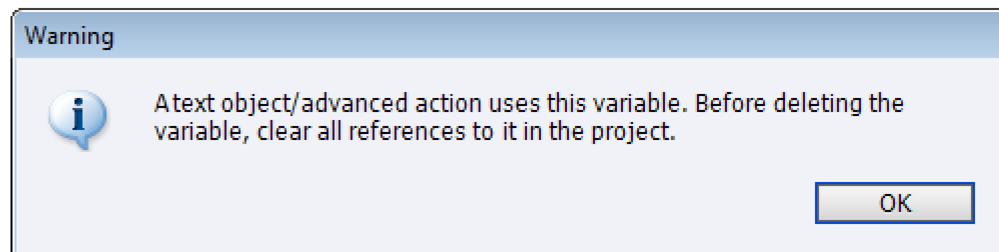
© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

The process *seems* simple enough…but there are some caveats:

1. Open the **Variables** dialog and select the variable to be deleted.
2. Click the **Remove** button.



3. Now things get a little more complex. Please note the following caveats about removing variables:
   - You can only remove user variables, not system variables.
   - You *can* remove a variable referenced by a *simple* action that assigns its value, but this will result in the action being reset to **Continue**.
   - You can only remove user variables that are **NOT** currently being used to display values in screen objects such as text captions or shapes. Additionally, you cannot remove any variables currently referenced by **Single**, **Standard** or **Conditional Actions** in your project file. If you try to remove one of these referenced variables, Captivate will show the following warning message.



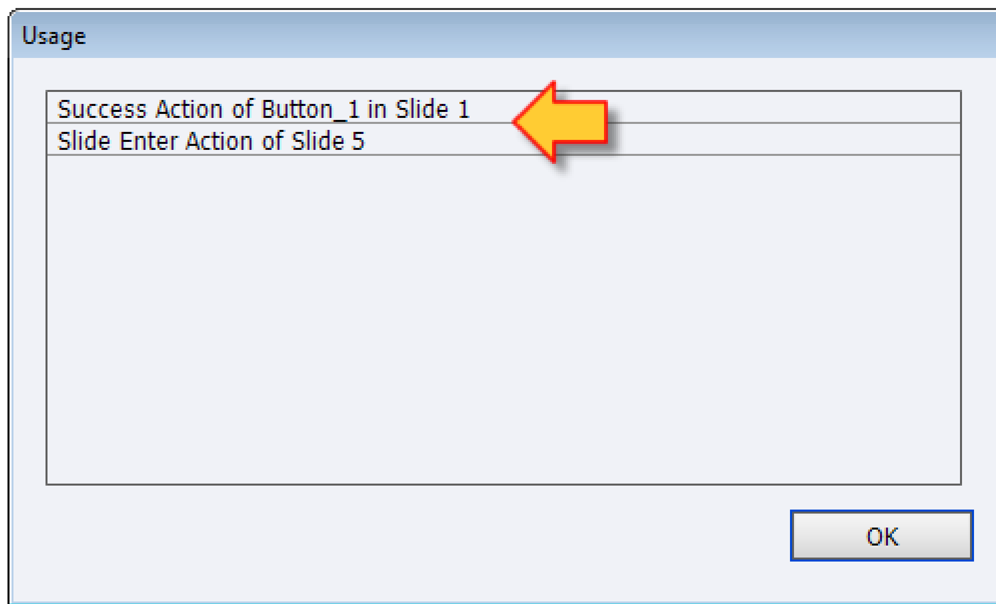
Unfortunately, this warning message doesn't give you any information about *which specific slides or advanced actions* use the variable and need to be modified before you can remove it. In a large project with many slides and actions, this can be like trying to find a needle in a digital haystack

Tracing variable usage in Cp7.1

The inability to trace where user variables were referenced in a project file was a huge weakness in Captivate functionality that was finally addressed with the **Cp7.0.1 update patch** released in mid November 2013.

The **Variables** dialog now has a **Usage** button (similar to the one on the **Advanced Actions** dialog) that when clicked will show where the currently selected variable is referenced on any slide in the project, as well as which specific run-time event is used to execute the action involved.

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED



This now makes it extremely easy to do any required project file housekeeping. However, there are some exceptions. Variables referenced in text-entry boxes, widgets and interactions (which are also widgets) don't get taken into account by the **Usage** dialog. So if you have a lot of widgets or interactions in your project, you may need to check them manually before starting any round of mass deletions. If your variable is not used anywhere, you'll just see a dialog that says: **Item is not in use**.

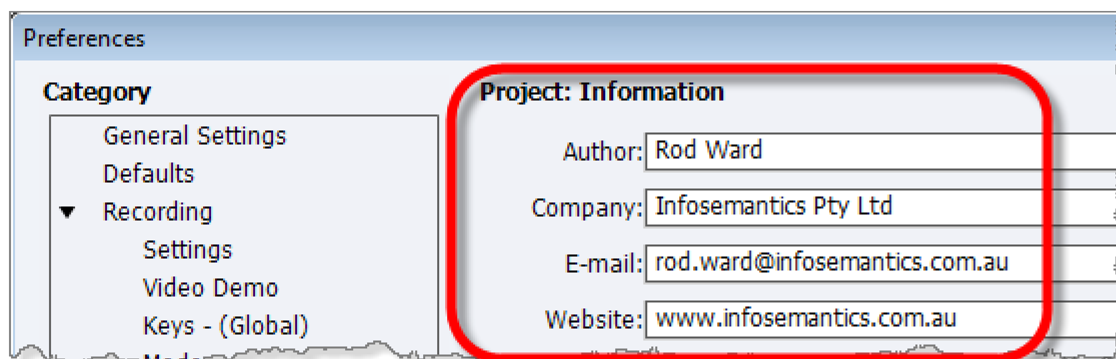
Displaying variables on slides at run-time

One of the easiest ways to make use of variables is to display values in text captions, shapes, and Smart Shape objects on slides at run-time. In the example below I show how this is done by inserting several display fields into a single text caption.

How to display project information

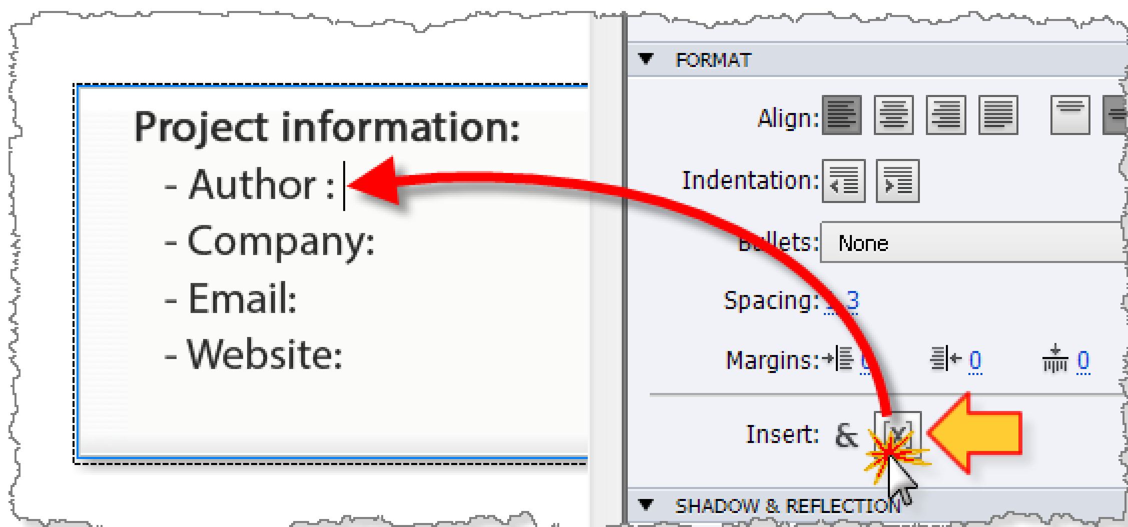
Let's imagine your project needs to have the e-learning author's name and contact information appear on a slide at run-time. Here's how you could do that:

1. Create a new blank project and open the **File > Project Info…** (This is quicker than navigating to the same dialog location using **Edit > Preferences > Project > Information**.)
2. Type relevant information into the fields provided.

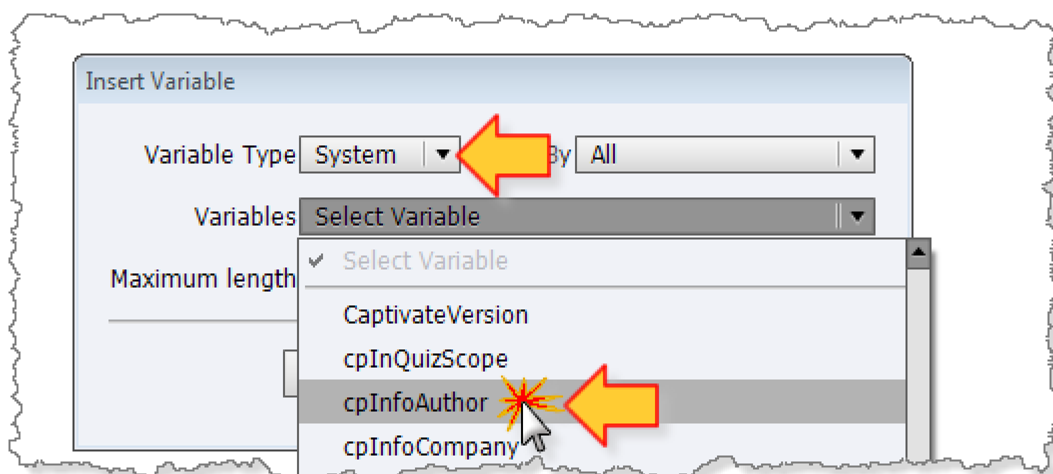


© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

3. On another slide in your project file, insert a text caption and some text to indicate the type of information it provides.
4. Place your mouse cursor at the point in the text caption where you want to insert the variable display field.
5. On the **Properties tab > Format** accordion, click the **Insert Variable** icon.



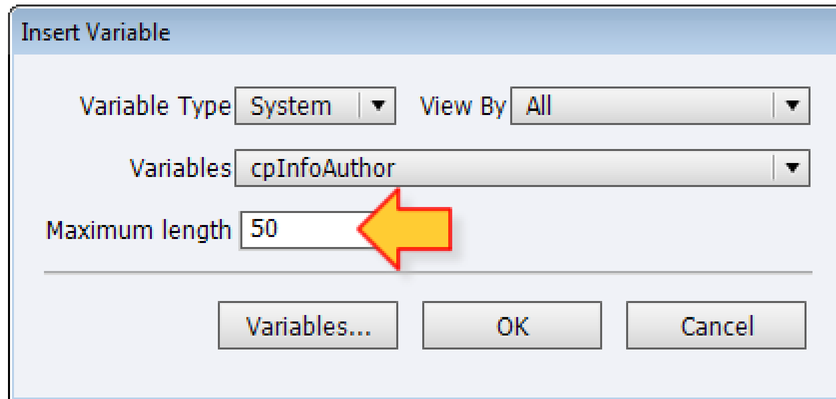
6. When the **Insert Variable** dialog opens, change the **Variable Type** to **System** and from the **Variables** drop-down list select **cpInfoAuthor** as the variable.



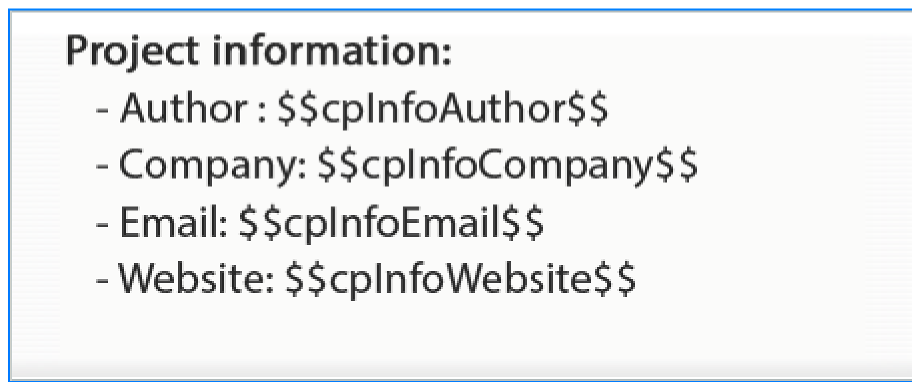
7. Pay close attention to the default value shown in the **Maximum length** field. It has gotchas!
   - **Gotcha # 1**: In earlier versions of Cp this field was set to **15 characters** by default. This length is fine if all you are doing is displaying a number, identifier code, or slide name. But it's usually too short to display personal names, company names, course names, etc. In Cp7 the default length has been increased to 50 characters, but you can change it to specify any length up to about 250 characters. So give careful thought to how much information you need to display and set the field length accordingly.
   - **Gotcha # 2**: There is a bug in versions prior to Cp7 where if you happen to be inserting a number of variables one after another, the **Maximum Length** field randomly reverts to a value of **0**. So be sure to take a moment to check this field before hitting **OK**.

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

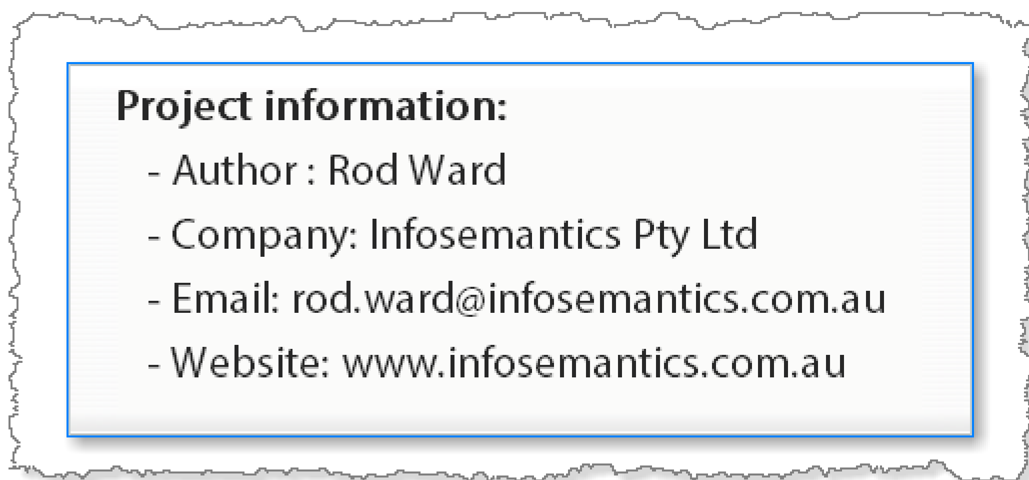
- **Gotcha # 3**: If you happen to guess wrong about the required number of characters to allow as the length of the display field, you cannot easily change it. You must *delete* the inserted field and *re-insert* another one to change the display length. So it's usually wise to bet a little on the high side to be safe.



8. Once you're happy with the details of the inserted variable, click the **OK** button. Your variable will appear by name, surrounded by double dollar signs (e.g. **$$cpInfoAuthor$$**).
9. Repeat the last few steps to insert any other variables in the same caption.



10. Publish your project to see how the variables are displayed in the caption.



© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

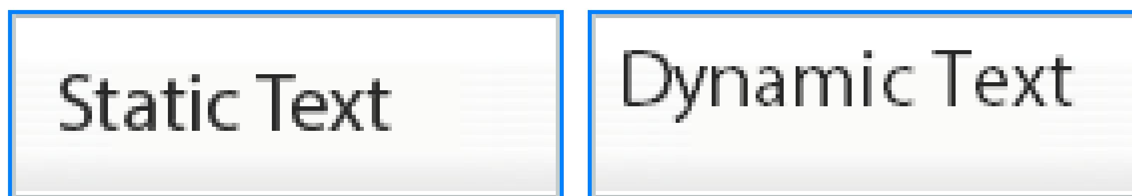
Why not just type the variable name with $$ characters on either end?

Yes you can insert a variable into a caption or shape by just typing the variable name and adding $$ characters at the beginning and end. However, while it may be slightly faster, the downside to this method it that it doesn't allow you to set the number of characters for the maximum display length. It will just assign the default amount, which is only 15 characters for Cp versions 5 through to 6.1 and 50 characters for Cp7. The 15 character default was far too small, and even 50 characters will sometimes be insufficient. So, call me a control freak, but I still prefer to insert variables via the dialog method and set the display length myself.

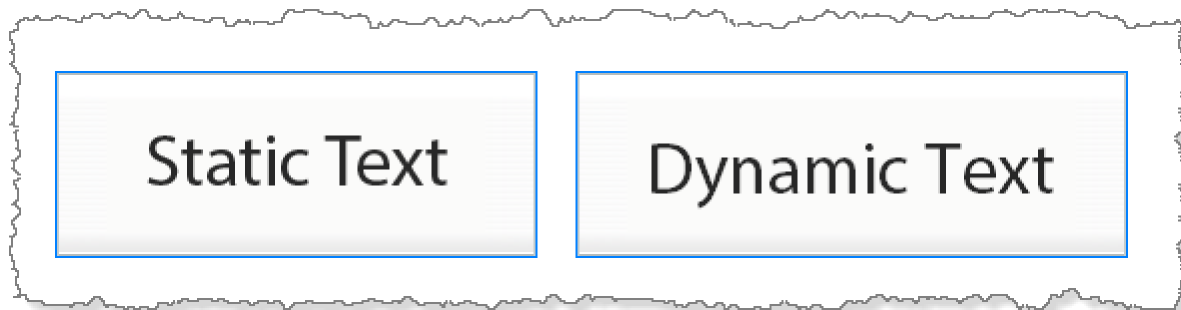
Overcoming dynamic text formatting issues

When you insert a variable into a caption or shape, don't be surprised if you need to fiddle around with the formatting a bit before it all looks right at run-time. These formatting issues are due to the fact that normal caption text is '*static* text' (because it doesn't change at run-time) while variable display text is 'dynamic text' (because it can change depending on the current value of the variable being displayed). In Captivate's published output, captions containing *static* text are actually converted into images. But since this means the text becomes an image too, this approach would not allow dynamic text to change on-the-fly at run-time. So, static and dynamic text strings are kept separate. The result can be certain inconsistencies in visual formatting.

For example, some early versions of Captivate did not allow dynamic text to be indented. If you applied indenting in Edit mode, the indenting disappeared at runtime. If you added variable text into the middle of a paragraph of static text in a caption, none of that text would honor the applied formatting. This often resulted in captions with text unattractively crammed into the top-left corner or positioned hard against the left side of the caption. There were also notable differences in the aliasing of the text. For example note the differences in the two run-time captions below from **Cp 5.5**.



The text in the caption above left is just typed in. The one on the right is displaying text supplied by a user variable. The workaround for such formatting issues is to place dynamic text inside a transparent caption positioned *on a layer above* another caption. This made it appear as if the underlying caption contained the dynamic text when in fact it was just acting as a background. Fortunately, these issues have been steadily diminishing with each new version and it's now quite difficult to tell static and dynamic text apart as you can see from the **Cp7** examples shown below.



Hopefully one day, in some future Captivate version, there will be no discernible differences at all.

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

Boolean variables

One of the most common types of variables that you will need to use when working with advanced actions is the **Boolean** variable. This data type was named after George Boole, a mathematician who developed Boolean Algebra way back in the 1850's, well before computers were ever thought of.

Unlike most variables that can hold any number of possible numeric or string (text) values, a Boolean variable can only hold one of two values. These are usually 0 or 1, TRUE or FALSE, YES or NO. A Boolean variable acts a lot like a light switch. The light is either ON or OFF.

You can set the value of a Boolean variable using the assign action, either as part of a single action, standard action, or conditional action. You'll find Boolean variables particularly useful in conditional actions to store the result of a condition that checks whether something is correct or incorrect.

Migrating variables from one project to another

Captivate has traditionally had no way to move variables or advanced actions from one project file to another. This deficiency has always been a huge hole in the application's otherwise fine functionality.

But then **Captivate 7.0** added the ability to use shared actions, with ability to export actions from one project and then import the same actions into another. One less-known feature of shared action functionality was the fact that all variables referenced by the shared action are migrated along with it and recreated in the new project file location at the time of import.

With the release of the update patch for Captivate 7.0.1 in mid November 2013 Adobe went one step further to allow any variables referenced by actions on a *copied slide* to be added to any project this slide was *pasted* into. This certainly helps.

However, there is a catch. If the target project already contains variables of the same, the incoming variables are renamed with a number appended to the end. You can see the messy end result here in the chapter about shared actions. So watch out for that issue.

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

Understanding Captivate's Run-time Events

*In this chapter we introduce the concept of **events**. Although you don't need **events** in order to use Captivate **variables**, you cannot use **actions** without using **events** to trigger or 'execute' them. So a good understanding of events is essential to mastering advanced actions. There are only about a dozen events to choose from. But as you read this chapter you may be surprised at how complex some of event scenarios can become.*

Events are points at run-time of an e-learning module where something happens or changes. In the software programming world, events usually have names beginning with the word **On**.

For example, look at these events and their meanings:

- **OnClick** – something got clicked;
- **OnMouseOver** – the user placed their mouse cursor over something on screen;
- **OnMouseOut** – the user moved their mouse cursor away from something on screen;
- **OnKeyEnter** – the user hit a key on the keyboard.

In the software world events are used to call functions, i.e. do stuff. In Captivate, events are used to execute **actions** in your published content at run-time, i.e. do stuff.

Captivate's run-time events have slightly different names to those shown above, but the same general concepts apply. They usually start with the word **On** (though there are some exceptions), and the name of the event gives you some idea about what happened or what changed to 'register' the event.

We'll first consider events exposed by different types of slides and then look at events from objects present on the slides. Some Cp objects have no events, while others can have several.

Slide events

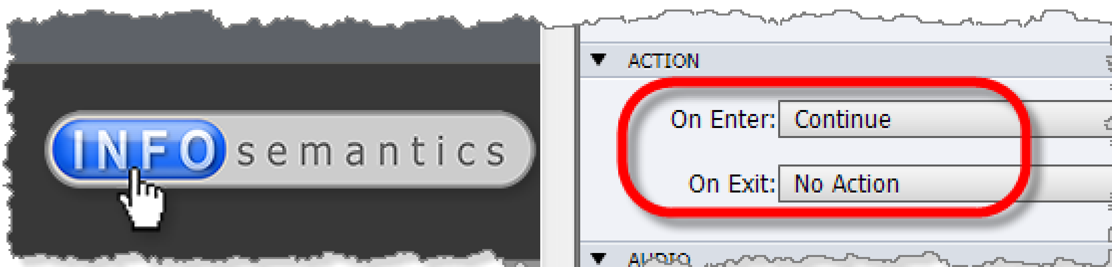
Master slides are the only Cp slide type that **doesn't** offer events to execute actions.

Normal slides

All normal slides have the following two events:

- **On Enter** – This occurs when the slide enters the timeline at run-time.
- **On Exit** – This event fires *after* the slide exits the timeline at run-time. It's important to note this fact because it means any actions triggered happen on the *next* slide, not this one.

These events are clearly visible when you select a slide in the filmstrip and then look at the **Properties tab > Action** accordion. The drop down menus beside these events show the currently selected action each event will execute.



© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

Avoid using the On Exit event

*You may have noted that the example above has no action set for the **On Exit** event of this slide. There's actually good reason for this. The **On Exit** event is only fired after the slide reaches the **very last frame** on its timeline. In many cases this frame is never reached because your user interacts with some object (e.g. button, playbar, or TOC item) that navigates them immediately to another slide in the project. This would mean the **On Exit** action would never get executed.*

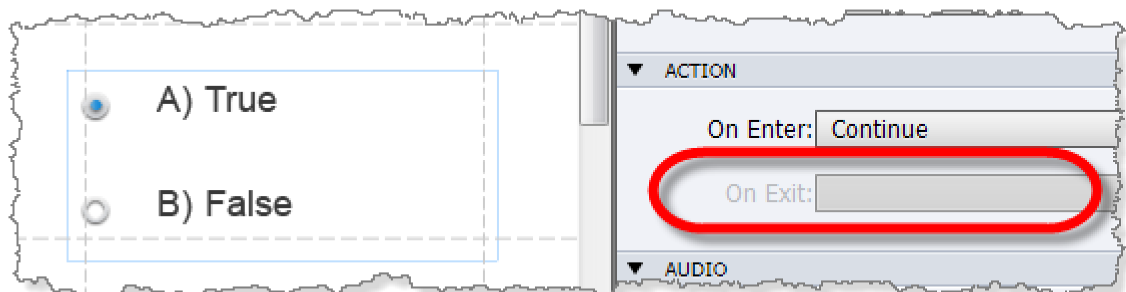
*So, since you cannot always predict how users will navigate your course, it's usually best not to use **On Exit** for anything critical. If you set the **On Exit** event to **No Action**, the slide will still progress to the next slide when it reaches the end of its timeline.*

*The most useful slide event of all is always going to be the **On Enter** event because it **always** gets fired for each slide, no matter how the slide was called.*

Question slides

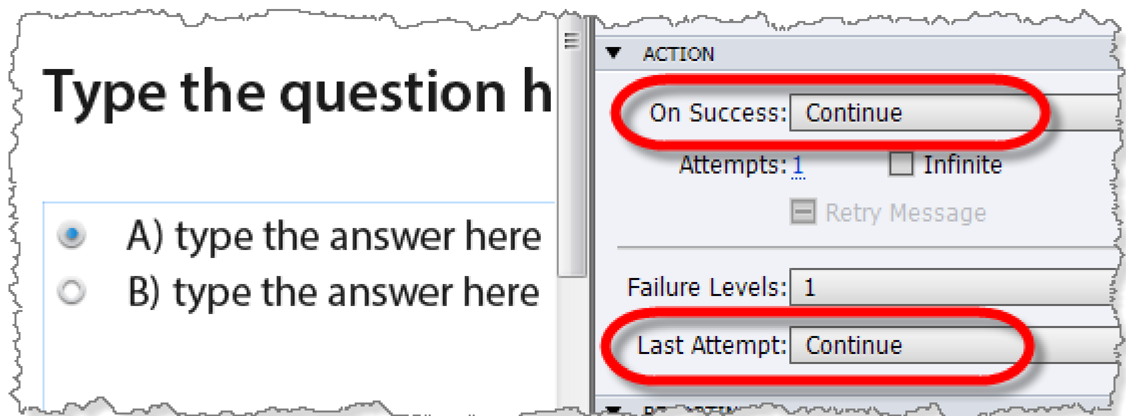
Question slides have the following events:

- **On Enter** – As with normal slides, this event occurs when the question slide enters the timeline at run-time. Question slides don't have an **On Exit** event. If you look on the question slide's **Properties tab > Action** accordion, the **On Exit** event is disabled.



This is because the **On Success** and **On Last Attempt** events actually replace the need for an **On Exit** event. The only way you're supposed to end up leaving a question slide is after either succeeding or failing to answer it correctly.

- **On Success** – This occurs when the question is correctly answered by the user.
- **On Last Attempt** – This occurs when the user has answered the question incorrectly and has no more attempts available for use.



© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

About the Multiple Choice quiz question Advanced Answer Option

While we're discussing quiz question slides, it's worth mentioning the **Multiple Answer** (radio button) quiz question type has a special trick up its sleeve that gives you yet another event you can use. It's called the **Advanced Answer Option** and it allows you to execute an action based on the *specific answer option* a user selects from the radio button array in the quiz question.

This means you could choose to play a sound, show a different feedback text box, assign a value to a user variable, or execute any number of other actions based on which answer the user selects. What you do is entirely up to your imagination and creativity.

However, don't get too excited about the **Advanced Answer Option**. It is only available with the **Multiple Choice** quiz question type and no other. You can of course fake a **True/False** quiz question by creating a **Multi-choice** question with only two answer options. But there is no way to use advanced answer options with **Multi-answer** (checkbox) quiz questions.

To get really creative with quiz questions, sometimes you need to create them from scratch yourself.

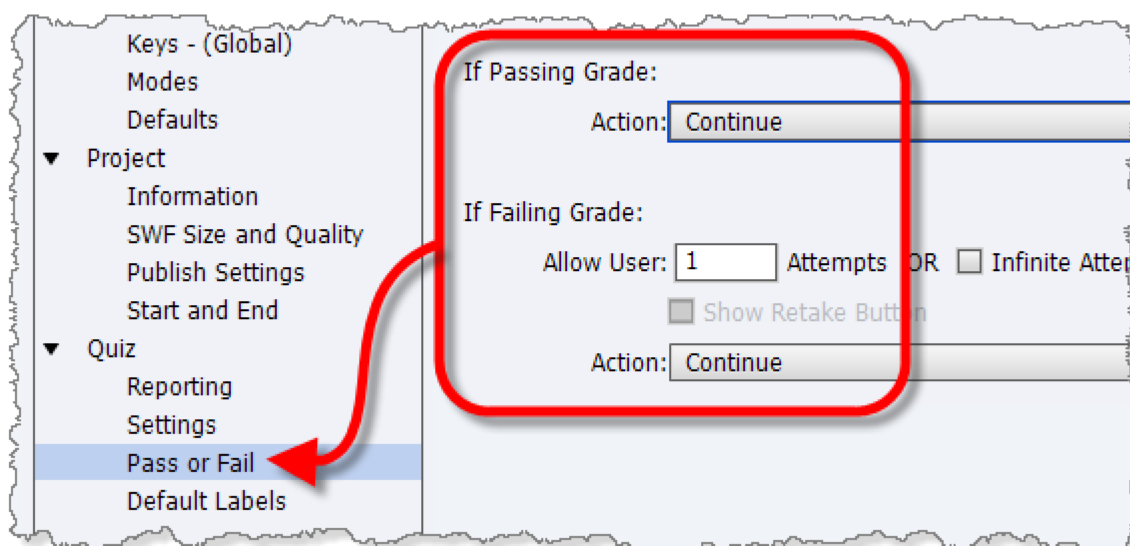
Quiz Result slides

Like question slides, the **Quiz Result** slide also has three events, but two of them are a little difficult to find because technically they are related to the quiz itself and would occur even if the **Quiz Result** slide were hidden or not present.

- **On Enter** – This occurs when the **Quiz Result** slide enters the timeline at run-time. Like quiz slides, **Quiz Result** slides also have the **On Exit** event disabled.
- **If Passing Grade** – This event occurs if the user successfully passes the quiz. It's roughly equivalent to the **On Success** action of a quiz question.
- **If Failing Grade** – This event occurs if the user fails the quiz and has no more attempts allowed. So it works in much the same way as the **On Last Attempt** event of a quiz question.

Like all of the previous slide types, the **On Enter** event action for a **Quiz Result** slide is configured from the **Properties tab > Action** accordion. But you won't find anywhere on the **Properties** or **Quiz Properties** tabs to configure actions to be triggered by these other two events, **If Passing Grade** and **If Failing Grade**. So where are they?

Well it turns out you need to go to the **Preferences > Quiz > Pass or Fail** screen to find them.

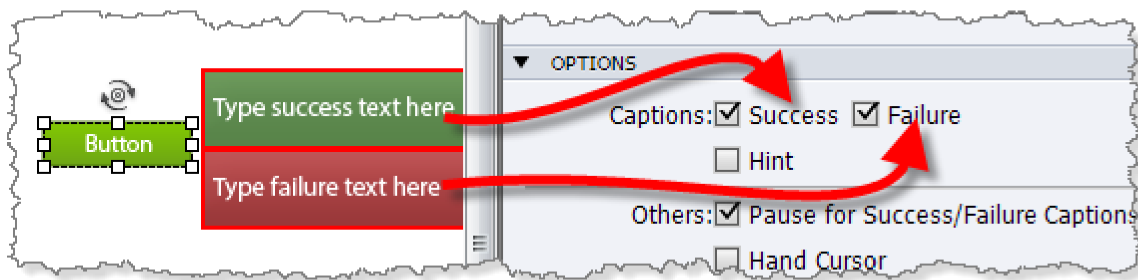


© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

However, there is something important you need to know about these two quiz events. They are only capable of triggering actions *after* the quiz has been evaluated as passed or failed. That means these events only fire *after* you click the **Continue** button on the **Quiz Results** slide. Or, if the **Quiz Results** slide is hidden, they only fire *after* exiting the final quiz question in the quiz.

Interactive object events

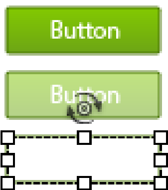
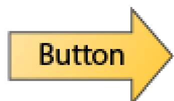
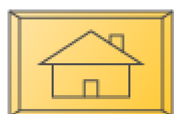
Now let's look at the events you can use with screen objects, starting with interactive objects. In Captivate's world an 'interactive object' is defined as any object that can evaluate to a **Success** or **Failure** condition. In Cp you always know when you are dealing with an interactive object because you will see options on the Properties tab to enable or disable **Success** and **Failure** captions.



Here are the main types of interactive objects you will be using to execute advanced actions:

Buttons

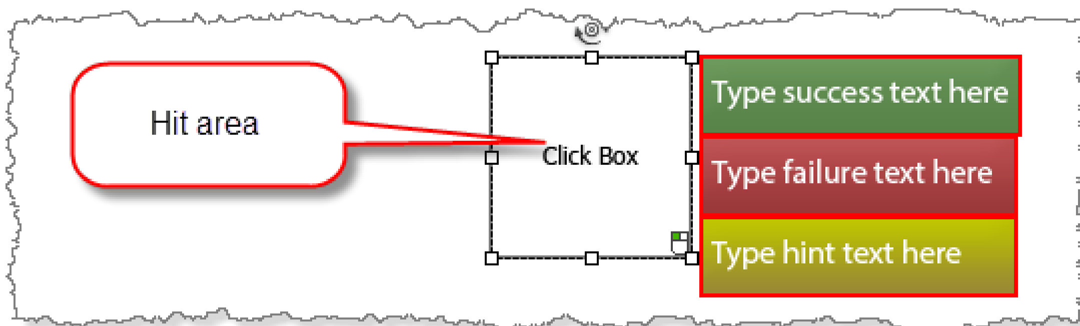
Captivate offers four button types to choose from:

	Text buttons – A simple button object with text label.
	**Transparent buttons** – So named because you can set the value of **Fill Alpha** down as low as 0% to make them invisible (like click boxes). But the way, the buttons shown at left are from **Cp6** or later versions. Those of you using **Cp5** or **5.5** will still be able to use transparent buttons but will not have as many formatting options
	Image buttons – These are actually composed of three images, one each for **Up**, **Over**, and **Down** states. The area occupied on screen by the button itself corresponds to its 'hit area'. Captivate comes with about 30 image buttons you can choose from, but you can also create your own with any decent graphics editing software application.
 	**Smart Shape buttons** (Cp6 or later only) – Captivate has had shape objects for several versions. But **Smart Shapes**, with the extra option to **Use as Button** were only added in **Cp6**. When used as a button, a **Smart Shape** becomes a clickable interactive object with its own hit area. However, unlike Cp text buttons and image buttons, **Smart Shape** buttons don't have up, over, and down 'states' to make their appearance change in response to user interaction. This is a big limitation that Adobe will hopefully address in a future version of Cp.

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

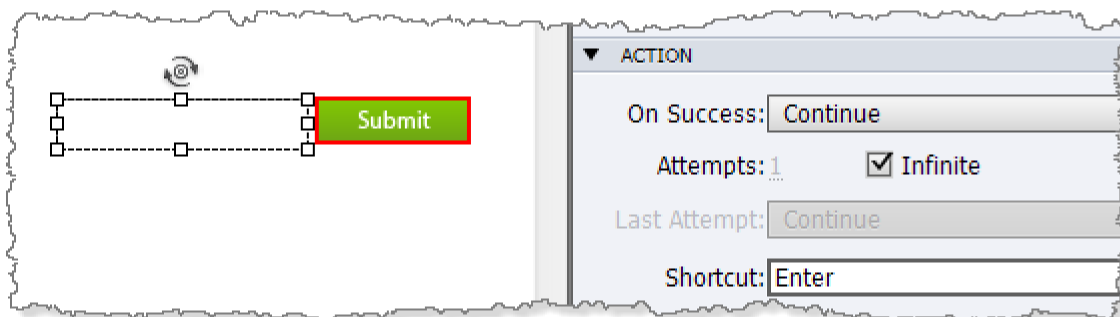
Click boxes

A click box is effectively an invisible 'hit area' that you can place over the top of other objects to make it appear as if those objects are clickable, when in reality the user is registering a mouse event via the click box instead.



Text-entry boxes (TEBs for short)

A TEB is essentially an input field that accepts numbers or text strings entered by the user, and then stores this data in an associated user variable. You need to turn on **Validation** in order to have the user input checked against one or more correct answers so as to register **Success** or **Failure** events.



TEBs are very versatile and useful interactive objects, but they also have a few extra wrinkles you need to be aware of. So we'll be giving them more in-depth attention in a later chapter about Accepting User Input.

Interactive widgets

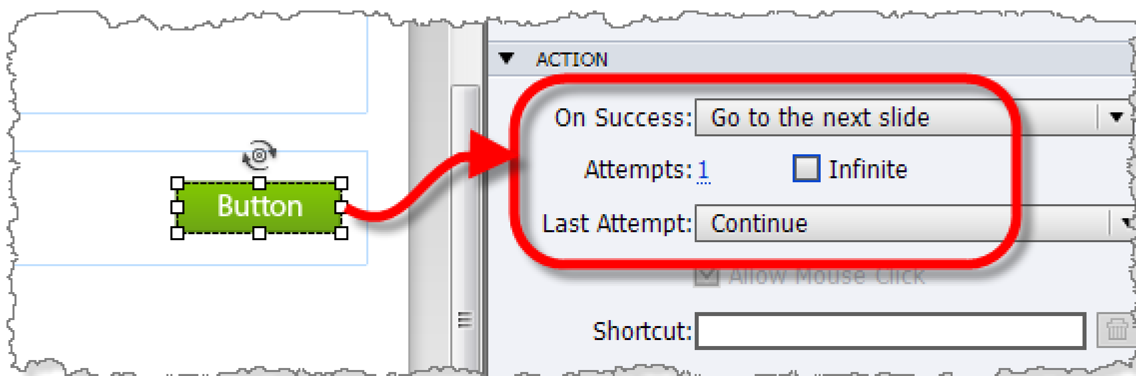
These are small pieces of software written in **ActionScript 3** and possibly also JavaScript (if designed to be compatible with **HTML5** output). Widgets extend the capabilities of Captivate beyond what it could do out of the box. But they are hard to pin down and describe because they may potentially be *any* kind of interaction the widget designer/developer can imagine and is clever enough to create. They are classed as interactive objects because they can also evaluate to either a **Success** or **Failure** condition and thereby trigger actions. However, exactly what constitutes success or failure is really up to the widget developer to decide.

For example, the Infosemantics Event Handler widget allows you to turn any Cp screen object into an interactive object that can trigger success or failure events when clicked, double-clicked, right-clicked, when rolled over with the mouse cursor, or when the mouse cursor is moved away. In the case of the Infosemantics Interactive Drag and Drop widget, a success condition is registered if the user drags and drops objects on top of their correct matching targets. These are just a couple of examples. If you want to see more, check out this page on the Infosemantics website.

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

Run-time events offered by interactive objects

All interactive objects have at least two events that can be used to execute actions. These events are clearly shown if you select an interactive object and then go to the **Properties tab > Action** accordion. For example, the screenshot below shows the events available for a button.



On Success

This event occurs when the object evaluates to a **success** condition.

- In the case of a **button**, **click box** or **Smart Shape** used as a button, clicking *within* the object's hit area is all that is required to register success.
- For a **TEB**, *if **Validation** is not turned on*, success may simply mean the user has typed something into the field provided and clicked the **Submit** button or hit the **ENTER** key on their keyboard. However, *if **Validation** is also turned on*, then success means the user's input was checked and evaluated to be a match to one of the pre-configured correct answers. (These subtle distinctions are the reason I mentioned before that we need to pay more attention to TEBs in a later chapter.)
- For an interactive widget, success is whatever the widget developer designed it to be. For example, if the widget is a drag and drop interaction, then success would be achieved by dropping objects on their correct targets. With widgets the possibilities are almost endless.

On Last Attempt

This event occurs when the object evaluates to a **failure** condition **AND the user has no more attempts remaining**. Take special note of the last part of that sentence about remaining attempts because this is an area of confusion for many Cp authors that I explain a little later in this chapter. If the user still has *any more attempts remaining*, then as far as Captivate is concerned, they haven't as yet *failed* the interaction, so no failure condition is registered, and there is no usable event.

Here are some examples of failure conditions for different types of objects if the user has flunked their last allowed attempt:

- For **buttons** or **click box** objects, failure means you did not click within the object's hit area.
- For a text-entry box, failure means the number or text you typed into the field did not match any one of the specified correct answers.
- For a drag and drop widget, failure means you did not drop objects onto their correct targets.

The On Focus Lost event

Text-entry boxes are something of a special case when it comes to interactive objects. Not only do

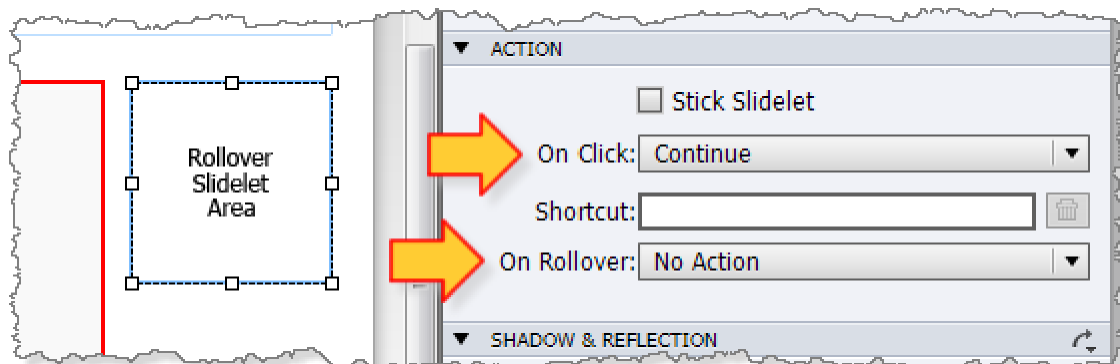
© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

they have the usual **On Success** and **On Last Attempt** actions, they also offer one more called **On Focus Lost**. This event gets registered if the user has entered some text into the text entry field and then clicks somewhere outside the field, thereby shifting the focus elsewhere on the slide.

On Focus Lost can be a very useful event when capturing user input from a number of TEBs if you need to know exactly which field the user has just interacted with, or you want to trigger advanced actions to perform more complex validation of entered data. We discuss this concept more in the chapter about Capturing User Input, especially the subheading about using TEBs for that purpose.

Rollover slidelet events

Technically-speaking, slidelets are not actually interactive objects because they do not register success and failure conditions. However, they do have a couple of events that may prove useful:



- **On Click** – As the name suggests, this event is registered when you click on the slidelet hit area. This enables slidelets to trigger events in a similar way to click boxes. So you could use a slidelet to create an interaction where rolling over its hit area displayed information (in the visible part of the slidelet) and then allow the user to click the area where their mouse is currently located to jump to another slide for more extensive information.
- **On Rollover** – This event is the only native *rollover* event in any native Captivate object that can be used to trigger actions. Almost all other mouse events are some variation of a click (left-click, right-click, double-click). My personal preference is to avoid using slidelets for anything, so I usually resort to Event Handler Widgets for triggering actions on mouse over events. But if you don't have these widgets then the slidelet might offer a viable alternative.

Why I don't recommend using slidelets

Rollover slidelets have been in Captivate now for several versions, but let me be quite candid in saying that I am definitely **NOT** a fan of using this particular object to build complex interactivity. Slidelets seem to have too many inconsistencies and bugs for my liking. I've also found that having any more than one or two on a slide can compromise run-time performance. Additionally, slidelets are not supported in **HTML5** output. So if your target audience is users of mobile computing, slidelets are not an option.

Drag and drop events

Captivate **6.1** introduced a new feature called **Drag and Drop** which caused a lot of excitement in the community. Prior to this enhancement the only way to have true drag and drop interactions was to buy one of the Infosemantics Drag and Drop widgets. (In fact, these widgets are still the only way that Cp users on versions **5.0** through to **6.0.1** can use drag and drop.)

One of the great things about Captivate's new drag and drop capability is that it also gives you some

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

more events to play with.

On Success

This event is registered when you drop all of the correct objects on their respective targets and the answer combination is submitted. You might have achieved this by clicking the Submit button provided, but you can also use the **Auto Submit** option as well.

On Last Attempt

As with all the other instances where we've seen **On Last Attempt** mentioned, this one is fired when your participant fails to achieve a success condition and they have no more attempts left. In drag and drop interactions, this means they haven't correctly dropped all the right objects onto their correct targets.

Like to know more about creating Drag and Drop interactions?

Drag and drop is in fact a huge topic on its own. So in order to keep this book about advanced actions within a reasonable size we won't be delving deeply into drag and drop interactions here. However, stay tuned! I have another e-book all about drag and drop in planning stages. Contact me if you'd like to register your interest in that book.

How to decide which objects and events to use

Now that you know about most of the available events, perhaps the big question in your mind is: *"With a dozen or more to choose from, how will I know which event to use and when?"*

Unfortunately this requires one of those maddening "*it depends*" answers. Which event you use in a given situation *depends* on two main factors:

1. What specific *results* are trying to achieve with your interaction design?
2. What kind of *objects* do you need to use to achieve that result? (This is because, as you saw above, different **objects** offer different run-time **events**.)

Event scenarios

Perhaps this whole event decision-making process is best explained by considering some examples of typical e-learning scenarios for which you need to create interactive solutions.

Let's say you need to trigger an action **when…**

…the right thing gets clicked

This is easy enough to achieve using a **click box**, **button**, **Smart Shape** or **interactive widget**. So, you can use the **On Success** event of those objects to execute a required action.

…the wrong thing gets clicked

You can use a **button** or **click box** set to only *one* allowed attempt, and use the **On Last Attempt** event. When the user clicks anywhere *outside* the hit area, they've immediately used up all allowed attempts and this will execute the action you specify.

But what if you need to allow multiple attempts on the correct area (perhaps because the user needs to click something repeatedly) but immediately detect any clicks outside this hit area? In that case you can set up a click box on a layer underneath the button and have the hit area of the click box cover the slide. The click box's **On Success** event can then be used to trigger an action in response to this 'failure'. (Since the correct button object is on a higher slide layer, if the user clicks the button, its hit area will register the hit instead of the click box underneath.)

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

See also my clarification of what Captivate really means by the term '**Infinite Attempts**'.

…the participant enters a correct answer into a text field

Use a **Text-entry Box** object, turn on **Validation** and specify one or more correct answers. When the user clicks the **TEB Submit** button or hits **ENTER** on the keyboard, Cp will evaluate the data found in the entry field to see if it matches any of the specified correct answers. If it does, then an **On Success** event is registered.

You can also use the **Short Answer** or **Fill-in-the-blanks** quiz question types to check user input for correct answers. However, there are some special wrinkles to be aware of with quiz questions. See the scenarios below about triggering actions when the participant answers a question correctly or fails to answer correctly.

…the participant rolls their mouse cursor over an object

Triggering actions from mouseover events in Captivate is not currently as easy as it should be. The only native Captivate object currently capable of registering rollover events is the **Rollover Slidelet** with its **On Rollover** event. (Captivate's **Rollover Caption** and **Rollover Image** objects are not interactive and therefore cannot trigger actions.) However, as I've already mentioned, I don't recommend using slidelets for various reasons.

An alternative solution is the Infosemantics Event Handler widget. It enables you to turn any Captivate object into an interactive object that can trigger events on **Roll Over** and **Roll Out** (as well as several other mouse events). However, the slidelet object and the **Event Handler** widget are both incompatible with HTML5 output at this time.

…the participant moves their mouse cursor away from an object

Captivate has no native object that is designed to register events on mouse out or rollout. Again, the only alternative solution is the Infosemantics Event Handler widget because it has a **Roll Out** event.

…the participant drags an object somewhere

This one should be obvious. You need to use the **On Success** or **On Last Attempt** events of either Captivate's native drag and drop functionality, or a drag and drop widget.

…as soon as a slide is played

In this scenario your best option is to use the **On Enter** slide event. The **On Exit** event of the previous slide is not a good choice in this situation because it only gets fired when the *final frame* of the slide is reached. That event may not fire in some cases, for example, if the user clicks something that jumps them to a different slide. Therefore, using the **On Enter** event of the slide where you want something to happen is a better choice because it will *always* occur no matter how the slide was reached.

I use lots of **On Slide Enter** events in my projects for setting up slides that need to respond to actions the user may have taken elsewhere in the course. For example, later I'll show you how to set up a main menu slide with objects that change state to indicate which parts of the e-learning module the learner has not attempted as yet, which parts have been attempted but not completed, and which parts have been successfully completed. We'll use the **On Enter** slide event in another example that replicates a dynamic drop-down menu in a software simulation.

…as soon as a slide is completed

This one is more difficult than it appears. It is certainly one situation where you *could* consider using the slide's **On Exit** event. However, in order to be *certain* the slide is played all the way to

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

the end and the event is fired, you would need to disable any other means of navigation away from the slide. That could require disabling the playbar and TOC (if your project has them).

Another way to achieve this goal could be to deliberately provide an interactive object that the user must click to progress further. But again, in order to be certain that the event is fired when the slide is completed you need to have the clickable object only appear after the user has viewed all content and also make it impossible for the learner to navigate away from the slide by any other means.

Sadly, there is currently no **On Focus Lost** event for slides. In view of the difficulties attached to reliably ensuring a slide is completely viewed, I tend to avoid building interactions that require this.

…the participant answers a quiz question slide correctly

Use the **On Success** event of the quiz question. However, this may not work exactly the way you expect it to. You need to be aware of a couple of subtleties that apply to quiz questions:

- If you have feedback captions turned on for the quiz slide, then the **On Success** event only fires ***AFTER*** you dismiss the feedback caption by clicking again somewhere on the slide.
- If you have feedback captions turned off, then the **On Success** event fires as soon as you select a correct answer and click the **Submit** button.

…the participant fails to answer a question slide correctly

Use the **On Last Attempt** event of the quiz question. As with the previous example above about correct answers in quiz questions, there are some subtleties to be aware of here:

- Firstly, if you have allowed infinite attempts on the quiz question, then the **On Last Attempt** event will never fire, because as far as Captivate is concerned the user can never fail this question. See my clarification of the meaning of the term '**Infinite Attempts**'.
- Secondly, if you have allowed *a specific multiple number attempts* on the quiz question (e.g. 2, 3, 4, etc), then the **On Last Attempt** will only fire when the user has failed to select a correct answer *and* there are no more attempts allowed.
- Lastly, the **On Last Attempt** event will only fire ***AFTER*** you dismiss any feedback captions present on the slide. If there are no feedback captions to dismiss, and all other check-points mentioned above have been passed, then the **On Last Attempt** event will fire.

So as you can see, using the **On Last Attempt** event in Captivate is not quite the same as saying you want something to happen when the learner *gets the question wrong*.

…the participant passes the quiz

Use the **On Success** event of the quiz. This event is closely related to the **Quiz Results** slide. But as we found with quiz question slide events, there are also some complexities here as well. Pay close attention to the following explanation:

- If the **Quiz Results** slide is *hidden*, then the **On Success** event of the quiz fires when the participant has achieved a passing score and passes beyond the final quiz slide.
- If the **Quiz Results** slide is *showing* (i.e. not hidden), then the quiz **On Success** event fires when the participant has achieved a passing score and clicks the **Continue** button on the **Quiz Results** slide and *allows it to play all the way to the end*. Please note the final part of that last sentence.

  It seems that the **On Success** event for a quiz is only registered on the last frame of the **Quiz Results** slide. If your user gets to the **Quiz Results** slide and then uses the **TOC**,

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

playbar, or **Smart Shape button** to jump to slides *beyond* the **Quiz Results** slide, then the **On Success** event does not get fired and no event attached to it will be executed.

…the participant fails the quiz

And this one means you'll be using (you guessed it) the **On Last Attempt** event of the quiz. The number of allowed attempts on the quiz is set in **Preferences > Quiz > Pass or Fail > If Failing Grade**. The normal default is one attempt, but you can set any number here all the way up to Infinite Attempts. However, just as we found to be the case with interactive objects, if you set attempts to **Infinite**, then the **On Last Attempt** event will *never* fire.

What if you need to trigger multiple actions from a single event?

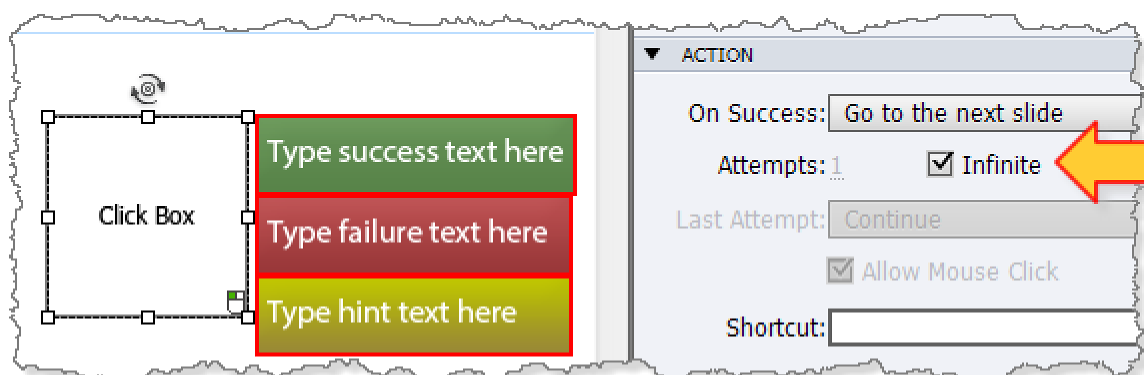
In some cases you need to do a combination of several things as a result of a single event. This is where the full power of advanced actions really becomes evident. But you need to walk before you can run. For now, just focus on getting your head around the concept that every action must be executed by some kind of run-time event. In later chapters we'll be discussing how to use Standard Actions and Conditional Actions to execute multiple events.

What 'infinite attempts' *really* means

Before we go any further we really need to clarify something about Captivate's run-time events that has many e-learning authors confused, and that is the true meaning of the term '**infinite attempts**' as it applies to interactive objects, quiz questions, and even the overall quiz itself.

The fact is that Captivate usually only ever gives you one **SUCCESSFUL** attempt at an activity, even if you've set the interactivity to allow *infinite* attempts. So, **Infinite Attempts** should more accurately be renamed as **Infinite UNSUCCESSFUL Attempts**, because that's what it really means.

Let's illustrate this concept by looking at a typical example using a click box object. As you can see from the screenshot below, this click box is set to allow **Infinite** attempts and the **On Success** action is to **Go to the next slide**. The **Last Attempt** event is disabled because you will never reach a last attempt when set to **Infinite**.



If I interact with this slide at run-time and click *inside* the click box hit area *once*, it will execute the assigned **On Success** action and jump to the next slide. So, the reality is that I only get *one* chance to execute a *successful* action.

Conversely, if I click *outside* the hit area, no success or failure event is registered. I get no *success* event because I didn't click inside the hit area. And I get no *failure* event registered because I still haven't reached the last of my infinite attempts. I can click as many times as I want, as long as I don't click the hit area and register a success. I'm allowed infinite **UNSUCCESSFUL** attempts.

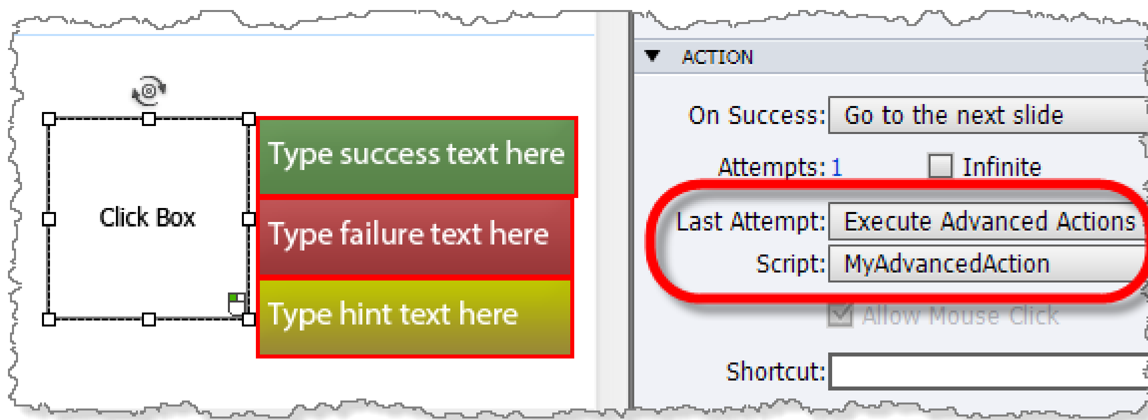
© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED

Setting up for failures (deliberately)

So what would I have to do in order to have the opportunity to trigger some action as a result of a user failure? I would need to *deselect* **Infinite attempts** and set it to a lower limit that the user is more likely to reach.

For example, if I wanted to execute *one* action when the user clicked *inside* the hit area (a success action) and a *different* action as soon as the user click *anywhere else*, then I would need to set the number of attempts to 1. This will then enable the user's first attempt to also be their **Last Attempt**, thereby triggering whatever action I configure for the **Last Attempt** event.

In the modified example shown below, I've created an advanced action (cleverly called **MyAdvancedAction**) and configured this to be executed after my last allowed *unsuccessful* attempt.



Although I've used a click box object in these examples, the same basic principle applies for all interactive objects, quiz questions, and even the overall quiz itself.

Why you need to know this

I've gone to some length explaining this popular misconception so that you don't end up falling into the same trap as many other Captivate authors who don't get this subtlety. They can often be seen posting on the Adobe Captivate Forum asking why their interactions won't allow them to click indefinitely on the same object to execute its assigned success action multiple times in a row. They often proudly proclaim to have set the number of attempts to **Infinite**, but still only get a single success action!

How to get around this interactive limitation

*There are in fact a couple of ways to beat this single successful attempt limitation. One method is called Micro Navigation and it was invented by the inimitable **Lilybiri**, **Queen of Advanced Actions** herself. The other method involves use of widgets from Infosemantics, particularly the Event Handler Interactive Widget. We will be discussing both of these methods later in this e-book.*

© 2013 by Infosemantics Pty Ltd. ALL RIGHTS RESERVED