

# Training Studio Templates Documentation

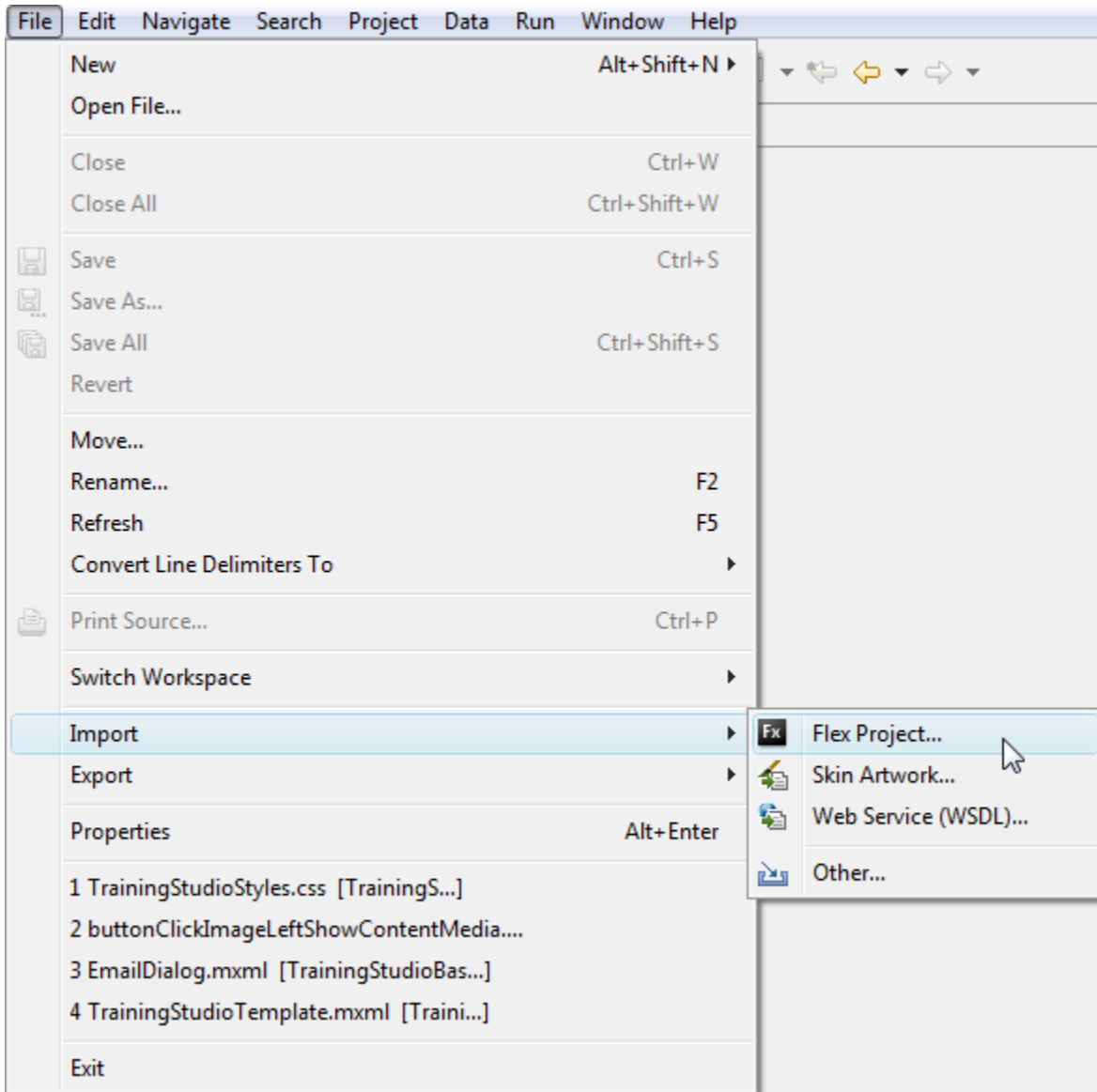
## Table of Contents

Loading the <i>TrainingStudioBase</i> Application .....	2
Architecture.....	4
Big Picture .....	5
Lesson Directory.....	7
start.swf.....	7
start.xml .....	7
TrainingStudioStyles.css.....	10
Graphical Buttons .....	16
Answer Graphics.....	17
Page Templates .....	21
Common Variables .....	22
Template Layout .....	25
Individual Templates .....	25
Template ActionScript and MXML.....	28
Deployment.....	34

This document describes how to load the *TrainingStudioBase* project. It then covers the design and structure of the Training Studio Templates and associated files.

## Loading the *TrainingStudioBase* Application

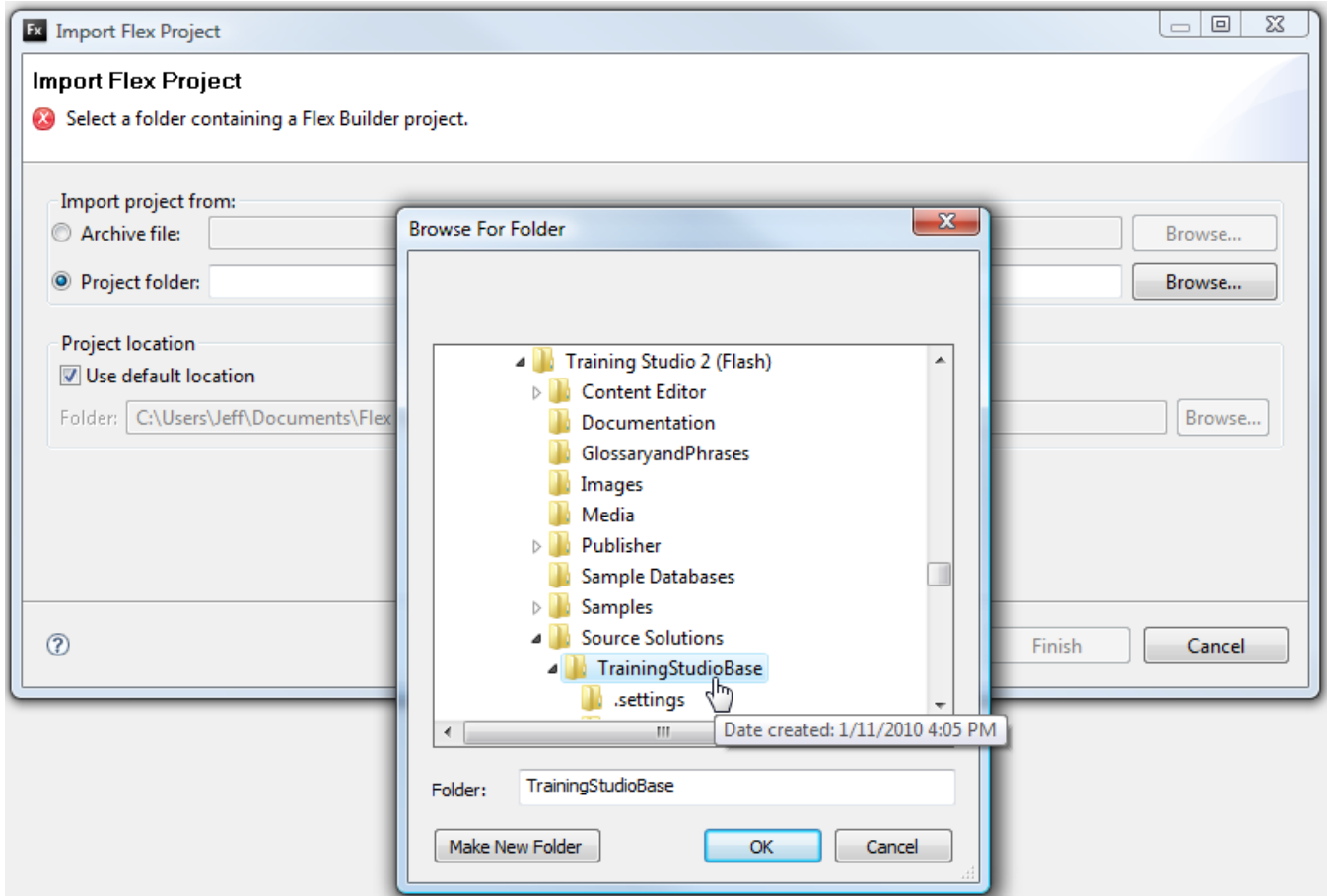
Once you install the *Training Studio Publisher* and associated files, you want to import the *TrainingStudioBase* project into Flex/Flash Builder<sup>1</sup> as shown below.



Select the “Project folder” choice and then navigate to the location of the *TrainingStudioBase* project (*C:\Program Files\Platte Canyon Multimedia Software Corporation\Training Studio 2 (Flash)\Source Solutions\TrainingStudioBase*<sup>2</sup> by default). This is shown below.

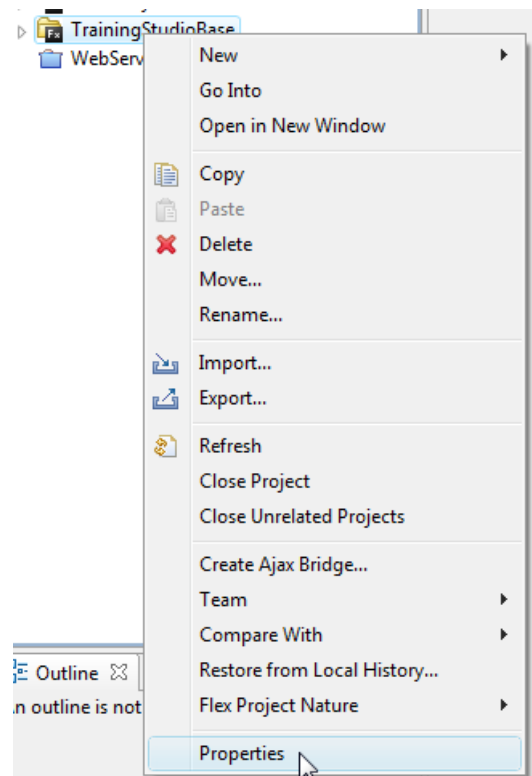
<sup>1</sup> *Flex Builder* is being renamed to *Flash Builder*. We will refer to *Flash Builder* for consistency.

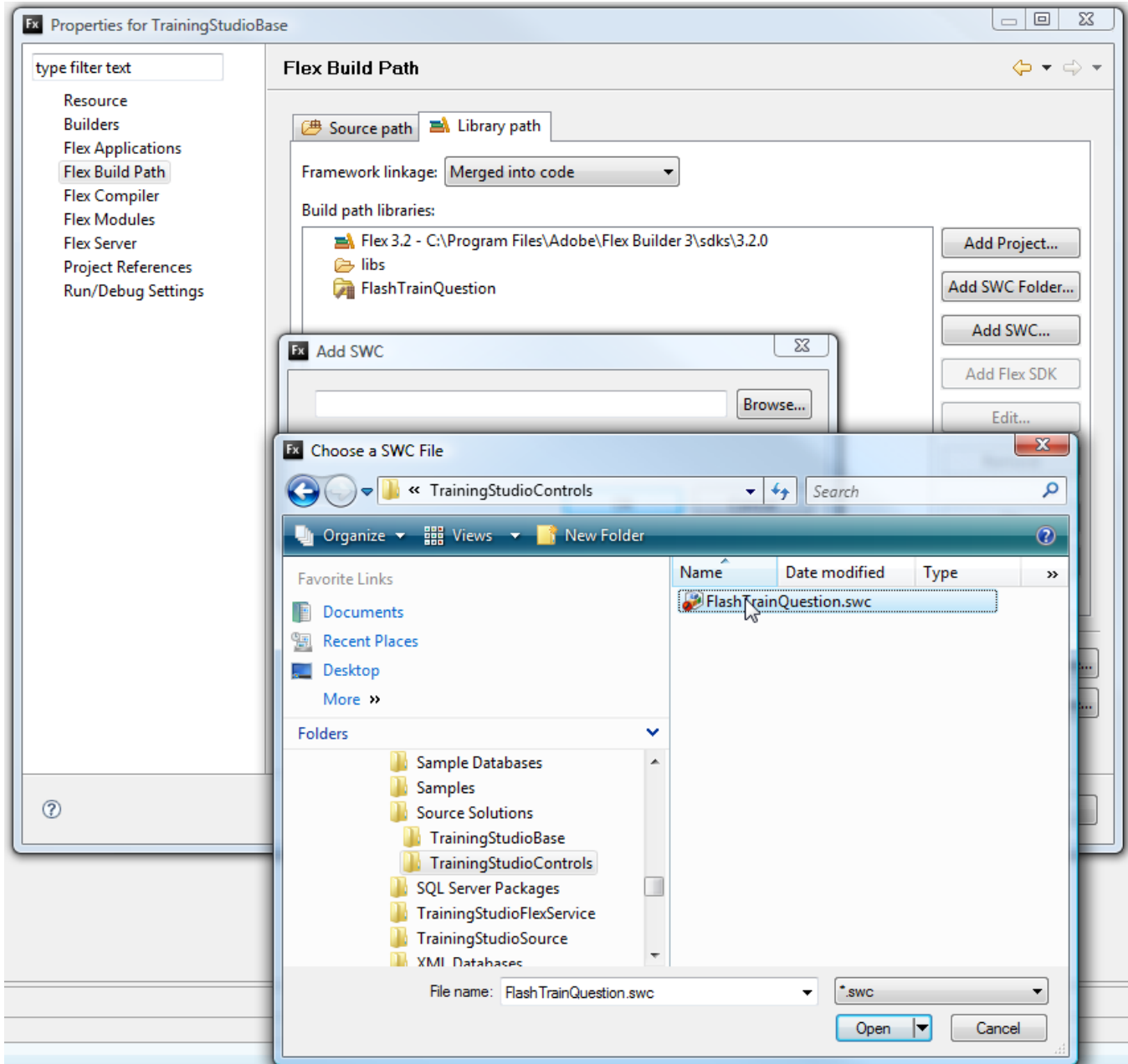
<sup>2</sup> This will be *C:\Program Files (x86)\Platte Canyon Multimedia Software Corporation\Training Studio 2 (Flash)\Source Solutions\TrainingStudioBase* on a 64-bit operating system.



You can choose the default location under *Documents* or select your own location for the project. We recommend choosing your own location. Be sure to create a directory for the project first as otherwise it will be mingled with other files and directories in the directory you choose.

Your last step is to update the reference to the *FlashTrainQuestion.swc* library. To do that, right-click on the project and go to *Properties* as shown to the right. Then select *Flex Build Path* and select the *Library path* tab. You will see *FlashTrainQuestion* in the list. You want to click the “Add SWC...” button and then browse to the path (C:\Program Files\Platte Canyon Multimedia Software Corporation\Training Studio 2 (Flash)\Source Solutions\TrainingStudioControls\FlashTrainQuestion.swc by default). This is shown below.

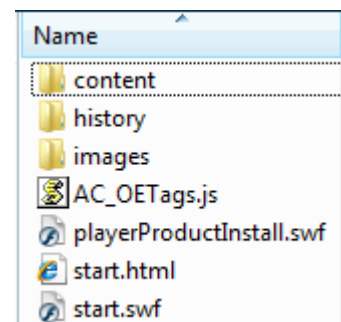




Once this is completed, delete the original *FlashTrainQuestion* reference. You are now all set to use *Flash Builder* to change *start.swf* and your templates.

## Architecture

You can look at the architecture of Training Studio lessons at several levels. At the highest level, the lesson consists only of these key files and directories<sup>3</sup>: *start.swf*, *start.html*, *playerProductInstall.swf*, *AC\_OETags.js*,

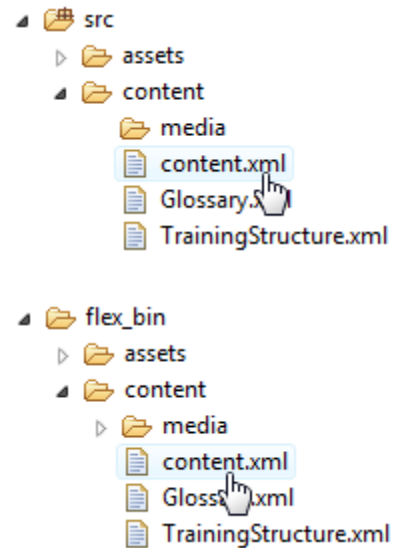


<sup>3</sup> If you publish to SCORM, there are additional files like *imsmanifest.xml*. These are for packaging purposes and not relevant to the idea of templates.

images, history, and content. This is shown to the right. Training Studio is essentially one big Flash Builder project. You will use Flash Builder to change styles, add/edit templates, etc. The entire project gets compiled into *start.swf*. The content directory contains three XML files: *content.xml* (the database of all the training content), *TrainingStructure.xml* (various settings plus language strings not related to individual training pages), and *Glossary.xml* (an optional glossary). These files are all created by the *Training Studio Content Editor*. There is a *media* subdirectory that contains the video, audio, graphics and other media files used by your lesson. The *images* directory contains any “structure” graphics such that are loaded dynamically<sup>4</sup> by the lesson. The *history* directory is one used by Flash itself. The directory structure is created with the *Training Studio Publisher*. The *TrainingStudioBase* project also has this structure so that you test your lessons from within Flash Builder. For example, you can edit the *content.xml* file in the `\src\content` directory as shown to the right. When you build your project, this file will be copied to the `\flex_bin\content` directory (also shown to the right). To test with particular media, put it directly in the `\flex_bin\content\media` directory.

### Big Picture

The figure below shows an XML version of a Training Studio database. This is normally named *content.xml* and stored in the *content* directory. The *template* (*static1* in this case) is used to grab the correct file (*static1.mxml*) from within the Flash Builder project (and compiled into the *start.swf* file), which is shown at the upper right. We cover templates in more detail later in the document. Other pieces of data such as *title*, *content\_1*, *content\_2*, *graphic\_1*, *graphic\_2*, etc. are used to populate the template. The graphics as well as the *RightNow.mp3* file are retrieved from the *media* directory. The graphics inserted into the *Image* elements defined within the template. The sound file is played programmatically.



<sup>4</sup> Any images or other files that are “Embedded” in Flash Builder are contained within *start.swf*.

The image shows the Training Studio interface with three main components:

- XML Code (Left):** A list of XML tags defining the slide's content, including page information, titles, subtitles, text blocks, media references, and graphics. The code is as follows:

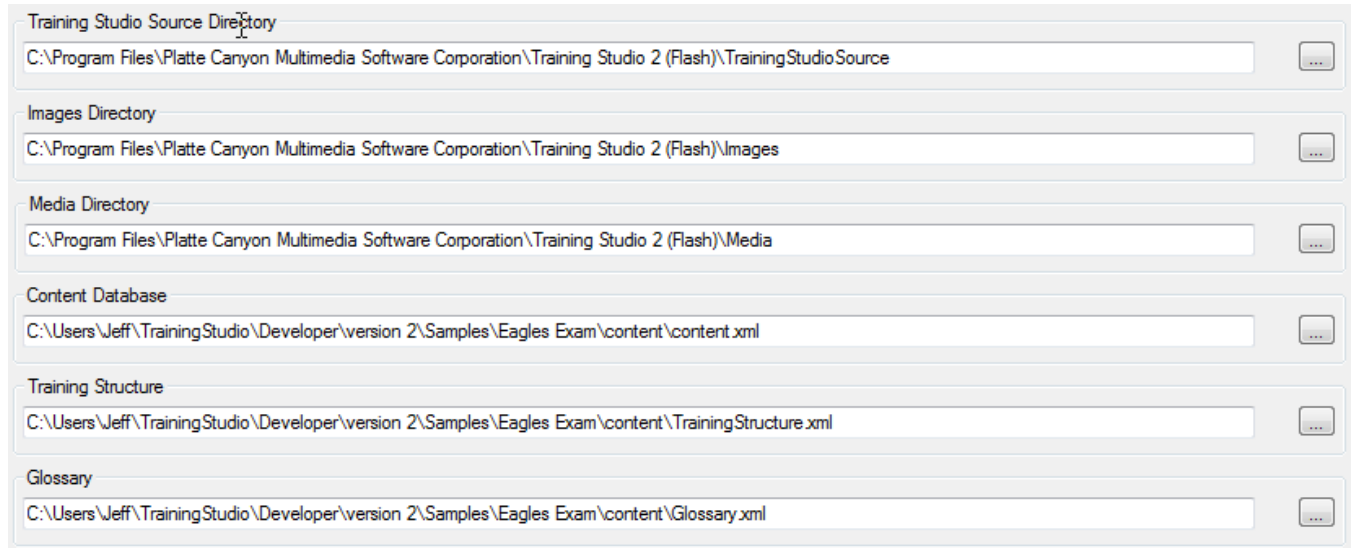
```
<Training>
  <pageId>2</pageId>
  <templateId>1</templateId>
  <pageNumber>30</pageNumber>
  <title>The Beginning (1999)</title>
  <subtitle>Fateful Conversation in Winter</subtitle>
  <content_1>There was a ToolBook List thread about h
Developer's Conference had been folded into the O
Conference.</content_1>
  <content_2>Jeff sent Chris an email saying that "may
ToolBook conference next year." Chris called on th
too late for THIS year.</content_2>
  <content_3>After talking to Slade Mitchell, Paulo Toso
and others, TBCON was born.</content_3>
  <media_0>RightNow.mp3</media_0>
  <graphic_1>jeff rhodes.gif</graphic_1>
  <graphic_2>chris bell.gif</graphic_2>
  <questionIsScored>>false</questionIsScored>
  <questionSendInteractionData>>false</questionSe
  <questionLockAfterAnswered>>false</questionLoc
  <questionDisableNextPageUntilAnswer>>false</que
  <answerCorrect_1>>false</answerCorrect_1>
  <answerCorrect_2>>false</answerCorrect_2>
  <answerCorrect_3>>false</answerCorrect_3>
  <answerCorrect_4>>false</answerCorrect_4>
  <answerCorrect_5>>false</answerCorrect_5>
  <answerCorrect_6>>false</answerCorrect_6>
  <answerCorrect_7>>false</answerCorrect_7>
  <answerCorrect_8>>false</answerCorrect_8>
  <templateType>static14</templateType>
</Training>
```
- File Explorer (Middle):** A Windows Explorer window showing the 'content' folder structure. It includes folders like 'media', 'history', and 'images', and various zip files. Specific files mentioned in the XML code are visible: 'jeff rhodes.gif', 'chris bell.gif', 'peter hoyt.gif', 'mark tate-smith.gif', 'andrew gould.gif', 'chris bell.gif', 'jeff rhodes.gif', and 'logo1.gif'.
- Rendered Slide (Bottom):** The final output of the XML code. It displays the title 'The Beginning (1999)', subtitle 'Fateful Conversation in Winter', and three paragraphs of text. Two circular images are placed on the right side of the slide, corresponding to the 'jeff rhodes.gif' and 'chris bell.gif' tags in the XML code. A 'Completed' status bar is at the bottom.

Blue arrows indicate the mapping from XML tags to their respective content in the rendered slide and file explorer. A red arrow points from the 'media' folder in the file explorer to the 'media\_0' tag in the XML code.

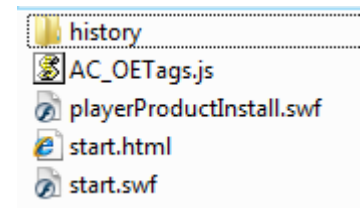
We will now look at each of the major architecture elements in turn.

## Lesson Directory

As discussed above, *start.swf* contains the main Training Studio functionality with all the external content in separate directories. The *Training Studio Publisher* builds the directory structure based on the data within the *Publish Lesson* screen shown below.



The *Training Studio Source Directory* contains *start.swf* as well as the other non-content files and directories required by Training Studio. This is shown in the screen capture to the right. When you update *start.swf* as described later in this document, be sure to update the copy in the source directory. The publish process then adds all the files in the selected images and media directories. It then creates the *content* directory, copying the *Content Database*, *Training Structure*, and *Glossary* files into it.



### **start.swf**

This is the compiled version of the templates, styles, embedded graphics, and logic of your lesson. Building the source files (included with Training Studio) updates this file. Note that the source project references *FlashTrainQuestion.swc*, located in the *\Source Solutions\TrainingStudioControls* directory. The source code for this component is not provided, though there is little chance that you would need to do anything with this component.

Adding, editing, or deleting templates involves editing the source files, rebuilding *start.swf*, updating your “Source Directory,” and publishing your lessons.

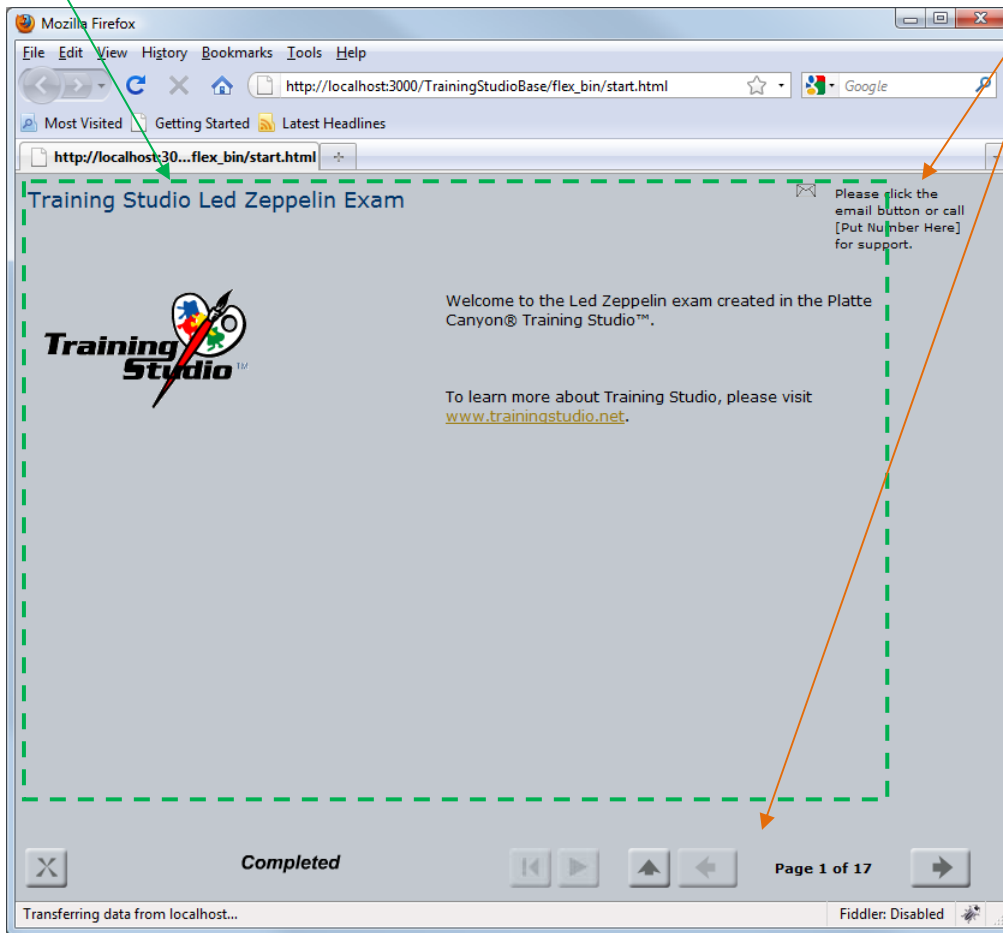
Within the *start.swf* are various *.mxml* (Macromedia eXtensible Markup Language) files, *.as* (ActionScript) files, an important style sheet (*TrainingStudioStyles.css*), and other supporting files.

### **start.mxml**

As shown in the figure below, *start.mxml* contains such items as the navigation buttons and audio controls. We recommend using *Flash Builder* to edit *start.mxml* and other Training Studio files.

contentRightImageLeft.mxml

start.mxml



The key elements of *start.mxml* are shown below.

<i>Item</i>	<i>Description</i>
SupportGrid	This is a grid that contains the “tech support email” graphic and associated technical support text. This can be turned off as part of the Training Structure. It defaults to a position and width of $x="650"$ , $y="8"$ , $width="150"$ . You can edit it if desired.
transition	This is the area of <i>start.mxml</i> that contains the individual page templates such as <i>contentRightImageLeft.mxml</i> in the figure above. It is a <i>ViewStack</i> control that defaults to these properties: $x="0.1"$ , $y="0.1"$ , $width="799.1"$ , $height="555"$ . Note that the width and height correspond to the width and height of the page templates. If you adjust the dimensions of the <i>transition</i> control, be sure to adjust the page templates to match. You would also want to change the <i>maxWidth</i> variable in <i>TSCCommon.as</i> as that is used in calculations related to displaying menus.
ExitBtn	This is a <i>GraphicalButton</i> component that user can click on to exit the

<i>Item</i>	<i>Description</i>
	<p>lesson. Its graphics are controlled by the <i>ExitBtn</i> style in <i>TrainingStudioStyles.css</i>. Its tooltip is controlled by the <i>ExitBtn_ToolTip</i> setting in the <i>Training Structure</i>. The message displayed to confirm exit is controlled by the <i>ExitBtn_Message</i> setting in the <i>Training Structure</i>. It is located by default at: <i>x="8" y="561"</i></p>
CompletionImage	<p>This is a <i>SWFLoader</i> control that displays <i>gonext.swf</i>, located in the <i>media</i> directory, by default when all interactions on a page are completed. It is located by default at: <i>x="179" y="556"</i></p>
RewindBtn	<p>This is a <i>GraphicalButton</i> component that user can click on to rewind the current audio file. It is disabled unless an audio file is playing. Its graphics are controlled by the <i>RewindBtn</i> style in <i>TrainingStudioStyles.css</i>. Its tooltip is controlled by the <i>ReplayBtn_ToolTip</i> setting in the <i>Training Structure</i>. It is located by default at: <i>x="411" y="561"</i></p>
PauseBtn	<p>This is a <i>GraphicalButton</i> component that user can click on to pause the current audio file. It is hidden unless an audio file is playing. Its graphics are controlled by the <i>PauseBtn</i> style in <i>TrainingStudioStyles.css</i>. Its tooltip is controlled by the <i>PauseBtn_ToolTip</i> setting in the <i>Training Structure</i>. It is located by default at: <i>x="451" y="561"</i></p>
PlayBtn	<p>This is a <i>GraphicalButton</i> component that user can click on to play the current audio file. It is disabled unless there is an audio file and is hidden once the audio starts playing. When the sound finishes or the user clicks the Pause button, this button is shown again. Its graphics are controlled by the <i>PlayBtn</i> style in <i>TrainingStudioStyles.css</i>. Its tooltip is controlled by the <i>PlayBtn_ToolTip</i> setting in the <i>Training Structure</i>. It is located by default at: <i>x="451" y="561"</i></p>
OptionsBtn	<p>This is a <i>GraphicalButton</i> component that user can click on to display an "Options" menu. This menu has an <i>Options</i> choice for turning the sound on and off and a <i>Glossary</i> choice to display the Glossary. Setting the <i>includeTBKTrackerStudentInfoLink</i> variable in <i>TSCCommon.as</i> file to true displays adds a "Student Information" link (controlled by the <i>StudentInformation_Text</i> setting in <i>Training Structure</i>) to this menu. This will bring up the <i>Student Information</i> screen when running via the <i>TBK Tracker Learning Management System</i>. Its graphics are controlled by the <i>OptionsBtn</i> style in <i>TrainingStudioStyles.css</i>. Its tooltip is controlled by the <i>OptionsBtn_ToolTip</i> setting in the <i>Training Structure</i>. It is located by default at: <i>x="510" y="561"</i></p>
PreviousBtn	<p>This is a <i>GraphicalButton</i> component that user can click on to move to the previous page of the lesson. It is disabled automatically on the first page of the lesson. Its graphics are controlled by the <i>PreviousBtn</i> style in <i>TrainingStudioStyles.css</i>. Its tooltip is controlled by the <i>PreviousBtn_ToolTip</i> setting in the <i>Training Structure</i>. It is located by default at: <i>x="551" y="561"</i></p>
MenuBtn	<p>This is a <i>Label</i> component that shows "Page X of Y" by default. This text is set by the <i>PageLabel_Text</i> setting in the <i>Training Structure</i>. In that, the {0} is replaced by the current page number and the {1} is replaced by the number of pages in the lesson. If the <i>DisableMenu</i> setting in the <i>Training Structure</i> is set to true, nothing will happen when the user clicks on this text. You might set this to true for an exam, for example. Otherwise, clicking on this text displays either a popup menu window (if the number</p>

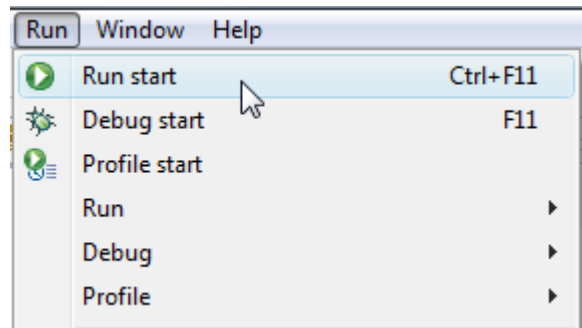
Item	Description
	of pages is more than the <i>numPagesToUsePopupMenu</i> variable in <i>TSCCommon.as</i> ) or a normal menu. Within either menu, the <i>includePageNumbersOnMenu</i> variable in <i>TSCCommon.as</i> controls whether the page number is shown before the page title. The default look of this label is controlled by the <i>MenuText</i> style in <i>TrainingStudioStyles.css</i> . Its “rollover” style is controlled by <i>MenuTextRollover</i> . Its tooltip is controlled by the <i>MenuBtn_ToolTip</i> setting in the <i>Training Structure</i> . It is located by default at: <i>x=“609” y=“569” width=“128”</i>
NextBtn	This is a <i>GraphicalButton</i> component that user can click on to move to the next page of the lesson. It is disabled automatically on the last page of the lesson. Its graphics are controlled by the <i>NextBtn</i> style in <i>TrainingStudioStyles.css</i> . Its tooltip is controlled by the <i>NextBtn_ToolTip</i> setting in the <i>Training Structure</i> . It is located by default at: <i>x=“745” y=“561”</i>

### TrainingStudioStyles.css

This file is very important and is the main reason that you will use *Flash Builder* to update *start.swf*. You edit the file by double-clicking on it in the *\src\Styles* directory in the *TrainingStudioBase* project. Graphics and other resources can be embedded into the styles using the *Embed* syntax as shown below. These are usually contained in the *src\images* directory (and then don’t need to be included in the *images* directory that you include when you publish since they are then included in *start.swf*), but you can use other directories if you want.

```
.NextBtn {
    overSkin:Embed("/images/right over.gif");
    upSkin:Embed("/images/right up.gif");
    downSkin:Embed("/images/right down.gif");
    disabledSkin:Embed("/images/right disabled.gif");
}
```

You can test your new styles by selecting *Run start* as shown to the right. If you want to use your own content, update the files in *\src\content*. If you want to use your own media, update the *\flex\_bin\content\media* directory.



Each of the style classes is explained below. They are listed in the same order as in the style sheet itself. The ones related to graphical buttons and answer buttons are explained in the following sections. This site is a good reference for coming up with the property CSS for the effects that you might want: <http://examples.adobe.com/flex3/consulting/styleexplorer/Flex3StyleExplorer.html>.

Class	Description
MenuText	text-align: center; font-size: 11; fontWeight: bold; color: #000000;

Class	Description
	This controls the <i>MenuBtn</i> described in the previous section. This is the original state of the label. Note that this and other colors are in hexadecimal format <sup>5</sup> .
MenuTextRollover	<pre>text-align: center; font-size: 11; fontWeight: bold; color: #ffffcc; textDecoration: underline;</pre> <p>This controls the rollover state of the <i>MenuBtn</i> described in the previous section.</p>
EvalBox	<pre>text-align: center; font-size: 11; color: #000000;</pre> <p>This controls the label that displays an evaluation message when using the demo version of Training Studio.</p>
PagesMenu	<pre>backgroundColor: #c2c8cf; rollOverColor: #d6d3ce; textRollOverColor: #003366; selectionColor: #ffffcc; dropShadowEnabled: true;</pre> <p>This controls the look of the popup menu that displays if the number of pages is less than or equal to the <i>numPagesToUsePopupMenu</i> variable in <i>TSCCommon.as</i>. You might want to adjust the various colors to more closely match your desired interface.</p>
ContentTransparent	<pre>fontSize: 12; color: #000000; backgroundAlpha: 0; borderStyle: none; focusAlpha: 0; /* no border */</pre> <p>This controls the look of the <i>content_0</i> through <i>content_10 TextArea</i> components contained in most templates. The <i>backgroundAlpha</i> sets the components to be transparent. The <i>focusAlpha</i> prevents a border on the component when it gets the focus.</p>
Hotspot	<pre>fontSize: 13; color: #634037;</pre> <p>This controls the “initial” state of <i>hotspot_1</i> through <i>hotspot_10 Label</i> components on templates like <i>mouseEnterLeftShowContentImage</i>. The state goes from <i>initial</i> when the page is opened to <i>current</i> when the user is on that hotspot to <i>completed</i> when the user interacts with the next hotspot.</p>

<sup>5</sup> One easy way to come up with this value is to go to a graphics program like PhotoShop. Select the color you want or enter normal RGB (Red, Green, Blue) values. The hex value will be in its own field. Copy it from there and put a # symbol in front.

<i>Class</i>	<i>Description</i>
HotspotCompleted	<p>fontSize: 13; color: #000066;</p> <p>This controls the “completed” state of <i>hotspot_1</i> through <i>hotspot_10 Label</i> components on templates like <i>mouseEnterLeftShowContentImage</i>. The state goes from <i>initial</i> when the page is opened to <i>current</i> when the user is on that hotspot to <i>completed</i> when the user interacts with the next hotspot.</p>
HotspotCurrent	<p>fontSize: 13; color: #009900; backgroundColor: #FFFF00;</p> <p>This controls the “current” state of <i>hotspot_1</i> through <i>hotspot_10 Label</i> components. Use this value if you want the hotspot to have a background color. If not, use the <i>HotspotCurrentNoBackground</i> style listed below. The state goes from <i>initial</i> when the page is opened to <i>current</i> when the user is on that hotspot to <i>completed</i> when the user interacts with the next hotspot.</p>
HotspotCurrentNoBackground	<p>fontSize: 13; color: #009900;</p> <p>This controls the “current” state of <i>hotspot_1</i> through <i>hotspot_10 Label</i> components on templates like <i>mouseEnterLeftShowContentImage</i>. Use this value if you don’t want the hotspot to have a background color. Otherwise, use the <i>HotspotCurrent</i> style listed above. The state goes from <i>initial</i> when the page is opened to <i>current</i> when the user is on that hotspot to <i>completed</i> when the user interacts with the next hotspot.</p>
ImageHotspot	<p>backgroundAlpha: 0; borderThickness: 0;</p> <p>This controls the “initial” state of <i>graphic_1</i> through <i>graphic_10 QImage</i><sup>6</sup> components on templates like <i>buttonClickImageLeftShowContentMedia</i>. The state goes from <i>initial</i> when the page is opened to <i>current</i> when the user is on that hotspot to <i>completed</i> when the user interacts with the next hotspot.</p>
ImageHotspotCompleted	<p>alpha: 0.5; borderThickness: 3; borderColor: #000066;</p> <p>This controls the “completed” state of <i>graphic_1</i> through <i>graphic_10 QImage</i> components on templates like <i>buttonClickImageLeftShowContentMedia</i>. The state goes from <i>initial</i> when the page is opened to <i>current</i> when the user is on that hotspot to <i>completed</i> when the user interacts with the next hotspot.</p>
ImageHotspotCurrent	<p>alpha: 0.5; borderThickness: 3; borderColor: #FFFF00;</p> <p>This controls the “current” state of <i>graphic_1</i> through <i>graphic_10 QImage</i></p>

<sup>6</sup> *QImage* is a special Training Studio component that allows the images to have a border and/or fill color.

<i>Class</i>	<i>Description</i>
	<p>components on templates like <i>buttonClickImageLeftShowContentMedia</i>. Use this value if you want the hotspot to have a background color. If not, use the <i>ImageHotspotCurrentNoBackground</i> style listed below. The state goes from <i>initial</i> when the page is opened to <i>current</i> when the user is on that hotspot to <i>completed</i> when the user interacts with the next hotspot.</p>
ImageHotspotCurrentNoBackground	<p>alpha: 0.5; borderThickness: 3; borderColor: #009900;</p> <p>This controls the “current” state of <i>graphic_1</i> through <i>graphic_10 QImage</i> components. Use this value if you don’t want the hotspot to have a background color. Otherwise, use the <i>ImageHotspotCurrent</i> style listed above. The state goes from <i>initial</i> when the page is opened to <i>current</i> when the user is on that hotspot to <i>completed</i> when the user interacts with the next hotspot.</p>
GlossaryBackground	<p>background-color: #92b9c1;</p> <p>This controls the background of the popup Glossary window.</p>
GlossaryList	<p>color: #220F04; font-size: 12; rollOverColor: #6699cc; selectionColor: #669999;</p> <p>This controls the <i>List</i> component that displays the glossary terms in the popup Glossary window.</p>
GlossaryDisplay	<p>color: #220F04; font-size: 12;</p> <p>This controls the <i>TextArea</i> component that displays the selected glossary definition in the popup Glossary window.</p>
CommentsBackground	<p>background-color: #92b9c1;</p> <p>This controls the background of the popup Comments window.</p>
CommentsList	<p>color: #220F04; rollOverColor: #6699cc; selectionColor: #669999;</p> <p>This controls the <i>List</i> component that displays the comment categories in the popup Comments window.</p>
ReviewersList	<p>color: #220F04; rollOverColor: #6699cc; selectionColor: #669999;</p> <p>This controls the <i>List</i> component that displays the reviewer categories in the popup Comments window.</p>
PagesMenuBackground	<p>background-color: #92b9c1;</p> <p>This controls the background of the popup menu window that displays if</p>

<i>Class</i>	<i>Description</i>
	the number of pages is greater than the <i>numPagesToUsePopupMenu</i> variable in <i>TSCCommon.as</i> .
PagesList	<p>color: #220F04;  rollOverColor: #6699cc;  selectionColor: #669999;</p> <p>This controls the <i>List</i> component that displays the page titles in the popup menu window.</p>
OptionsBackground	<p>background-color: #92b9c1;</p> <p>This controls the background of the popup Options window.</p>
soundOnStyle	<p>color: #220F04;  font-size: 12;</p> <p>This controls the “Sound On” check box on the popup Options window.</p>
Title	<p>color: #003366;  fontSize: 18;  backgroundAlpha: 0;  borderStyle: none;</p> <p>This controls the Title <i>Text</i> component that is on all templates.</p>
Subtitle	<p>color: #003366;  fontSize: 14;  backgroundAlpha: 0;  borderStyle: none;</p> <p>This controls the Subtitle <i>Text</i> component that is on all templates.</p>
EmailDialogBackground	<p>background-color: #92b9c1;</p> <p>This controls the background color of the window that pops up when the user clicks the optional tech support email graphic.</p>
InstructionsColor	<p>color: #000000;  text-align: left;  font-size: 12;</p> <p>This controls text such as the instructions on the support email window.</p>
SupportMessage	<p>color: #000000;  text-align: left;  font-size: 10;</p> <p>This controls the technical support text that optionally shows on start.xml.</p>
Message	<p>color: #000000;  font-size: 12;</p> <p>This controls status text on various popup windows.</p>
QuestionTextColor	<p>color: #220F04;  font-size: 12;</p>

<i>Class</i>	<i>Description</i>
	<p>This controls the label that shows the text of questions for question templates.</p>
AnswersInitialTextColor	<p>color: #2D2805; textSelectedColor: #2D2805; textRollOverColor: #2D2805;</p> <p>This is the initial color of the various answers (check boxes, buttons, etc.) on question templates. Notice that we set the <i>textSelectedColor</i> and the <i>textRollOverColor</i> styles as well so that the color doesn't change when we select the object or roll our mouse into it.</p>
AnswersSelectedTextColor	<p>color: #1111F9; textSelectedColor: #1111F9; textRollOverColor: #1111F9;</p> <p>This is the selected color of the various answers (check boxes, buttons, etc.) on question templates. An answer goes into selected state if there is more than one correct answer and the question has not yet been scored. Notice that we set the <i>textSelectedColor</i> and the <i>textRollOverColor</i> styles as well so that the color doesn't change when we select the object or roll our mouse into it.</p>
AnswersCorrectTextColor	<p>color: #128212; textSelectedColor: #128212; textRollOverColor: #128212;</p> <p>This is the "correct" color of the various answers (check boxes, buttons, etc.) on question templates. The object turns into this state once the question is scored and the answer is correct. If you are doing a more serious exam and don't want the user to know whether the answer was correct or incorrect, set both this, <i>AnswersIncorrectTextColor</i> (as well as the ones for your question graphics as discussed below) to a neutral color like gray. Notice that we set the <i>textSelectedColor</i> and the <i>textRollOverColor</i> styles as well so that the color doesn't change when we select the object or roll our mouse into it.</p>
AnswersIncorrectTextColor	<p>color: #128212; textSelectedColor: #128212; textRollOverColor: #128212;</p> <p>This is the "incorrect" color of the various answers (check boxes, buttons, etc.) on question templates. The object turns into this state once the question is scored and the answer is incorrect. If you are doing a more serious exam and don't want the user to know whether the answer was correct or incorrect, set both this, <i>AnswersCorrectTextColor</i> (as well as the ones for your question graphics as discussed below) to a neutral color like gray. Notice that we set the <i>textSelectedColor</i> and the <i>textRollOverColor</i> styles as well so that the color doesn't change when we select the object or roll our mouse into it.</p>
FeedbackInstructionsTextColor	<p>color: #342B68; font-size: 12;</p>

<i>Class</i>	<i>Description</i>
	This controls the instructions and the feedback text that are displayed below each question in the question templates.
TargetsInitialTextColor	color: #EA611A;  This controls the “Targets” in the <i>question_matchingLines.mxml</i> template.
MainBackgroundColor	backgroundGradientColors: #c2c8cf, #c2c8cf; /* no gradient since the same color */  You can also set up a background image but it is not straightforward to have it tiled. This site is a good reference for coming up with the property CSS for the effects that you might want: <a href="http://examples.adobe.com/flex3/consulting/styleexplorer/Flex3StyleExplorer.html">http://examples.adobe.com/flex3/consulting/styleexplorer/Flex3StyleExplorer.html</a> .
initialImage	borderThickness: 0;  This controls the initial state of the images in the <i>question_hotObjects.mxml</i> , <i>question_hotSpotImageText.mxml</i> , and <i>question_hotSpot.mxml</i> templates.
selectedImage	borderColor: #1111F9; borderThickness: 3;  This controls the selected state of the images in the <i>question_hotObjects.mxml</i> , <i>question_hotSpotImageText.mxml</i> , and <i>question_hotSpot.mxml</i> templates. The images will become selected if there are multiple correct answers and the question has not yet been scored.
correctImage	borderColor: #128212; borderThickness: 3;  This controls the initial state of the images in the <i>question_hotObjects.mxml</i> , <i>question_hotSpotImageText.mxml</i> , and <i>question_hotSpot.mxml</i> templates. If you are doing a more serious exam and don’t want the user to know whether the answer was correct or incorrect, set both this and <i>incorrectImage</i> to a neutral color like gray.
incorrectImage	borderColor: #ED2919; borderThickness: 3;  This controls the initial state of the images in the <i>question_hotObjects.mxml</i> , <i>question_question_hotSpotImageText.mxml</i> , and <i>question_hotSpot.mxml</i> templates. If you are doing a more serious exam and don’t want the user to know whether the answer was correct or incorrect, set both this and <i>correctImage</i> to a neutral color like gray.

## Graphical Buttons

Rather than setting the states of the button graphics programmatically, it is most efficient in Flash to do it via the style sheet. Notice the use of the *Embed* syntax in the styles below. If you want to use different button graphics (Next, Previous, etc.), the easiest approach is to use the same names as the existing graphics, overwrite the ones in the `\src\images` directory, and rebuild *start.swf*. However, you can edit

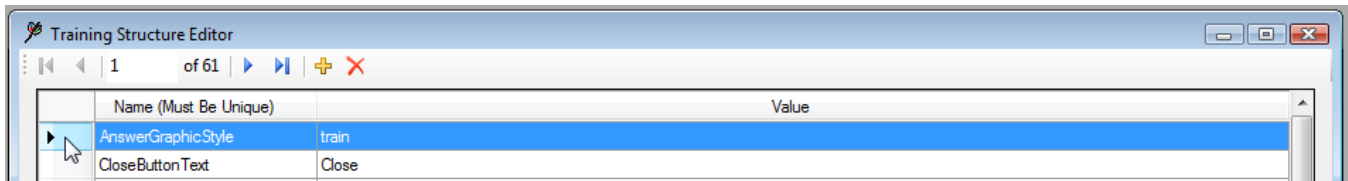
the TrainingStudioStyles.css and change the file names or directories. These are listed below. Most have four states (*overSkin*, *upSkin*, *downSkin*, and *disabledSkin*); those which cannot be disabled just have the first three states. Gif, JPG, and PNG formats are all acceptable.

<i>Class</i>	<i>Description</i>
ExitBtn	overSkin:Embed("/images/exit over.gif"); upSkin:Embed("/images/exit up.gif"); downSkin:Embed("/images/exit down.gif");
NextBtn	overSkin:Embed("/images/right over.gif"); upSkin:Embed("/images/right up.gif"); downSkin:Embed("/images/right down.gif"); disabledSkin:Embed("/images/right disabled.gif");
PreviousBtn	overSkin:Embed("/images/left over.gif"); upSkin:Embed("/images/left up.gif"); downSkin:Embed("/images/left down.gif"); disabledSkin:Embed("/images/left disabled.gif");
OptionsBtn	overSkin:Embed("/images/menu over.gif"); upSkin:Embed("/images/menu up.gif"); downSkin:Embed("/images/menu down.gif"); disabledSkin:Embed("/images/menu disabled.gif");
PlayBtn	overSkin:Embed("/images/play over.jpg"); upSkin:Embed("/images/play up.jpg"); downSkin:Embed("/images/play down.jpg"); disabledSkin:Embed("/images/play disabled.jpg");
PauseBtn	overSkin:Embed("/images/pause over.jpg"); upSkin:Embed("/images/pause up.jpg"); downSkin:Embed("/images/pause down.jpg"); disabledSkin:Embed("/images/pause disabled.jpg");
RewindBtn	overSkin:Embed("/images/rewind over.jpg"); upSkin:Embed("/images/rewind up.jpg"); downSkin:Embed("/images/rewind down.jpg"); disabledSkin:Embed("/images/rewind disabled.jpg");
ResetBtn	overSkin:Embed("/images/reset-over.png"); upSkin:Embed("/images/reset-up.png"); downSkin:Embed("/images/reset-down.png"); disabledSkin:Embed("/images/reset-disabled.png");
ScoreBtn	overSkin:Embed("/images/score-over.png"); upSkin:Embed("/images/score-up.png"); downSkin:Embed("/images/score-down.png"); disabledSkin:Embed("/images/score-disabled.png");
ShowAnswersBtn	overSkin:Embed("/images/showAnswers-over.png"); upSkin:Embed("/images/showAnswers-up.png"); downSkin:Embed("/images/showAnswers-down.png"); disabledSkin:Embed("/images/showAnswers-disabled.png");

## **Answer Graphics**

Many of the question templates can have graphics associated with each of the question states: initial, selected, correct, and incorrect. As with the graphical buttons discussed in the previous section, it is

most efficient to have each of these states set up in the TrainingStudioStyles.css file as their own classes. Training Studio determines which class to use by looking at the *AnswerGraphicStyle* setting in the *Training Structure* as shown below.



For example, an *AnswerGraphicStyle* of train means that these styles are used: *trainnormal*, *trainselected*, *traincorrect*, and *trainincorrect*.

If the *AnswerGraphicStyle* is blank, then no answer graphics are used.

Again notice the use of the *Embed* syntax in the styles below. If you want to use different graphics, the easiest approach is to use the same names as the existing graphics, overwrite the ones in the `\src\images` directory, and rebuild *start.swf*. However, you can edit the TrainingStudioStyles.css and change the file names or directories. These are listed below. In addition to the four states (*overSkin*, *upSkin*, *downSkin*, and *disabledSkin*) that we saw for the graphical buttons, we have various selected states (for check boxes), color (to change the text of the button as well), and more. Note also that these use the *Icon* rather than *Skin* properties. This allows them to be used for check box and radio button components as well as buttons. If you are creating a more serious exam where you don't want the user to know whether the answers she selected is right or wrong, you can set both the correct and incorrect classes to use the same graphics and colors.

<i>Class</i>	<i>Description</i>
canyoncorrect	overIcon:Embed("/media/canyon_correct.png"); upIcon:Embed("/media/canyon_correct.png"); downIcon:Embed("/media/canyon_correct.png"); disabledIcon:Embed("/media/canyon_correct.png"); selectedOverIcon:Embed("/media/canyon_correct.png"); selectedUpIcon:Embed("/media/canyon_correct.png"); selectedDownIcon:Embed("/media/canyon_correct.png"); selectedDisabledIcon:Embed("/media/canyon_correct.png"); color: #128212; textSelectedColor: #128212; textRollOverColor: #128212;
canyonincorrect	overIcon:Embed("/media/canyon_incorrect.png"); upIcon:Embed("/media/canyon_incorrect.png"); downIcon:Embed("/media/canyon_incorrect.png"); disabledIcon:Embed("/media/canyon_incorrect.png"); selectedOverIcon:Embed("/media/canyon_incorrect.png"); selectedUpIcon:Embed("/media/canyon_incorrect.png"); selectedDownIcon:Embed("/media/canyon_incorrect.png"); selectedDisabledIcon:Embed("/media/canyon_incorrect.png"); color: #ED2919; textSelectedColor: #ED2919;

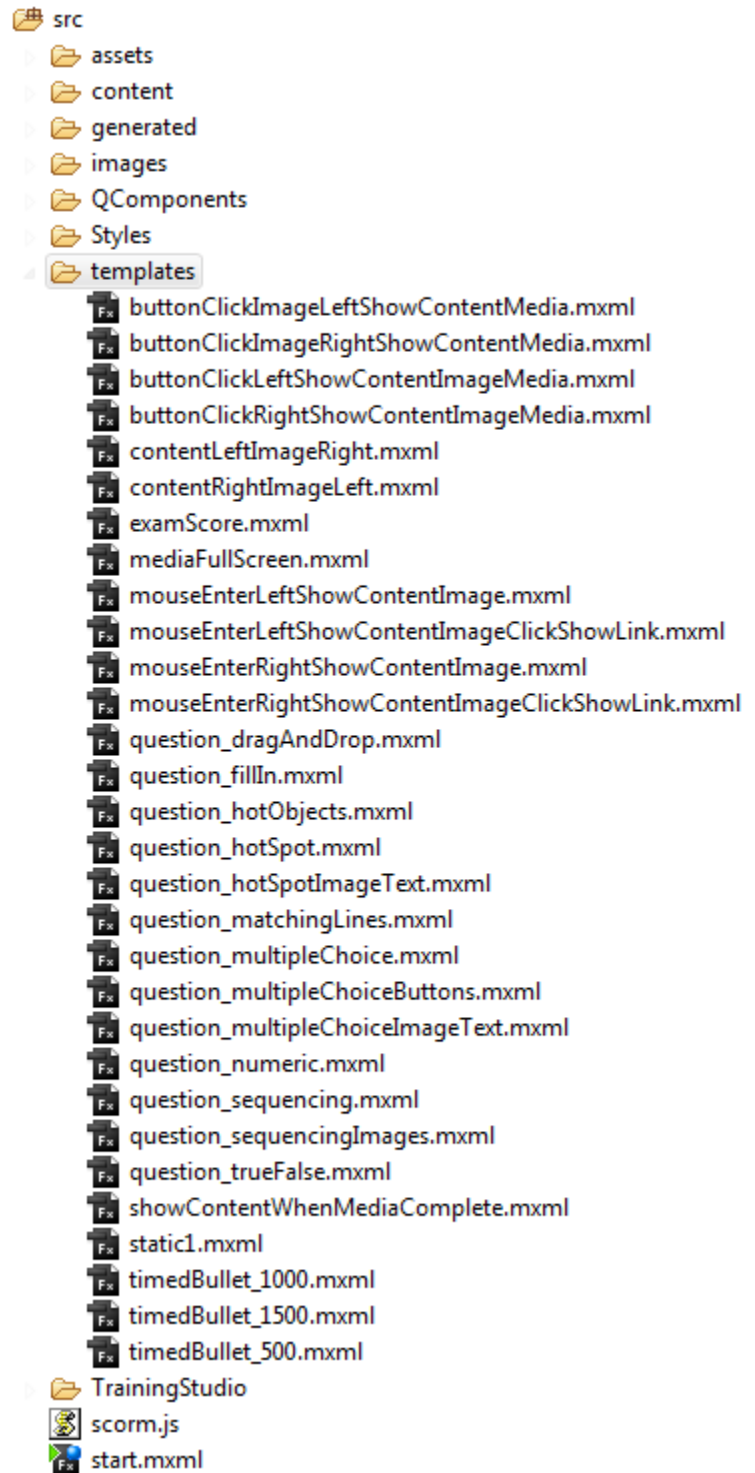
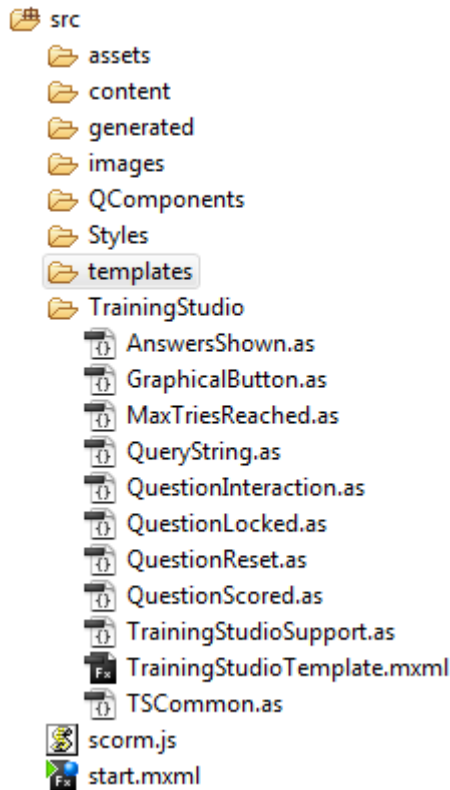
<i>Class</i>	<i>Description</i>
	textRollOverColor: #ED2919;
canyonnormal	overIcon:Embed("/media/canyon_normal.png"); upIcon:Embed("/media/canyon_normal.png"); downIcon:Embed("/media/canyon_normal.png"); disabledIcon:Embed("/media/canyon_normal.png"); color: #2D2805; textSelectedColor: #2D2805; textRollOverColor: #2D2805;
canyonselected	overIcon:Embed("/media/canyon_selected.png"); upIcon:Embed("/media/canyon_selected.png"); downIcon:Embed("/media/canyon_selected.png"); disabledIcon:Embed("/media/canyon_selected.png"); selectedOverIcon:Embed("/media/canyon_selected.png"); selectedUpIcon:Embed("/media/canyon_selected.png"); selectedDownIcon:Embed("/media/canyon_selected.png"); selectedDisabledIcon:Embed("/media/canyon_selected.png"); color: #1111F9; textSelectedColor: #1111F9; textRollOverColor: #1111F9;
screwcorrect	overIcon:Embed("/media/screw_correct.png"); upIcon:Embed("/media/screw_correct.png"); downIcon:Embed("/media/screw_correct.png"); disabledIcon:Embed("/media/screw_correct.png"); selectedOverIcon:Embed("/media/screw_correct.png"); selectedUpIcon:Embed("/media/screw_correct.png"); selectedDownIcon:Embed("/media/screw_correct.png"); selectedDisabledIcon:Embed("/media/screw_correct.png"); color: #128212; textSelectedColor: #128212; textRollOverColor: #128212;
screwincorrect	overIcon:Embed("/media/screw_incorrect.png"); upIcon:Embed("/media/screw_incorrect.png"); downIcon:Embed("/media/screw_incorrect.png"); selectedOverIcon:Embed("/media/screw_incorrect.png"); selectedUpIcon:Embed("/media/screw_incorrect.png"); selectedDownIcon:Embed("/media/screw_incorrect.png"); selectedDisabledIcon:Embed("/media/screw_incorrect.png"); color: #ED2919; textSelectedColor: #ED2919; textRollOverColor: #ED2919;
screwnormal	overIcon:Embed("/media/screw_normal.png"); upIcon:Embed("/media/screw_normal.png"); downIcon:Embed("/media/screw_normal.png"); disabledIcon:Embed("/media/screw_normal.png"); color: #2D2805; textSelectedColor: #2D2805; textRollOverColor: #2D2805;
screwselected	overIcon:Embed("/media/screw_selected.png"); upIcon:Embed("/media/screw_selected.png");

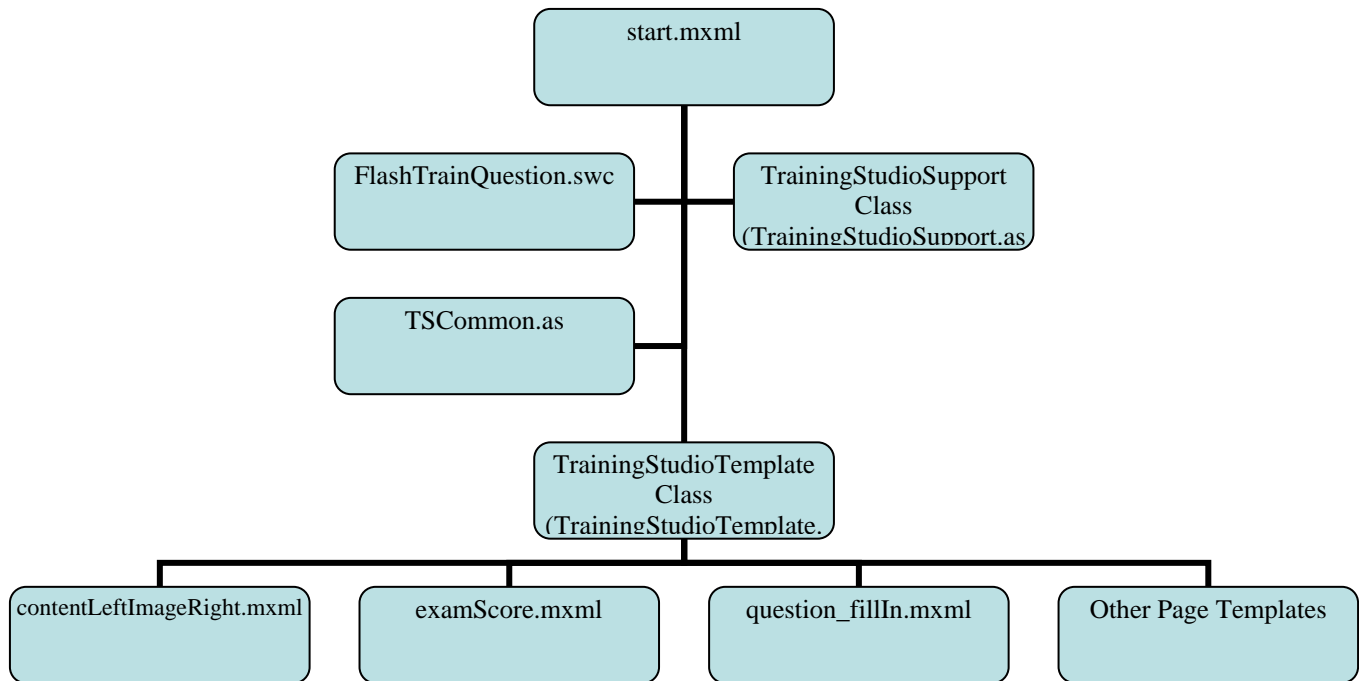
<i>Class</i>	<i>Description</i>
	downIcon:Embed("/media/screw_selected.png"); disabledIcon:Embed("/media/screw_selected.png"); selectedOverIcon:Embed("/media/screw_selected.png"); selectedUpIcon:Embed("/media/screw_selected.png"); selectedDownIcon:Embed("/media/screw_selected.png"); selectedDisabledIcon:Embed("/media/screw_selected.png"); color: #1111F9; textSelectedColor: #1111F9; textRollOverColor: #1111F9;
traincorrect	overIcon:Embed("/media/train_correct.png"); upIcon:Embed("/media/train_correct.png"); downIcon:Embed("/media/train_correct.png"); disabledIcon:Embed("/media/train_correct.png"); selectedOverIcon:Embed("/media/train_correct.png"); selectedUpIcon:Embed("/media/train_correct.png"); selectedDownIcon:Embed("/media/train_correct.png"); selectedDisabledIcon:Embed("/media/train_correct.png"); color: #128212; textSelectedColor: #128212; textRollOverColor: #128212;
trainincorrect	overIcon:Embed("/media/train_incorrect.png"); upIcon:Embed("/media/train_incorrect.png"); downIcon:Embed("/media/train_incorrect.png"); disabledIcon:Embed("/media/train_incorrect.png"); selectedOverIcon:Embed("/media/train_incorrect.png"); selectedUpIcon:Embed("/media/train_incorrect.png"); selectedDownIcon:Embed("/media/train_incorrect.png"); selectedDisabledIcon:Embed("/media/train_incorrect.png"); color: #ED2919; textSelectedColor: #ED2919; textRollOverColor: #ED2919;
trainnormal	overIcon:Embed("/media/train_normal.png"); upIcon:Embed("/media/train_normal.png"); downIcon:Embed("/media/train_normal.png"); disabledIcon:Embed("/media/train_normal.png"); color: #2D2805; textSelectedColor: #2D2805; textRollOverColor: #2D2805;
trainselected	overIcon:Embed("/media/train_selected.png"); upIcon:Embed("/media/train_selected.png"); downIcon:Embed("/media/train_selected.png"); disabledIcon:Embed("/media/train_selected.png"); selectedOverIcon:Embed("/media/train_selected.png"); selectedUpIcon:Embed("/media/train_selected.png"); selectedDownIcon:Embed("/media/train_selected.png"); selectedDisabledIcon:Embed("/media/train_selected.png"); color: #1111F9; textSelectedColor: #1111F9; textRollOverColor: #1111F9;

## Page Templates

Templates are stored in the `\src\templates` directory as shown to the right. Each is an `.mxml` file that contains both the markup (e.g., the components and layout in XML format) and the code (ActionScript). You won't normally need to edit the code. Each template is discussed in detail later in this document.

From a code standpoint, each question template inherits from the `TrainingStudioTemplate` component as shown below. They (and the rest of Training Studio) then use the `TrainingStudioSupport` class, the other ActionScript classes located in the `\src\TrainingStudio` directory (shown below), and the `FlashTrainQuestion.swc` component. Note that the source code for the `FlashTrainQuestion` is not included in the project.





Content is stored in an Access, SQL Server, or XML database and associated XML files created and edited by the *Training Studio Content Editor* application. The data for the individual training pages is an XML file that matches the name of the database while in design mode () and renamed *content.xml* when published with the *Training Studio Publisher* application. Common information across the training for button captions, tooltips, question/lesson settings, and the operation of various menu options is stored in *TrainingStructure.xml*. The Glossary (if used) is stored in *Glossary.xml*. All graphics, audio, video, and other media are contained in the `\media` directory. The *start.swf* file reads all the XML files into ActionScript arrays and then provides this to the individual template files, which are responsible for populating themselves by setting their text, importing images, playing audio and video, etc.

## Common Variables

There are various properties defined in the *TrainingStudioSupport* class. However, the ones that developers would normally change are located in the *TSCCommon* class. They are defined as *static* so that an instance of the *TSCCommon* is not needed. The variables are listed below. Ones new to version 2 are shown with an \*.

<i>Variable Name</i>	<i>Description</i>
bulletChar:String = " ";	This is the first of two characters used the represent bullets in the database.
bulletChar2:String = " ";	This is the second of two characters used the represent bullets in the database.
commentCategoryArray:ArrayCollection;	This is used to store the comment categories used by the <i>Comments</i> window while within a session.
commentReviewerArray:ArrayCollection;*	This is used to store the reviewer categories used by the <i>Comments</i> window while within a session.
currentGlossaryItem:String;	This holds a reference to the glossary item the user just clicked.

## Training Studio Templates Documentation

<i>Variable Name</i>	<i>Description</i>
<code>dataExtension:String = "xml";</code>	This is the extension of the content, glossary, and training structure "XML" files. You can change to a different extension such as <code>rem</code> to avoid xml files being viewed in the browser.
<code>dbName:String;</code>	This is a short version of the <code>contentName</code> (database) used internally.
<code>defaultContentName:String = "content";</code>	// This is used if no <code>contentName</code> passed into the starting movie. Can be generic like <i>content</i> to refer to <code>content.xml</code> . This is what will be used within the Flash Builder environment. If you are testing a particular set of content within Flash Builder, you can set this to a particular file name. In that case, Training Studio will look for that directory name as a subdirectory of the content folder. For example, setting this to "Beatles" will cause Training Studio to look for the <code>\content\Beatles\Beatles.xml</code> file. It will look for <i>TrainingStructure.xml</i> , <i>Glossary.xml</i> , and the <i>media</i> directory in <code>\content\Beatles</code> .
<code>defaultScormVersion:ScormVersionEnum = new ScormVersionEnum(ScormVersionEnum.None);*</code>	This is the SCORM/AICC version to use for the lesson. Valid values are <i>ScormVersionEnum.None</i> , <i>ScormVersionEnum.Aicc</i> , <i>ScormVersionEnum.Version_12</i> , and <i>ScormVersionEnum.Version_13</i> . SCORM can be turned on with the query string ( <code>?scormversion=12</code> or <code>?scormversion=13</code> ) but cannot be turned off that way for security reasons. Similarly, AICC can be turned on with the query string ( <code>?useAicc=true</code> ) but cannot be turned off that way for security reasons.
<code>disableMenu:Boolean = false;</code>	This can set to true from within Training Structure or here. You might do this for an exam.
<code>enableNextBtn:Boolean = false;</code>	This stores whether the <i>Next</i> button is enabled.
<code>enablePrevBtn:Boolean = false;</code>	This stores whether the <i>Previous</i> button is enabled.
<code>hotObjectsRotateAngle:Number = 10;*</code>	This determines the angle which answers on the <i>question_hotObjects.mxml</i> template rotate when selected.
<code>hyperlinkChar:String = "~";</code>	This is the delimiter for hyperlinks. You could change it from <code>~</code> if that character will occur in your training content.
<code>hyperlinkColor_External:String = "9B7B0B";*</code>	This is the color for hyperlinks that show external files in hexadecimal format.
<code>hyperlinkColor_Glossary:String = "FF0000";*</code>	This is the color for "glossary" hyperlinks in hexadecimal format.
<code>hyperlinkColor_Pages:String = "0000FF";*</code>	This is the color for hyperlinks that jump to other pages in hexadecimal format.
<code>includePageNumbersOnMenu:Boolean = true;*</code>	Set this to true to have page numbers go before each title on both the popup menu and the menu window. For example, if page 1 has a title of "Introduction," setting this to true will cause the menu item to look like this: 1. Introduction
<code>includeTBKTrackerStudentInfoLink:Boolean = false;</code>	This determines whether to add a "Student Information" link to the <i>Options</i> button. This link in turn sends the <i>ShowStudentInfo</i> message to the container application using Flash's <i>ExternalInterface</i> class. It is intended for use by the Platte Canyon® TBK Tracker™ Learning

## Training Studio Templates Documentation

<i>Variable Name</i>	<i>Description</i>
	Management System, but you are welcome to handle that message in another projector.
intervalId:Number = 0;*	This is a timer identifier used in <i>timedBullet_1000</i> and other templates.
masterContentArray:ArrayCollection= new ArrayCollection();	This is an array of “dictionaries” with the index being the page number – 1 due to the fact that the array starts from 0. Each element is an associative array. The current page’s dictionary is represented by the <i>pageArray</i> variable.
maxWidth:Number = 800;	This is the width of the <i>start.mxml</i> . It is used primarily to calculate the position of the popup menu.
menuArrayCollection:ArrayCollection;*	This is used to store the page names and numbers for display by the popup menu.
menuHeightPerRow:int = 19;*	This is used to calculate the position of the popup menu. You might edit this if the menu goes off the screen for some reason.
menuWidth:int = 350;*	This is the default width for the popup menu. It is calculated once the menu is displayed once. If you are unhappy with the initial size of the menu, you can try editing this value. A bigger value moves the menu to the left (only the first time the user clicks on it).
navMenuId:mx.controls.Menu;	This is used for popup menu if there are not too many items ( <i>numPagesToUsePopupMenu</i> ).
numOptionsInMenu:int;*	This is set in the initialization code to help position the Options menu
numPages:Number;	Holds the number of pages of training, which comes from the number of records in the database.
numPagesToUsePopupMenu:Number = 30;	After this number of pages, we use the popup menu window rather than a <i>Menu</i> component.
obfuscateContent:Boolean = false;*	Set this true if you are obfuscating your content in the content.xml file. You will also need to set a matching value in the <i>TrainingStudioConfigurationEditor.exe.config</i> file.
pageArray: Dictionary;	This is the dictionary representing the content for the current page. It is used by all <i>templates</i> and elsewhere within Training Studio.
pageNum:Number; // set to 1 if not a different value from the LMS	Holds the current page number of the training.
randomizeAnswers:Boolean = true;*	This sets whether answers on the question templates are randomized.
reviewKey:String = "1111111111";*	This is the key that allows your lessons to send comments to the <i>TrainingStudioFlexService</i> . That service needs to have a matching value in its <i>Web.config</i> file.
reviewOn:Boolean = true;	Setting this to false turns off the ability for the user to launch the <i>Comments</i> window via pressing Ctrl+Shirt+M.
reviewerContactNumber:String;	This is reviewer information for the <i>Comments</i> window used within a session.
reviewerEmailAddress:String;	This is reviewer information for the <i>Comments</i> window used within a session.
reviewerFirstName:String;	This is reviewer information for the <i>Comments</i> window used within a session.
reviewerLastName:String;	This is reviewer information for the <i>Comments</i> window

<i>Variable Name</i>	<i>Description</i>
	used within a session.
reviewerOrganization:String;	This is reviewer information for the <i>Comments</i> window used within a session.
sendExamScoresOnlyWhenReachScorePage:Boolean = false;	This can be set in the <i>Training Structure</i> or here to only send an exam score if the student reaches an <i>examScore</i> template page.
soundOn:Boolean = true;	This stores whether the sound for audio files is currently on.
underlineHyperlinks:Boolean = true;	Change to false to have hyperlinks just colored without an underline.

## Template Layout

As discussed earlier in this document, templates are shown in the *ViewStack* component named “transition” on the *start.mxml* application. You can change its position if desired. Each template is its own *mxml* file stored in the `\src\templates` director and compiled into *start.swf*. It corresponds to a *template* selected by the user of the *Training Studio Content Editor* application and stored in the database. When a student goes to a new training page, the *start.swf* determines the template name and loads that template component, which is then responsible for loading its content, images, and media. Templates are typically independent of content and user interface. There is a one to one correspondence between the column names in the database (nodes in the XML file) and the object names in the templates. Objects with “0” at the end are typically shown immediately without interaction. Objects named “1” through “10” are shown in response to interactions. Clicking or rolling over *hotspot\_1*, for example, shows *content\_1*, displays or plays *graphic\_1* and/or *media\_1* (can hold media or even PDF and other document). Content shows in the *DisplayField* while common images show in the *graphic\_0* movie clip and common .swf media shows in the *media\_0* movie clip. Any .flv files play in the one *QVideo* component named *MediaPlayer*.

## Individual Templates

We will now go through each individual template. As mentioned above, we recommend using *Flash Builder* for editing templates and updating *start.swf*.

One thing to watch out for when adding templates is that they need to be referenced in *start.mxml*. This is necessary so that the “transition” *ViewStack* control can programmatically load the template. That is the reason for this set of code in *start.mxml*:

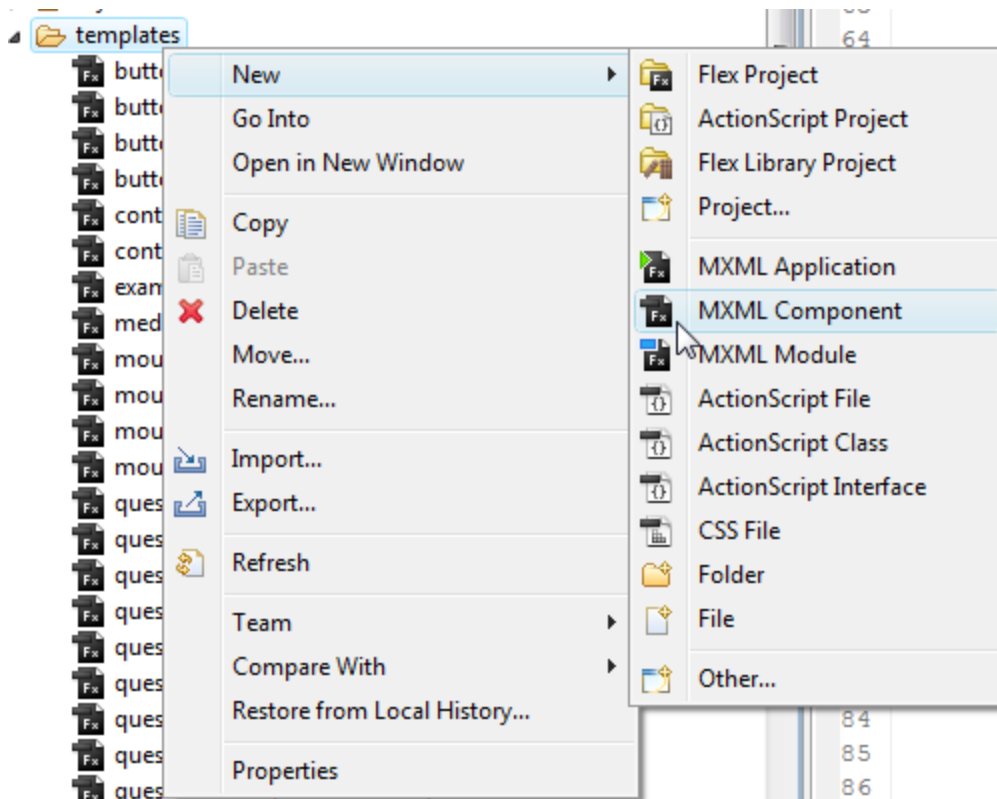
```
// following are just so getDefinitionByName() works
private var _buttonClickImageLeftShowContentMedia:buttonClickImageLeftShowContentMedia;
private var _buttonClickImageRightShowContentMedia:buttonClickImageRightShowContentMedia;
private var _buttonClickLeftShowContentImageMedia:buttonClickLeftShowContentImageMedia;
private var _buttonClickRightShowContentImageMedia:buttonClickRightShowContentImageMedia;
private var _contentLeftImageRight:contentLeftImageRight;
private var _contentRightImageLeft:contentRightImageLeft;
private var _examScore:examScore;
private var _mediaFullScreen:mediaFullScreen;
private var _mouseEnterLeftShowContentImage:mouseEnterLeftShowContentImage;
private var
_mouseEnterLeftShowContentImageClickShowLink:mouseEnterLeftShowContentImageClickShowLink;
private var _mouseEnterRightShowContentImage:mouseEnterRightShowContentImage;
private var
_mouseEnterRightShowContentImageClickShowLink:mouseEnterRightShowContentImageClickShowLink;
```

## Training Studio Templates Documentation

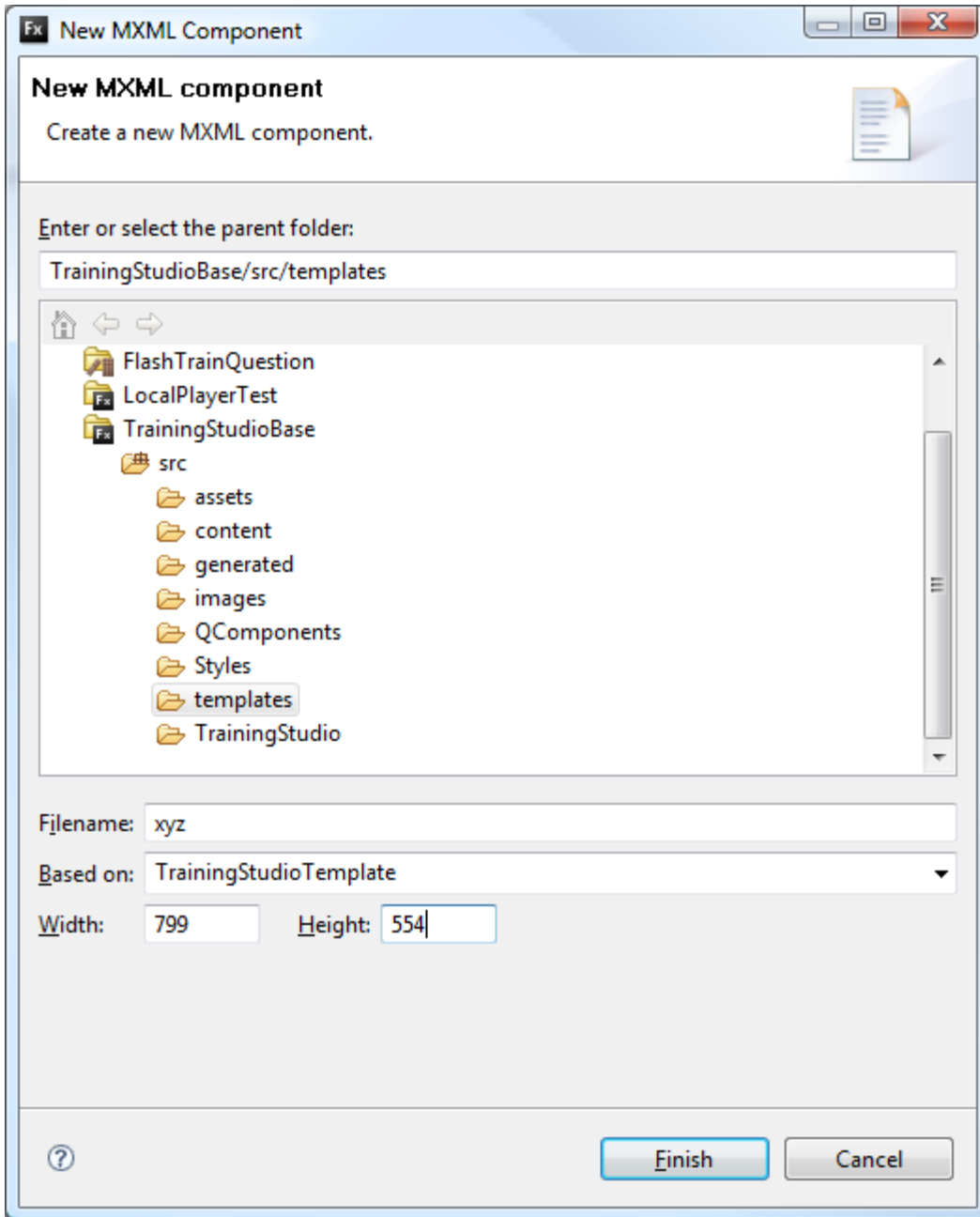
```
private var _question_dragAndDrop:question_dragAndDrop;  
private var _question_fillIn:question_fillIn;  
private var _question_hotObjects:question_hotObjects;  
private var _question_hotSpot:question_hotSpot;  
private var _question_hotSpotImageText:question_hotSpotImageText;  
private var _question_matchingLine:question_matchingLines;  
private var _question_multipleChoice:question_multipleChoice;  
private var _question_multipleChoiceButtons:question_multipleChoiceButtons;  
private var _question_multipleChoiceImageText:question_multipleChoiceImageText;  
private var _question_numeric:question_numeric;  
private var _question_sequencing:question_sequencing;  
private var _question_sequencingImages:question_sequencingImages;  
private var _question_trueFalse:question_trueFalse;  
private var _showContentWhenMediaComplete:showContentWhenMediaComplete;  
private var _static1:static1;  
private var _timedBullet_1000:timedBullet_1000;  
private var _timedBullet_1500:timedBullet_1500;  
private var _timedBullet_500:timedBullet_500;
```

If you want to add a template named “xyz,” you would follow these steps.

1. Right-click on the templates directory within *Flash Builder* and select a new MXML Component as shown below.



2. Give it a name (xyz.xml) and select *TrainingStudioTemplate* as the object that it is “Based on.” Set the desired size (799 x 554 by default). This is shown below.



3. Copy an existing template like *contentLeftImageRight.xml* and paste its contents into *xyz.xml*. You can then edit it as needed.

4. Add a variable in *start.xml* like the ones above:

```
private var _xyx:xyz;
```

You would then *build* the project and copy the *start.swf* file from the *\flex\_bin* directory to your “TrainingStudioSource” directory.

## Template ActionScript and MXML

Templates are responsible for populating themselves. Both the code and the objects are located in the MXML file. *buttonClickImageLeftShowContentMedia.xml* is shown below. Others are quite similar. Explanations are in orange.

```
<?xml version="1.0" encoding="utf-8"?>
<TrainingStudioTemplate xmlns="TrainingStudio.*"
xmlns:mx="http://www.adobe.com/2006/mxml" backgroundAlpha="0.0" width="799"
height="554" xmlns:TrainingStudio="TrainingStudio.*"
xmlns:ex="Components.*" xmlns:media="flash.media.*" xmlns:ns1="QComponents.*"
creationComplete="loadData(event)">
```

This shows that the component inherits from *TrainingStudioTemplate*. The *backgroundAlpha* means that it is transparent, allowing *start.mxml* to show through. The various *xmlns:* elements reference various namespaces for the mxml components at the bottom of the file. The *creationComplete* element means that the *loadData* function will be called as soon as the template has been fully created.

```
<mx:Style source="/Styles/TrainingStudioStyles.css"></mx:Style>
```

This line references the *TrainingStudioStyles.css* file. You could add additional style sheets if desired. The leading / makes it relative from the *src* directory rather from the */src/templates* directory.

```
<mx:Script>
<![CDATA[
import flash.utils.Dictionary;
import TrainingStudio.*;
import mx.controls.Image;
```

This is where all the ActionScript for the template is located. The *import* lines reference namespaces where code is located. The *TrainingStudio.\** line allows the template to work with items in the *TrainingStudio* directory.

```
// private variables
private var supportRefId:TrainingStudioSupport =
start(mx.core.Application.application).supportRefId;
```

The line above is quite important is contained in all templates. *supportRefId* is a reference to an instance of the *TrainingStudioSupport* class. This is where most of the shared functionality of Training Studio is contained.

```
private var lastHotspot:Image = null;
private var completionArray:Array;
private var numHotspots:int = 0;
```

A *private* variable is one that cannot be seen or referenced from outside this object (the current template). These ones are used to keep track of completion and the "last" hotspot accessed (in order to set its style to "ImageHotspotCompleted."

```
// event handlers

private function loadData(e:Event):void {
var pageArrayLocal:Dictionary = TSCommon.pageArray; // associative array
```

## Training Studio Templates Documentation

The `pageArrayLocal` variable is a *Dictionary* that represents the current training page. The key matches up to the node or column in the database. They are *title*, *subtitle*, *content\_0*, etc.

```
var keyId:Object;
var keyString:String;
var keyValue:String;
var hotspotNum:int;

for (keyId in pageArrayLocal) {
    keyString = keyId.toString();
    keyValue = pageArrayLocal[keyId].toString();

    // handle unique ones here. Handle the rest in TrainingStudioTemplate
```

We loop through each of the keys and only handle the ones that need special handling by this template.

```
switch(keyString) {

    // let "media_0" get handled by template
    case "media_1":
    case "media_2":
    case "media_3":
    case "media_4":
    case "media_5":
    case "media_6":
    case "media_7":
    case "media_8":
    case "media_9":
    case "media_10":
        break;
```

We let the *TrainingStudioTemplate* class handle *media\_0*, since we want any initial sound, video, or Flash movie to play. The rest of the media (*media\_1* - *media\_10*) is only played in response to the hotspot interaction. So we don't do anything except *break* when we encounter them.

```
// let graphic_0 be handled by template
case "graphic_1":
case "graphic_2":
case "graphic_3":
case "graphic_4":
case "graphic_5":
case "graphic_6":
case "graphic_7":
case "graphic_8":
case "graphic_9":
case "graphic_10":
    var hotspotId:Image = Image(this[keyString]);

    supportRefId.PopulateImage(hotspotId, keyValue, true);
    hotspotId.visible = true;
    hotspotId.alpha = TrainingStudioSupport.hotObjectsAlpha;
    hotspotNum = supportRefId.getFieldNum(keyString);
```

## Training Studio Templates Documentation

```
numHotspots = Math.max(hotspotNum, numHotspots);  
  
break;
```

Again, we let the *TrainingStudioTemplate* class handle *graphic\_0*, since we want any initial graphic to be displayed. The rest of the graphics (*graphic\_1* - *graphic\_10*) are used for the actual hotspots in this template. We first build a reference to the associated *Image* object using the *Image(this[keyString])* syntax. We call the *PopulateImage()* method of the *supportRefId* instance of our *TrainingStudioSupport* class. We then show the image and set its *alpha* (transparency)<sup>7</sup>. Finally, strip the number (1, 2, 3, etc.) from then name of the object and use it to populate our *numHotspots* variable. We use this to determine when all the hotspots have been selected.

```
default:  
    super.PopulateTemplate(supportRefId, this, keyString, keyValue);
```

This line is where all the keys that we didn't specifically handle are sent to the parent (*super* in ActionScript terminology<sup>8</sup>). In some templates, there is no unique handling at the template level.

```
    }  
}  
  
completionArray = new Array((numHotspots - 1));
```

We use the *numHotspots* variable to create an associated *completionArray*. We will use this array below.

```
    }  
  
    private function hotspotClickHandler(e:MouseEvent):void {
```

This function is called in response to the *mouseDown* event in the mxml below<sup>9</sup>.

```
        var targetId:Image = Image(e.currentTarget); // use currentTarget rather  
        than target for Image controls
```

We use either the *target* or *currentTarget* property of our *MouseEvent* object to figure out *which* image the user interacted with.

```
        var hotspotName:String = targetId.name;  
        var hotspotNum:int = supportRefId.getFieldNum(hotspotName);
```

We grab the name and then the associated number in order to work our naming scheme. *hotspot\_1* goes with *media\_1*, *graphic\_1*, and *content\_1* and so on.

```
        if (lastHotspot != null) {
```

---

<sup>7</sup> Note that you can edit the *hotObjectsAlpha* variable in *TrainingStudioSupport.as* if desired. This is a *static* or shared variable, which is why we reference it by the class name (*TrainingStudioSupport*) rather than the *instance* name (*supportRefId*).

<sup>8</sup> This is roughly equivalent to forwarding a message in ToolBook® OpenScript®. In Visual Basic®.NET, you would use *MyBase*.

<sup>9</sup> Changing from a *mouseDown* or *click* interaction to a *rollover* interaction is as simple as changing the event name in the MXML.

## Training Studio Templates Documentation

```
        lastHotspot.styleName = "ImageHotspotCompleted";
    }
```

The first time through, the *lastHotspot* variable will be null. After that, it will refer to the hotspot previous to this interaction. In that case, we set its *styleName* to "ImageHotspotCompleted." This is how we get it to turn blue or otherwise show completion.

```
lastHotspot = targetId;
```

We set the *lastHotspot* variable so we'll be able to change its *styleName* the next time through.

```
targetId.styleName = "ImageHotspotCurrent";
targetId.alpha = 1;
```

We change the *styleName* of this hotspot to "ImageHotspotCurrent" to denote which one we are currently looking at.

```
completionArray[hotspotNum - 1] = true;
```

We set the associated element<sup>10</sup> of our *completionArray* to "true." Once all the elements are "true," the page is completed.

```
supportRefId.StopFlashMovies(media_0, media_1, media_2, media_3, media_4,
media_5, media_6, media_7, media_8, media_9, media_10);
// can change above to StopMedia if want sound and video stopped too.
```

We call the *StopFlashMovies* method to stop any Flash movies that are currently playing.<sup>11</sup>

```
supportRefId.showTextImageMedia(DisplayField, targetId.name, graphic_0,
media_0, sound_0, MediaPlayer, true, false);
// include media but don't hide video
```

We call the *showTextImageMedia* to display the associated content, play any associated media, and display any associated graphic. Note that we pass the object references to display the content (*DisplayField*), show the graphic (*graphic\_0*), or play the media (*media\_0*, *sound\_0*, or *MediaPlayer*). The parameters at the end determine whether to include media and whether to hide any current video.

```
supportRefId.getHotspotCompleted(completionArray, numHotspots);
```

We pass our *completionArray* and the *numHotspots* variable to the *getHotspotCompleted* method. This will show the "goNext.swf" file in the *CompletionImage* placeholder if all the interactions are completed.

```
}
```

```
// overrides
override public function StopMedia():void {
    supportRefId.StopAllMedia(this.MediaPlayer, this.media_0, this.media_1,
this.media_2, this.media_3, this.media_4, this.media_5, this.media_6,
```

---

<sup>10</sup> We subtract 1 since arrays are zero-based. So hotspot 1 corresponds with element 0 and so forth.

<sup>11</sup> Note the comment that points out that we can *StopMedia* if desired. This is defined later in the code. Note that the ability to stop Flash movies depends on the version of the Flash player on the user's machine.

## Training Studio Templates Documentation

```
        this.media_7, this.media_8, this.media_9, this.media_10);  
    }
```

This *overrides* the same function in the *TrainingStudioTemplate*. Each template overrides this function since it needs to pass references to the individual media objects.

```
    ]]>  
</mx:Script>
```

The rest of the template contains the MXML of the objects themselves. You would normally only need to edit the *x*, *y*, *width*, and/or *height*. Note that the layer order is from the top down. So if you want one object to be on top of another, put it earlier in the MXML.

```
<mx:Text x="8" y="8" id="title" styleName="Title" width="642" height="32"  
link="linkHandler(event)"/>
```

We handle the *link* event to enable hyperlinks. The *linkHandler* function is defined in *TrainingStudioTemplate.mxml*.

```
<mx:Text x="8" y="44" id="subtitle" styleName="Subtitle" width="779" height="32"  
link="linkHandler(event)"/>  
<mx:TextArea x="328" y="96" id="DisplayField" styleName="ContentTransparent "  
width="440" height="147" link="linkHandler(event)" editable="false"/>  
<mx:TextArea x="8" y="495" id="content_0" styleName="ContentTransparent "  
width="760" height="55" link="linkHandler(event)" editable="false"/>  
<mx:SWFLoader x="328" y="242" id="graphic_0" visible="false"/>
```

Notice that many of these objects have their *visible* property set to false. This keeps them from flashing or showing a broken link when they are not used by a particular page.

```
<mx:SWFLoader x="464" y="248" id="media_0" visible="false"/>  
<ns1:QVideo x="464" y="248" width="320" height="240" id="MediaPlayer"  
visible="false">  
</ns1:QVideo>
```

You'll likely want to change the *width* and *height* of the *MediaPlayer* to match the most common dimensions of your video.

```
<ns1:QImage x="24" y="96" id="graphic_1" styleName="ImageHotspot "  
buttonMode="true" useHandCursor="true" mouseDown="hotspotClickHandler(event)"  
visible="false" scaleContent="true" />
```

Notice that we handle the *mouseDown* event since this is "buttonClick" interaction. All of the hotspots call the same *hotspotClickHandler* function shown above.

```
<ns1:QImage x="24" y="242" id="graphic_2" styleName="ImageHotspot "  
buttonMode="true" useHandCursor="true" mouseDown="hotspotClickHandler(event)"  
visible="false" scaleContent="true" />  
<ns1:QImage x="24" y="388" id="graphic_3" styleName="ImageHotspot "  
buttonMode="true" useHandCursor="true" mouseDown="hotspotClickHandler(event)"  
visible="false" scaleContent="true" />  
<ns1:QImage x="164" y="96" id="graphic_4" styleName="ImageHotspot "  
buttonMode="true" useHandCursor="true" mouseDown="hotspotClickHandler(event)"  
visible="false" scaleContent="true" />
```

## Training Studio Templates Documentation

```
<ns1:QImage x="164" y="242" id="graphic_5" styleName="ImageHotspot"
buttonMode="true" useHandCursor="true" mouseDown="hotspotClickHandler(event)"
visible="false" scaleContent="true" />
<ns1:QImage x="164" y="388" id="graphic_6" styleName="ImageHotspot"
buttonMode="true" useHandCursor="true" mouseDown="hotspotClickHandler(event)"
visible="false" scaleContent="true" />

<mx:SWFLoader x="0" y="0" id="media_1" visible="false" />
<mx:SWFLoader x="0" y="0" id="media_2" visible="false" />
<mx:SWFLoader x="0" y="0" id="media_3" visible="false" />
<mx:SWFLoader x="0" y="0" id="media_4" visible="false" />
<mx:SWFLoader x="0" y="0" id="media_5" visible="false" />
<mx:SWFLoader x="0" y="0" id="media_6" visible="false" />
<mx:SWFLoader x="0" y="0" id="media_7" visible="false" />
<mx:SWFLoader x="0" y="0" id="media_8" visible="false" />
<mx:SWFLoader x="0" y="0" id="media_9" visible="false" />
<mx:SWFLoader x="0" y="0" id="media_10" visible="false" />
<mx:TextArea x="0" y="0" id="content_1" styleName="ContentTransparent"
link="linkHandler(event)" editable="false" visible="false" />
<mx:TextArea x="0" y="0" id="content_2" styleName="ContentTransparent"
link="linkHandler(event)" editable="false" visible="false" />
<mx:TextArea x="0" y="0" id="content_3" styleName="ContentTransparent"
link="linkHandler(event)" editable="false" visible="false" />
<mx:TextArea x="0" y="0" id="content_4" styleName="ContentTransparent"
link="linkHandler(event)" editable="false" visible="false" />
<mx:TextArea x="0" y="0" id="content_5" styleName="ContentTransparent"
link="linkHandler(event)" editable="false" visible="false" />
<mx:TextArea x="0" y="0" id="content_6" styleName="ContentTransparent"
link="linkHandler(event)" editable="false" visible="false" />
<mx:TextArea x="0" y="0" id="content_7" styleName="ContentTransparent"
link="linkHandler(event)" editable="false" visible="false" />
<mx:TextArea x="0" y="0" id="content_8" styleName="ContentTransparent"
link="linkHandler(event)" editable="false" visible="false" />
<mx:TextArea x="0" y="0" id="content_9" styleName="ContentTransparent"
link="linkHandler(event)" editable="false" visible="false" />
<mx:TextArea x="0" y="0" id="content_10" styleName="ContentTransparent"
link="linkHandler(event)" editable="false" visible="false" />
<ns1:QImage x="0" y="0" id="graphic_7" styleName="ImageHotspot" buttonMode="true"
useHandCursor="true" mouseDown="hotspotClickHandler(event)" scaleContent="false" />
<ns1:QImage x="0" y="0" id="graphic_8" styleName="ImageHotspot" buttonMode="true"
useHandCursor="true" mouseDown="hotspotClickHandler(event)" visible="false"
scaleContent="true" />
<ns1:QImage x="0" y="0" id="graphic_9" styleName="ImageHotspot" buttonMode="true"
useHandCursor="true" mouseDown="hotspotClickHandler(event)" visible="false"
scaleContent="true" />
<ns1:QImage x="0" y="0" id="graphic_10" styleName="ImageHotspot"
buttonMode="true" useHandCursor="true" mouseDown="hotspotClickHandler(event)"
visible="false" scaleContent="true" />
<media:Sound id="sound_0" />
<media:Sound id="sound_1" />
<media:Sound id="sound_2" />
<media:Sound id="sound_3" />
<media:Sound id="sound_4" />
<media:Sound id="sound_5" />
<media:Sound id="sound_6" />
<media:Sound id="sound_7" />
<media:Sound id="sound_8" />
```

```
<media:Sound id="sound_9" />
<media:Sound id="sound_10" />
</TrainingStudioTemplate>
```

## Deployment

The expected way to deploy training is to use the *Training Studio Publisher* application to create a SCORM package, AICC package, or CD directory structure where the user simply launches the start.html file. When using the Publisher, the directory structure looks like this:

```
start.html
start.swf
other files
  content (directory)
    content.xml
    TrainingStructure.xml
    Glossary.xml
    Media (directory)
      all media files
```

However, it is possible to deploy in a modified directory structure where you have a single *start.swf* that is shared by multiple lessons. In that case, the directory structure looks something like this:

```
start.html
start.swf
other files
  content (directory)
    Lesson1 (directory)
      Lesson1.xml12
      TrainingStructure.xml
      Glossary.xml
      Media (directory)
        all media files
    Lesson2 (directory)
      Lesson2.xml
      TrainingStructure.xml
      Glossary.xml
      Media (directory)
        all media files
  More Lessons
```

In this situation, you need to pass the desired information via the query string. For example, this URL will launch the training with the *BeatlesExam\_en* database and a SCORM version of 1.2.

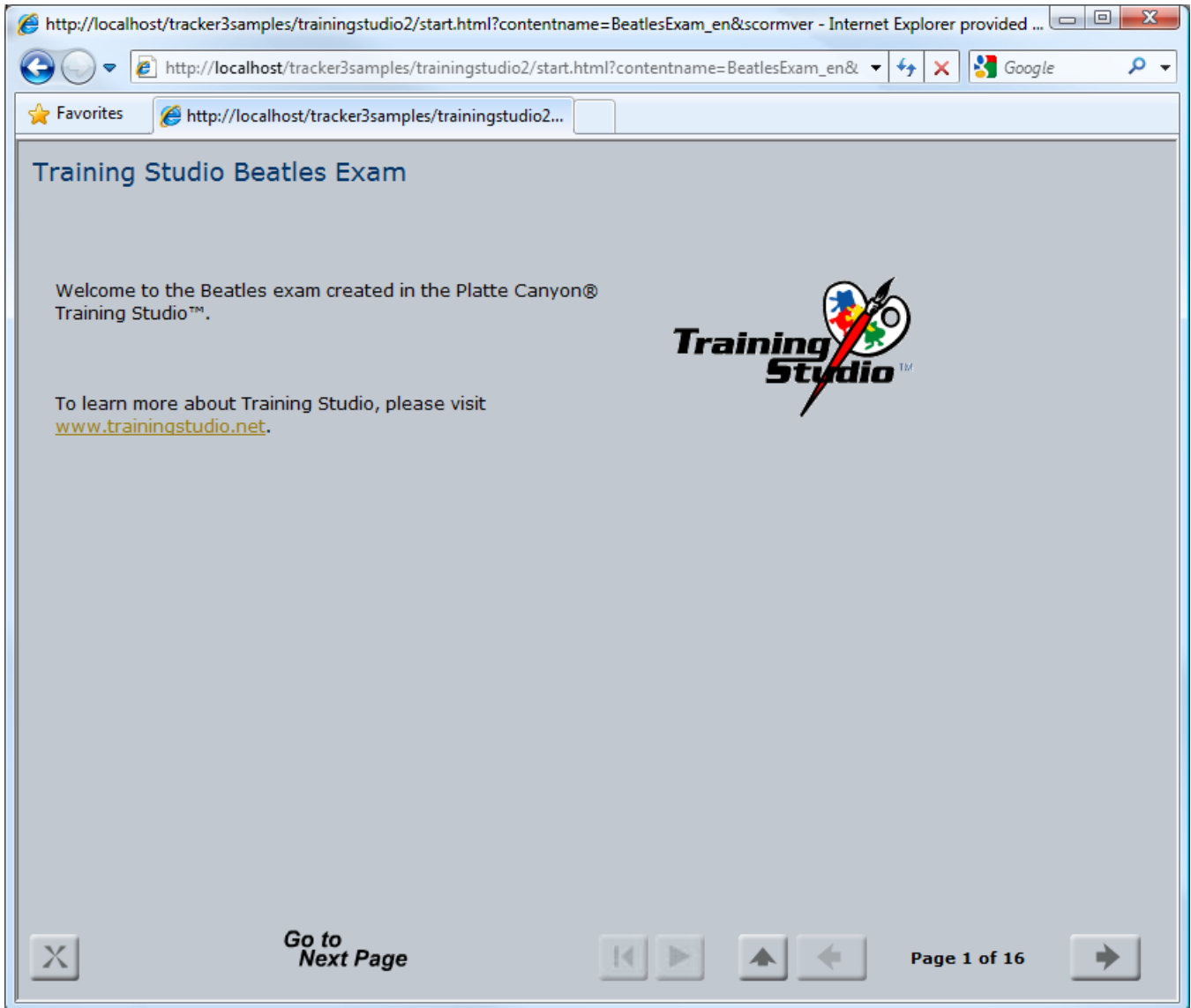
---

<sup>12</sup> Note that the directory name and the xml file name must match when using this deployment method. That is because both are read from the *contentname* query string parameter.

## Training Studio Templates Documentation

`http://localhost/tracker3samples/trainingstudio2/start.html?contentname=BeatlesExam_en&scormversion=12`

The resulting window is shown below:



If you deploy on a web server and can put your content in a directory such as *training*, the path would look more like this:

`../training/start.html?contentname=BeatlesExam_en&scormversion=12`

This technique is a useful way to test your content without having to publish first via *Training Studio Publisher*.