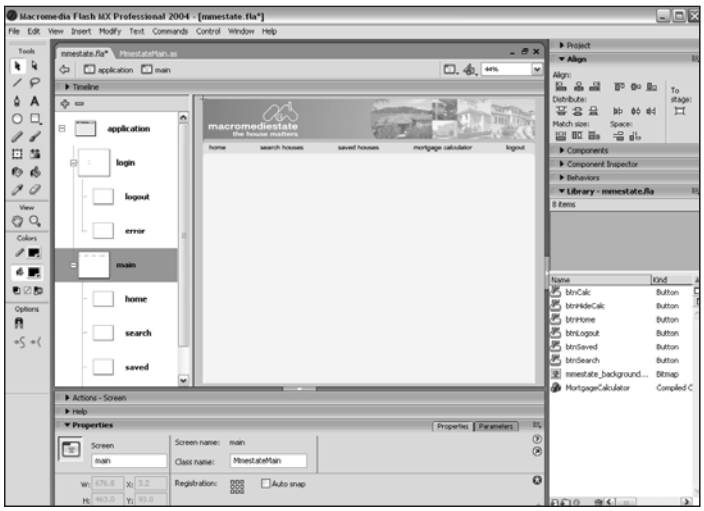
7 Building Applications with Screens

You have created a one-page application and turned it into a component. But, what do you do if you want to build a multipage application, one in which you can move between displays in a nonlinear fashion? One option is to build your application using screens, a programming metaphor new to Macromedia Flash MX 2004 and unique to the Professional version. Screens give you the ability to visually organize your application pages in the authoring environment. A second option is to create each unique application page or functionality as its own MovieClip or SWF that can be loaded into a main application SWF. This method will be covered in Lesson 11, "Creating Visual Objects Dynamically." Finally, an old technique, not used in this book, is to use separate frames of the Timeline to represent different application states. You then use Timeline functions to move between different frames in the Timeline.

Screens are part of the new application-building framework introduced in Flash MX Professional 2004 to aid in rapid application development. This framework consists of various classes, components, and development tools to provide additional functionality over that native to the Flash Player. The main features of the application framework include screens, user interface components, data connector components, behaviors, and built-in data binding. In this lesson, you learn to create applications with screens. In Lesson 8, "Using the Flash Application Framework," you are introduced to the other pieces of the application framework.



In this lesson, you lay out the structure of the Macromediestate application using screens and create the navigation to enable movement between the application pages.

Up to now, you worked on the MortgageCalculator component. Now that you have finished that project, you need a new one. From here on out, you will build a web site for Macromediestate, the fictitious real estate company that was introduced in Lesson 1, "Learning the Flash Interface." The finished Macromediestate application will include login/logout functionality, a home page displaying dynamic data, an interface to search and display homes for sale, the ability for the user to save his or her favorite homes, and the integration of the mortgage calculator you built in the first part of the book.

What You Will Learn

In this lesson, you will:

- Create a Form-based application using screens
- Set the visibility of a screen in the authoring environment
- Set the initial runtime visibility of a screen
- Architect an application with screens
- Look at the Form screen API
- Place code in the Timeline of a screen
- Place code on a screen object
- Place code in a class associated with a screen

Approximate Time

This lesson takes approximately 70 minutes to complete.

Lesson Files

Asset Files:

/fpad2004/lesson07/start/assets/mmestate_background.png
/fpad2004/lesson07/start/assets/navbuttons.fla

Starting Files:

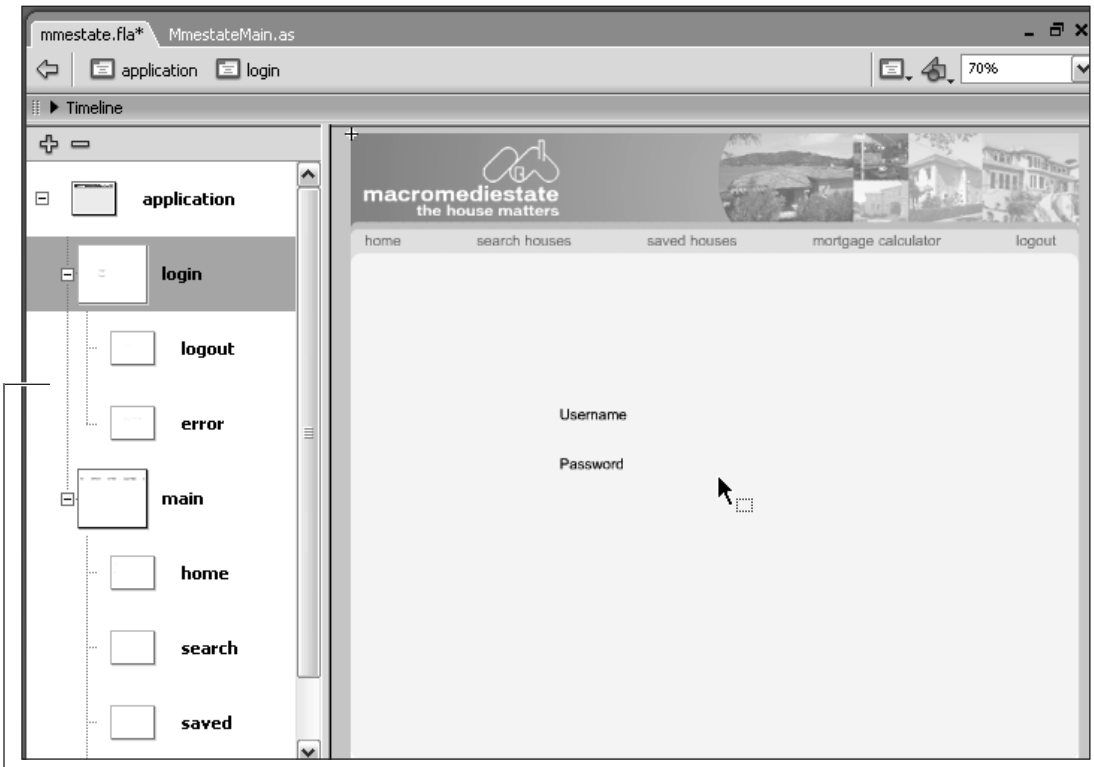
None

Completed Files:

/fpad2004/lesson07/complete/mmestate.fla
/fpad2004/lesson07/complete/classes/MmestateMain.as

Creating a Form-Based Application

In the last lesson, you learned how to link a MovieClip (which contains the visual elements) to a class file (which contains the code to manipulate the visual elements). This is exactly what a screen is, a MovieClip with a linked class file, with the additional functionality of a thumbnail version of the screen displayed in a Screen Outline pane in the authoring environment. This special pane appears only for screens when you create a new screens-based application. This Screen Outline pane allows you to visually architect and manipulate the structure of a multipage application.



Screen Outline pane

There are two types of screens-based applications: form screens and slide screens.

A **Flash Form Application** lets you create structured screens-based applications but does not provide any framework for navigating between the screens. A Flash Form Application is ideal for creating typical, rich Internet applications that have nonlinear navigation. You lay out and organize the content in different screens (which can be overlaid or nested) and then you link the screens together, rolling your own navigation using ActionScript. Form-based applications use the form screen as the default screen type.

A **Flash Slide Presentation** has built-in functionality for linear navigation between the screens. It is ideal for creating Flash documents containing sequential content, such as a slide show or a multimedia presentation. By default, you navigate between slide screens using the keyboard's arrow keys. You can

design the screen navigation so that screens replace or overlay one another when the next slide is viewed. Slide-based applications use the slide screen as the default screen type.

Note *You can also mix slide screens and form screens within one screens-based application, allowing you to create even more complex structures in a presentation or application.*

Both types of screens descend from the built-in MovieClip class, but have additional functionality with many more properties and methods defined in their associated classes (mx.screens.Form or mx.screens.Slide) and the classes they inherit from. Screens are actually based on the same architecture as the prebuilt components mentioned briefly in the last lesson, inheriting from the UIObject and UIComponent classes. You learn more about the architecture and functionality defined for the prebuilt components in Lesson 9, "Learning the UI Component Framework."

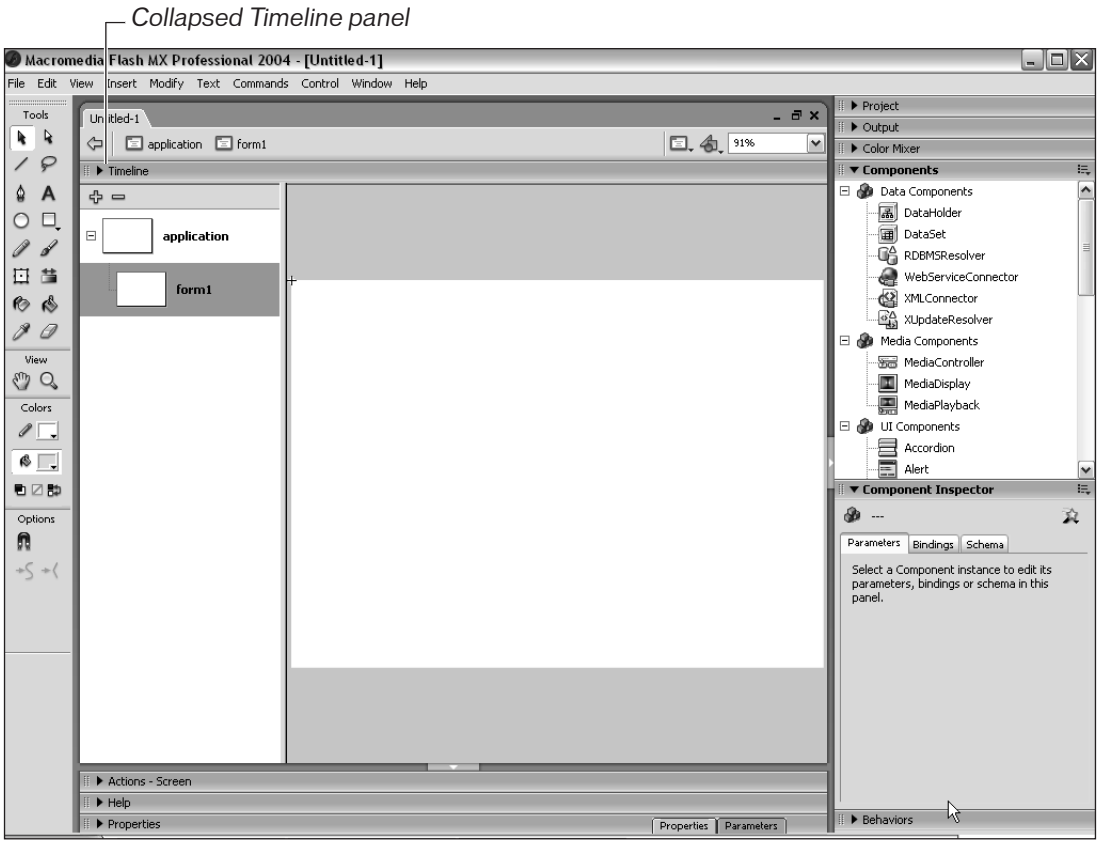
To create a screens-based application, you select Flash Form Application or Flash Slide Presentation when you create a new document. You must create a screens-based application from scratch; you cannot convert an existing application. After you create a screens-based application, you manipulate the screens using the Screen Outline pane where you can add, delete, and rename screens—as well as change their position relative to other screens or nest them within other screens. Each screen is a unique instance that cannot be reused by any other application. You can set parameters for a screen in the Property inspector. You can also use ActionScript to manipulate and control screens.

Note *Screen instances do not appear in the Library panel and cannot be reused.*

In this exercise, you create a new Form-based application and begin creating the Macromediestate application. You import a background image that is used for the entire application and also begin creating a `login` screen, which will be the first screen a user sees. You create nested `logout` and `error` screens, which overlay messages on the `login` screen when the user has logged out or entered incorrect information. In this exercise, you add text and image content to the screens.

1. Select File › New. In the New Document dialog box, select Flash Form Application and click OK.

The new Form application document opens in an authoring environment that looks different from the normal Flash documents that you saw in the previous six lessons. The Timeline panel is collapsed and there is a new Screen Outline pane on the left side of the Stage.



2. Click the `application` screen in the Screen Outline pane.

The `application` screen is the root screen or container for the application. You can change its name, but you cannot delete it. The `application` screen is the parent screen; all other screens must be children screens. You place content you want to appear on every screen on the `application` screen.

3. Select Modify › Document. In the Document Properties dialog box, make the document *700* px (pixels) wide and *600* px (pixels) tall. Click OK.

You are matching the document size to the size of the background image you will import for the `application` screen in the next step. By default, all child Form screens have the same size and same registration point (in the upper left corner) as the `application` screen. You can change the size and location of an individual Form screen with code.

4. Select File › Import › Import to Stage. In the Import dialog box, navigate to mmestate_background.png in /fpad2004/mmestate/assets/. Click Open (Windows) or Import (Macintosh). Select "Import as single flattened bitmap" and then click OK.

Whatever you place in the `application` screen will be visible on all children screens.

5. Click the `form1` screen in the Screen Outline pane.

The background image you imported should become grayed-out, indicating that it is on a screen that no longer has focus and cannot be edited.

6. Double-click `form1` in the Screen Outline pane and rename it *login*.

The name you specify in the Screen Outline pane is also used as the instance name and linkage name for the screen object. For this reason, it is a best practice to name your screen instance using all lowercase or camelcase, exactly as you have been doing for other class instances.

7. On the Stage for the `login` screen, create a static text field with the text *Username* using Arial font, black, size 14.

In this exercise, you add only the text that will appear on the `login` screen. You add the input fields and a button in the next lesson.

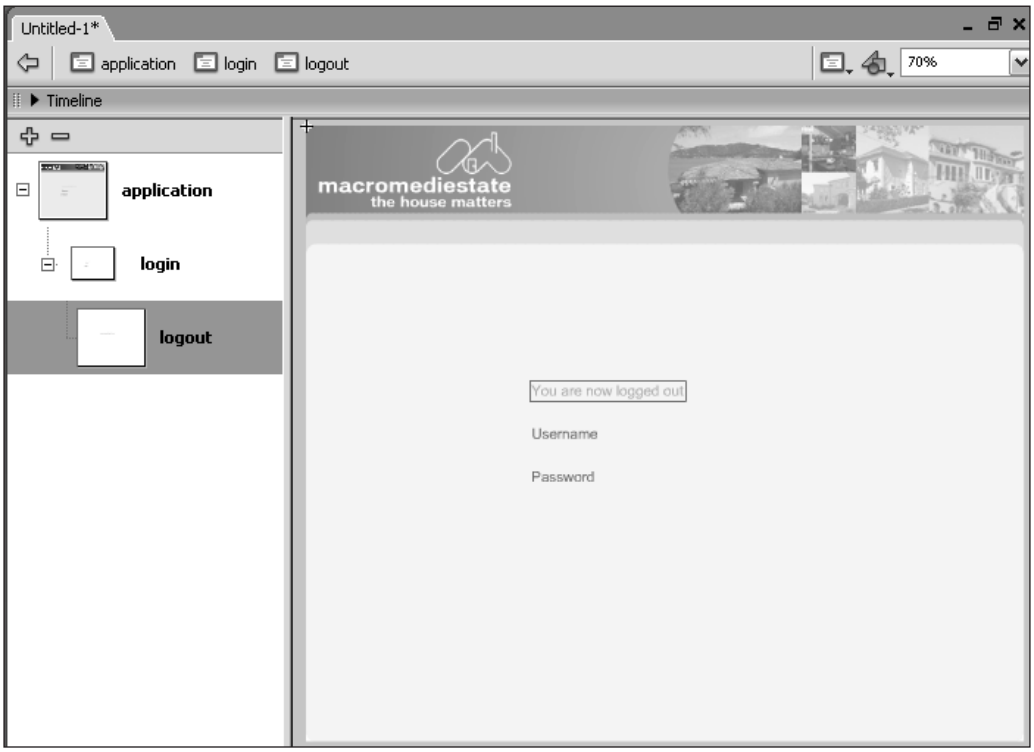
8. Beneath that text field, create a second static text field with the text *Password* using Arial font, black, size 14.

Use the Align panel to align the fields appropriately.

9. Right-click/Control-click the `login` screen in the Screen Outline pane and select Insert Nested Screen.

You will create two nested screens for the `login` screen: a `logout` screen and an `error` screen. Because they are nested screens, the content of both screens will always be displayed in addition to the content in the parent `login` screen and never be displayed independently. The `logout` screen will be displayed over the top of the `login` screen when the user has logged out of the application so the user can log back in if they want. The `error` screen also appears over the top of the `login` screen and will display whenever the user enters an invalid username or password; this functionality will be added in a later lesson.

10. Double-click `form2` in the Screen Outline pane and rename it *logout*. On the Stage for the screen, create a static text field above the existing text fields in the `login` screen with the text *You are now logged out*. Use Arial font, blue (#007ED6), size 14.

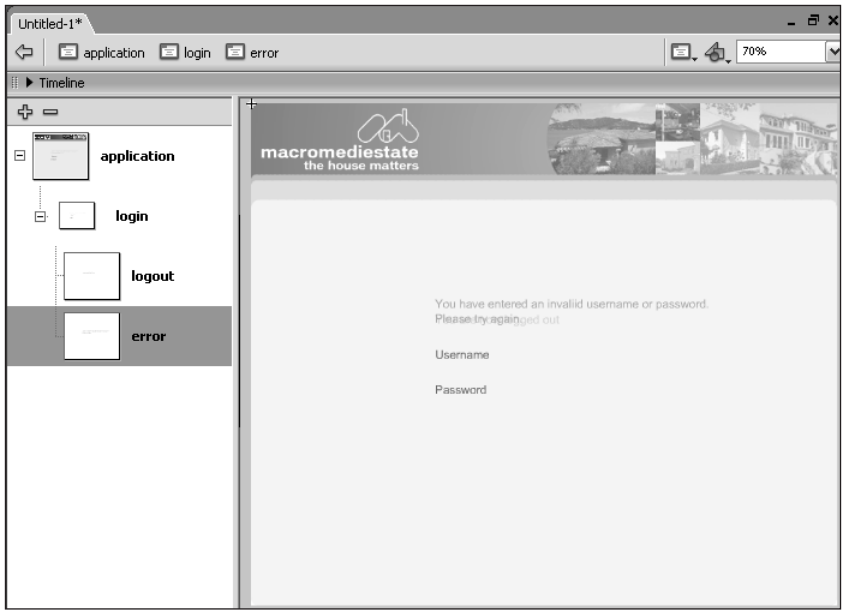


Because these fields are located in different screens, you cannot use the Align panel to align them. Instead, you can align them by eye or you can cut and paste the logout text from the `logout` screen into the `login` screen, align the fields, and then cut the logout text from the `login` screen and use Paste in Place to put it back in the `logout` screen.

11. Right-click/Control-click the `login` screen in the Screen Outline pane and select Insert Nested Screen.

You will now create the nested `error` screen.

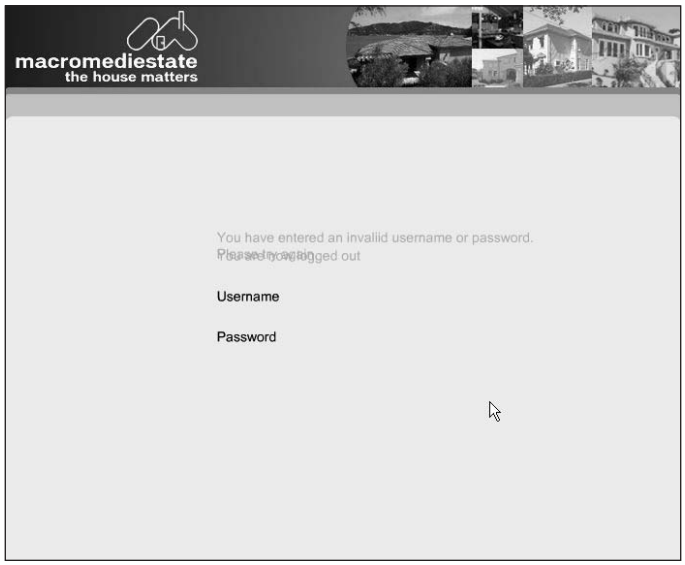
12. Double-click `form3` in the Screen Outline pane and rename it *error*. On the Stage, create a static text field directly on top of the `logout` screen text field with the text *You have entered an invalid username or password. Please try again*. Use Arial font, red (#FF0000), size 14.



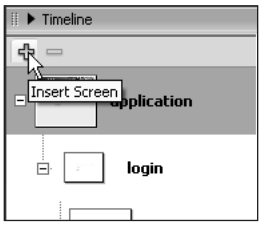
Use the Align panel to align the fields appropriately.

13. Test the application.

You should see all the content in all the screens displayed.



14. Select the `application` screen in the Screen Outline pane and click the Insert Screen button.

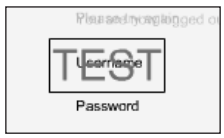


You can also right-click/Control-click a screen and select Insert Screen. When you insert a new screen, the new screen is placed directly below the selected screen and at the same level (it is not nested). The exception is that if you have the root `application` screen selected, a new child screen is created because there can be only one root `application` screen.

15. In the new `form4` screen, use the Text tool to add the text *TEST* with a font size of 42. Place it on top of the username text field.

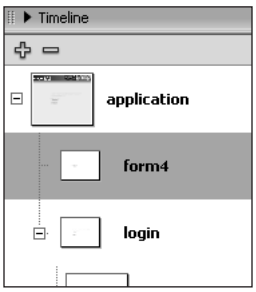
Leave the font color red.

16. Test the application.



You should see this new content displayed *over* the content in the other layers. This is because the stacking order of the screen content is determined by the screen positions in the Screen Outline pane. The content in screens located at the top of the Screen Outline pane is displayed at the bottom; the content in screens located at the bottom of the Screen Outline pane is displayed at the top.

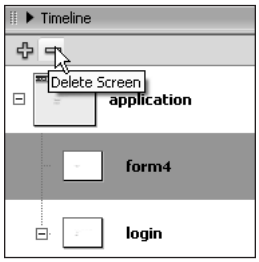
17. Click the `form4` screen in the Screen Outline pane and drag and drop it above the `login` screen.



18. Test the application.

The TEST content should now be displayed *beneath* the content in the other layers.

19. Select the `form4` screen in the Screen Outline pane and click the Delete Screen button.



The `form4` screen is deleted. You can also right-click/Control-click the screen and select Delete Screen.

20. Save the file as *mmestate.fla* in /fpad2004/mmestate/.

You will use this file as the starting file for the next exercise. It should resemble the finished file: /fpad2004/lesson07/intermediate/mmestate_screens.fla.

Setting Screen Visibility in the Authoring Environment

There are two different places that you can and will need to set visibility for a particular screen: in the authoring environment and at runtime. Right now in the authoring environment, all content in every screen is displayed on the Stage at the same time. This is a convenient way to place content that will be displayed at the same time—such as the `login` and the nested `logout` screen—but it is difficult to work with content that will *never* be displayed at the same time—such as the `logout` and `error` screens. You can *hide* screens in the authoring environment, so that their content is not visible unless that screen is selected in the Screen Outline pane.

You hide a screen by right-clicking/Control-clicking the screen in the Screen Outline panel and selecting Hide Screen.

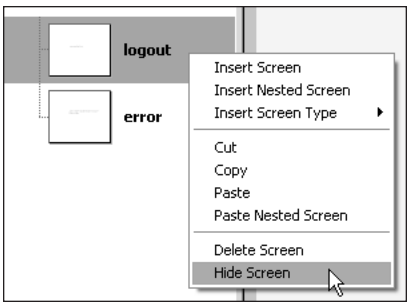
In this exercise, you hide the `login`, `logout`, and `error` screens in the authoring environment so that the contents of each screen are displayed only when that particular screen is selected in the Screen Outline pane.

1. Return to mmestate.fla in /fpad2004/mmestate/.

Return to the file you created in the last exercise or open /fpad2004/lesson07/intermediate/mmestate_screens.fla and save it as *mmestate.fla* in /fpad2004/mmestate/.

Right now, *all* content in *all* screens is displayed on the Stage. This is not desirable because all this content will never be displayed at the same time and it is difficult to design the application visuals.

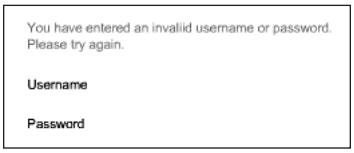
2. Select the `logout` screen in the Screen Outline pane. Right-click/Control-click it and select Hide Screen.



The contents of the `logout` screen are no longer visible unless the `logout` screen is selected.

3. Select the `error` screen in the Screen Outline pane.

The logged-out text should no longer be visible on the Stage.

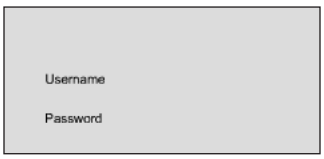


4. Right-click/Control-click the `error` screen and select Hide Screen.

Similarly, you need to hide the `error` screen so that its contents are not displayed when you select the `logout` screen. The content in these two screens is never displayed simultaneously.

5. Select the `application` screen in the Screen Outline pane.

The error text should no longer be visual on the Stage.



6. Right-click/Control-click the `login` screen and select Hide Screen.

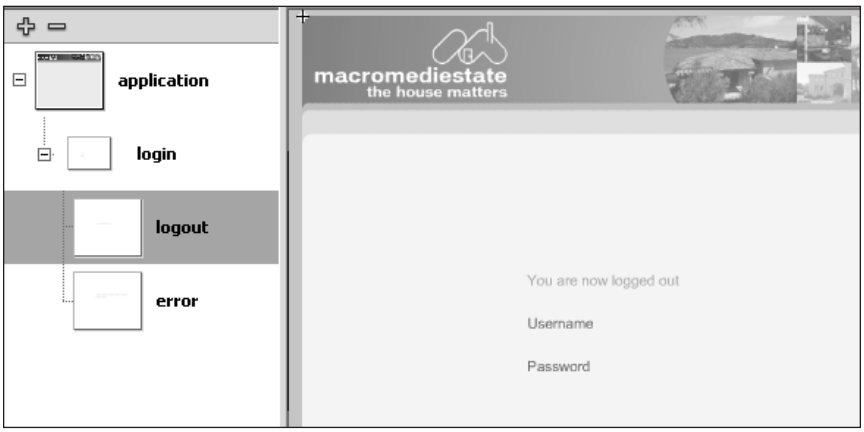
Right now, the `login` screen is the only main page in the application so hiding this screen is not necessary. In a later exercise, though, you will add more screens. Because the `login` screen will not be displayed on every screen in the application, its contents will most likely overlap with other content; thus, you also want to hide it in the authoring environment so its contents are displayed only when it is selected.

7. Select the `application` screen in the Screen Outline pane.

The login page text should no longer be visible on the Stage.

8. Select the `logout` screen in the Screen Outline pane.

The username and password text fields are still visible. Why? Because the content of a parent screen is always displayed when a child screen is selected.



9. Save the file.

It should resemble the finished file: /fpad2004/lesson07/intermediate/mmestate_hide.fla.

10. Test the application.

Even though you have hidden screen content in the authoring environment, the content on every screen is still displayed at runtime. You learn to set the initial runtime visibility of screens in the next exercise.

Setting Initial Runtime Visibility of a Screen

In the last exercise you learned to control the display of the screens in the authoring environment so that the content for all the screens was not displayed at one time. In this exercise, you learn to set the visibility of the screens *at runtime* so that the content for all the screens is not shown at one time in the application. You can set the *initial* visibility of a screen in two ways: in the authoring environment at design time or in code to be executed at runtime.

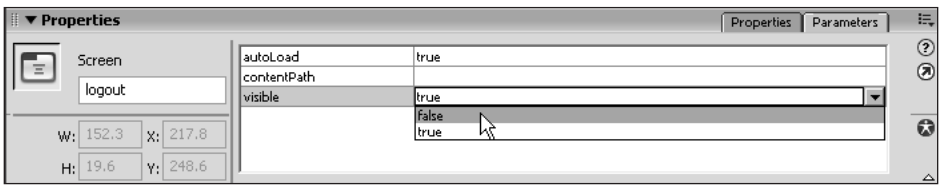
At design time, you can set the initial visibility of a screen by setting the `visible` parameter to true or false in the Parameters tab of the Property inspector. The visibility of a screen can also be set in code; any value set in code that is executed immediately will override any value set in the authoring environment. In later exercises in this lesson, you will learn about the Form screen application programming interface and how to manipulate a screen visibility and much more by using code.

In this exercise, you make the `login` and `application` screens the only screens initially visible; the `logout` and `error` screens will not be initially visible.

1. Return to mmestate.fla in /fpad2004/mmestate/.

Return to the file you created in the last exercise or open /fpad2004/lesson07/intermediate/mmestate_hide.fla and save it as *mmestate.fla* in /fpad2004/mmestate/.

2. Select the `logout` screen in the Screen Outline pane. Click the Parameters tab in the Property inspector and set the screen's `visible` parameter to *false*.



The logout message will no longer be initially visible in the application when you test it. You make the `logout` screen visible again in a later exercise when you add navigation buttons to the application.

3. Select the `error` screen in the Screen Outline pane. In the Parameters tab of the Property inspector, set its `visible` parameter to *false*.

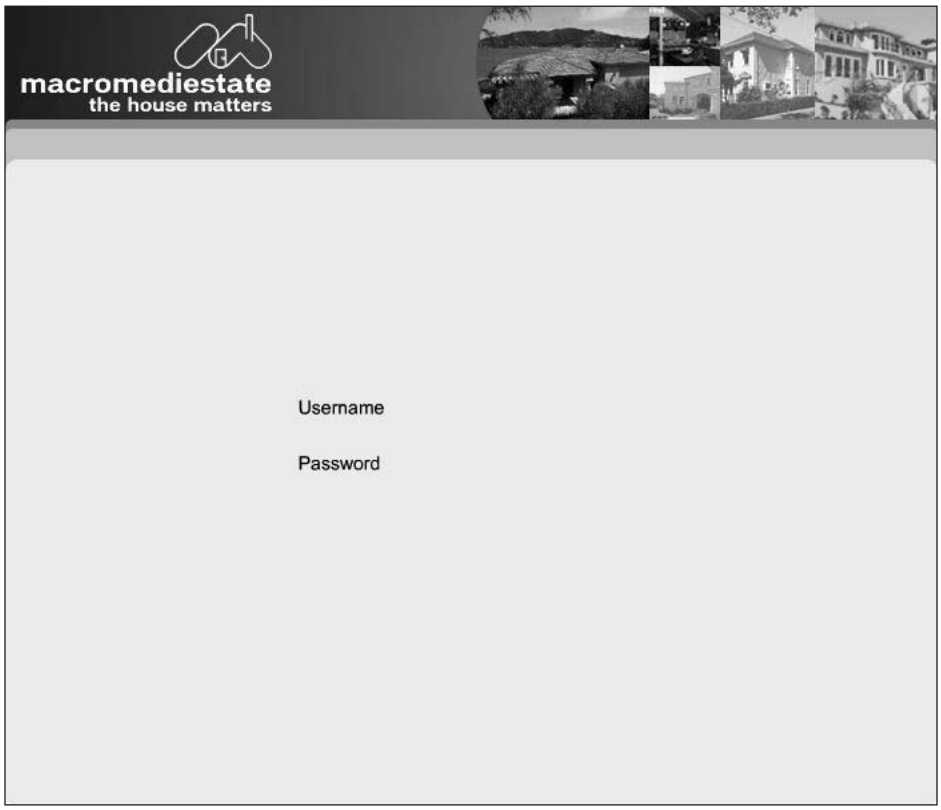
The error message will no longer be initially visible in the application. You make the `error` screen visible again in Lesson 13, "Consuming Web Services," when you add application logic to check login values against a database.

4. Save the file.

It should resemble the finished file: /fpad2004/lesson07/intermediate/mmestate_visible.fla.

5. Test the application.

Now only the content in the `application` and `login` screens is initially displayed.



Architecting an Application with Screens

Now that you have a little practice creating and manipulating screens, you are ready to lay out the structure for the rest of the application. Right now, the Macromediestate application consists only of the `login` screen; you need to build the rest.

Here are some considerations to keep in mind when you are architecting an application with screens:

Content to display on every screen. Place the content you want displayed on every page in the application in the `application` screen. This content usually includes any header text, logos, and possibly a footer. It might also contain navigation if your application does not require a login. If

your application requires a login, you will not want the navigation on the `application` screen because you do not want it visible and functional until after the user has successfully logged in.

Content to be displayed on multiple screens, but not all screens. If you have content to be displayed on more than one screen, place this content on a screen and then make all screens that should display this content (along with their own content) nested screens.

In this exercise, you look at the finished Macromediestate Flash application (the SWF not the FLA!) and then finish creating and laying out the necessary screens in your application to achieve comparable functionality. You will add navigation to traverse these pages in a later exercise in this lesson; you add additional content to these screens throughout the rest of the book.

1. Open mmestate.swf in /fpad2004/lesson16/complete/.

Open the SWF. You are opening the finished Macromediestate application that you will build throughout the rest of this book. Examine the finished application to figure out how you should originally architect it using screens. Think about what content should be placed on different screens, what screens should be sibling screens, and what ones should be nested screens.

Try logging into the application using a random username and password and see what happens. Log in successfully to the application using *fpad* for both the username and password. Be sure to navigate to all the different parts of the application.

Note *If you did not install the application server files and the database, some parts of the application will not be functional.*

2. Return to mmestate.fla in /fpad2004/mmestate/.

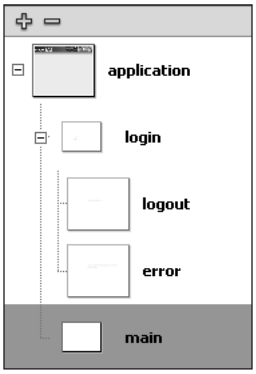
Return to the file you created in the last exercise or open /fpad2004/lesson07/intermediate/mmestate_visible.fla and save it as *mmestate.fla* in /fpad2004/mmestate/.

3. Select the `application` (or `login`) screen in the Screen Outline pane and click the Insert Screen button.

A new screen is created at the same level as the `login` screen.

4. Double-click the new form in the Screen Outline pane and rename it *main*.

This screen will contain the content to be displayed on all the pages in the application after a user successfully logs in. In this application, this content includes all the navigation buttons.



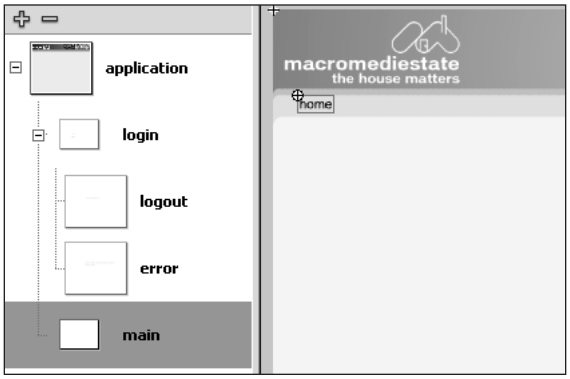
5. Select File › Import › Open External Library.

You will import prebuilt navigation buttons from the library of another FLA document.

6. In the Open as Library dialog box, browse to navbuttons.fla in /fpad2004/mmestate/assets/ and click Open.

You should now see the Library panel for navbuttons.fla. The document itself has not been opened, just its library, so that you can drag out items from its library to use in other documents. Notice that the background color of the Library panel is gray, which is a visual cue that indicates that it is not the library of the currently selected document.

7. Drag out an instance of the `btnHome` button and place it in the gray bar under the Macromediestate logo. Name the instance *home_btn* in the Property inspector.



8. Similarly, drag out instances of the `btnSearch`, `btnSaved`, `btnCalc`, and `btnLogout` buttons and place them in the gray bar under the logo. Give them the instance names *search_btn*, *saved_btn*, *calc_btn*, and *logout_btn* in the Property inspector.

Use the Align panel to align the fields appropriately.

9. In the Property inspector, set the `visible` parameter of the `main` screen to *false*.

You do not want the `main` screen to be initially visible. You do not want it to be visible until after the user has successfully logged in to the application.

10. Right-click/Control-click the `main` screen in the Screen Outline pane and select Insert Nested Screen.

A new nested screen is created under the `main` screen. You will create a nested screen for each of the screens in the application that are displayed after a user successfully logs in.

11. Double-click the new form in the Screen Outline pane and rename it *home*. Right-click/Control-click the screen and select Hide Screen.

You do not want this content to always be displayed in the authoring environment.

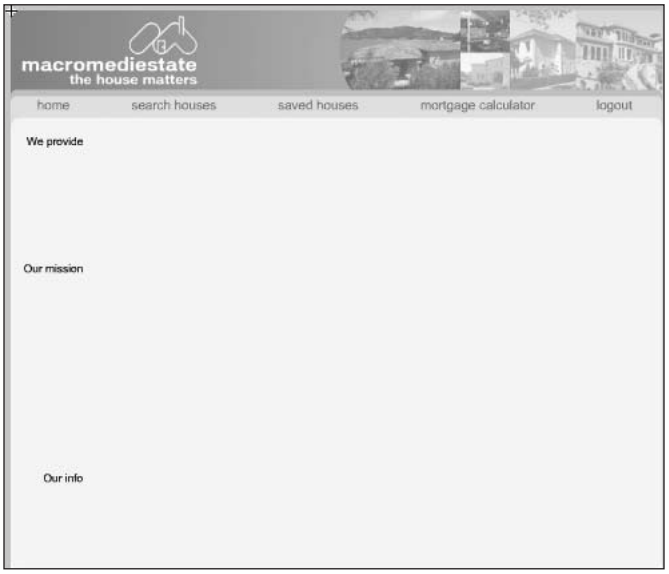
12. On the Stage for the `home` screen, create a static text field in the upper left corner with the text *We provide*. Use Arial font, black, size 12.

In Lesson 8, you will load the content that will go next to this field dynamically from an XML file.

13. Create a second static text field in the middle left part of the Stage and type in the text *Our mission*. Use Arial font, black, size 12.

You will also load the content to go next to this field dynamically from an XML file.

14. Create a third static text field in the lower left corner with the text *Our info*. Use Arial font, black, size 12.



This field is for contact information. You could now add all the contact information (which does not change) right next to this text field. Instead of typing it all in, though, you will load it from the XML file as well. Use the Align panel to right-align the three fields.

15. Select the `main` screen in the Screen Outline pane. Right-click/Control-click and select Insert Nested Screen.

Next you will create a nested screen for the search page.

16. Double-click the new form in the Screen Outline pane and rename it *search*. Right-click/Control-click the screen and select Hide Screen. In the Property inspector, set the `visible` parameter to *false*.

This step sets up the screen for the search page. You will add content to this screen in a later lesson.

17. Right-click/Control-click the `search` screen in the Screen Outline pane and select Insert Screen.

Next you will create a nested screen for the saved page.

18. Double-click the new form in the Screen Outline pane and rename it *saved*. Right-click/Control-click the new screen and select Hide Screen. In the Property inspector, set the `visible` parameter to *false*.

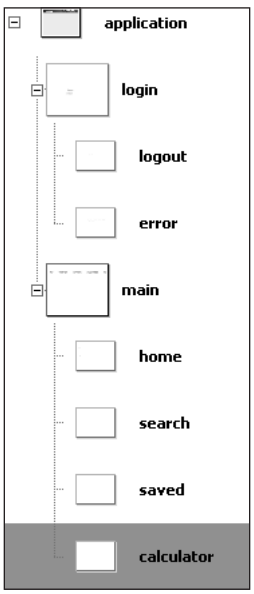
This step sets up the screen for the saved search page. You will add content to this screen in later lessons.

19. Right-click/Control-click the `saved` screen in the Screen Outline pane and select Insert Screen.

Next you will create a nested screen for the mortgage calculator screen.

20. Double-click the new form in the Screen Outline pane and rename it *calculator*. Right-click/Control-click the new screen and select Hide Screen. In the Property inspector, set the `visible` parameter to *false*.

This step sets up the screen to display the mortgage calculator.



21. From the Components panel, drag out an instance of the MortgageCalculator component and place it on the Stage.

It does not matter exactly where you place the calculator; it is draggable and the user can drag it where they like. This screen will not close automatically when the user navigates to another page. It will stay visible until the user closes it.

Note *The MortgageCalculator component will exist in the Components panel only if you did the last exercise of Lesson 6, "Creating Components." To install the component, follow the installation steps in the last exercise of Lesson 6.*

22. Save the file and test the application.

It should resemble the finished file: /fpad2004/lesson07/intermediate/mmestate_layout.fla.

Make sure you only see the `login` screen. None of the new content including the navigation buttons should be visible. You hook up the buttons and make them functional in a later exercise.

Using the Form Screen API

To manipulate a screen object using code, you need to become familiar with the application programming interface for the Form screen class (mx.screens.Form), consisting of all the properties and methods you can use to manipulate a Form screen. The inheritance for this class is as follows:

MovieClip > UIObject > UIComponent > View > Loader > Screens > Form

You can find the source code for all these classes in the Classes folder in the First Run and Local Settings folders. For Windows, these two directories are <boot drive>:\Program Files\Macromedia\Flash MX 2004\<language>\First Run\Classes\ and <boot drive>:\Documents and Settings\<user>\Local Settings\Application Data\Macromedia\Flash MX 2004\<language>\Configuration\Classes\. For Macintosh, these directories are <Macintosh HD>/Applications/Macromedia Flash MX 2004/<language>/First Run/Classes/ and <Macintosh HD>/Users <username>/Library/Application Support/Macromedia/Flash MX 2004/<language>/Configuration/Classes/.

Here is a general sketch of the functionality provided by each of these classes.

UIObject. Full class name is mx.core.UIObject, and it inherits from the MovieClip class. It is the base class for all the prebuilt components and has no visual component. It provides additional functionality over the MovieClip class for implementing styles, broadcasting events, and resizing by scaling.

UIComponent. Full class name is mx.core.UIComponent, and it inherits from the UIObject class. It is also a base class for all the prebuilt components and has no visual component. It provides additional functionality for receiving focus and keyboard input, enabling and disabling components, and resizing by layout.

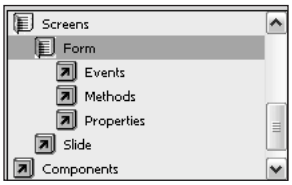
View. Full class name is mx.core.View, and it inherits from the UIComponent class. It is the base class for all view and container components and provides the functionality for managing loaded content.

Loader. Full class name is mx.controls.Loader, and it inherits from the View class. It is a container component into which you can dynamically load content (JPGs or SWFs), monitor the loading progress, and rescale the content.

Screens. Full class name is mx.screens.Screen, and it inherits from the Loader class. It has a user-friendly API for keeping track of and managing its children screens.

Form. Full class name is mx.screens.Form, and it inherits from the Screens class. It has properties and methods with the word *form* in them for keeping track of and managing its children screens.

The properties, methods, and events for the Form class are listed in the Actions toolbox under Screens > Form. The members listed in the toolbox are those specifically for the Form class as well as many for its parent classes.



Descriptions for a few common properties, methods, and events are included here:

`Visible`. A property (from the UIObject class) that sets whether a Form screen is visible at runtime when its parent screen is visible. This is the parameter you previously set in the Property inspector.

`contentPath`. A property (from the Loader class) that contains a string path to a JPG or SWF to be loaded into the screen.

`numChildForms`. A property (from the Form class) that returns the number of child forms that the Form screen contains.

`Load()`. A method (from the Loader class) that loads the content specified in the `contentPath` property into the screen.

`addEventListener()`. A method (from the UIEventDispatcher class) that registers a listener object for a particular screen event.

`getChildForm(index)`. A method (from the Form class) that returns a child Form screen at the given index.

`Reveal`. An event (from the UIObject class) broadcast when a screen is made visible.

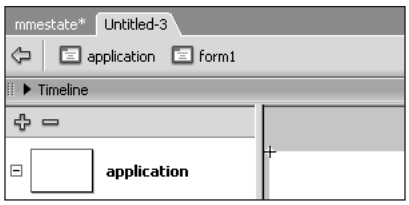
`Load`. An event (from the UIObject class) broadcast when a screen is finished loading into the Flash Player.

To become familiar with the Forms API, open the /fpad2004/lesson07/intermediate/mmestate_layout.fla and browse the form screens branch of the actions toolbox in the Actions panel.

Placing Code in the Timeline of a Screen

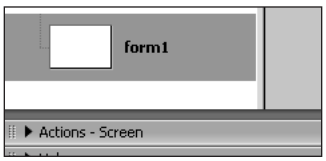
Just as with a normal MovieClip, there are three places to add code associated with a screen: in the Timeline of a screen, in event handlers on a screen object, or in a separate class file associated with a screen. You will add your code in each of these locations in the next three exercises.

By default, the Timeline panel is not open when creating Form-based applications. The Timeline is initially hidden to simplify the interface and not confuse more traditional, non-Flash programmers.



When you open the Actions panel for a particular screen, you are placed *on* the object—that place you learned about in Lesson 4, "Creating Button and MovieClip Objects," for MovieClips in which you have to place all your code inside an event handler for that class instance. You can tell because the title of the Actions panel is Actions – Screen instead of Actions – Frame.

In Lesson 4, you learned that when you open the actions panel for a particular screen, you are placed *on* the object. You have to add all your code for MovieClips inside an event handler for that class instance.



To add code to the Timeline instead of on the object, you need to select a layer or the first frame in a layer (which is usually called actions) and then add code to the Actions panel. Any code added to the Timeline of a screen is executed when that screen is loaded, which is when the application is loaded into the Flash Player.

In this exercise, you create a script to make the navigation buttons functional, enabling navigation between the different screens in the applications. You place your script in the Timeline. You will move the code to the screen object and then to an external class file in the next two exercises.

1. Return to mmestate.fla in /fpad2004/mmestate/.

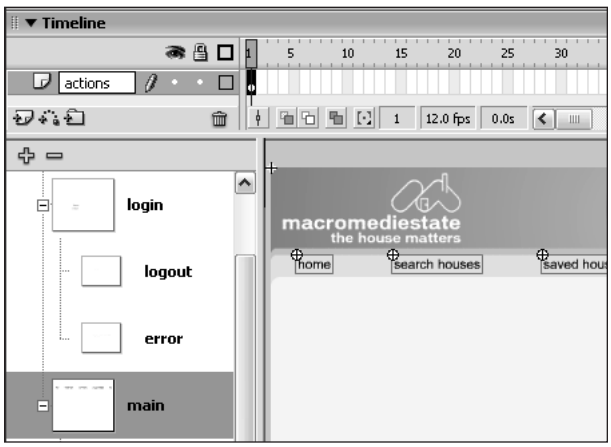
Return to the file you used in the last exercise or open /fpad2004/lesson07/intermediate/mmestate_layout.fla and save it as *mmestate.fla* in /fpad2004/mmestate/.

2. Select the `main` screen in the Screen Outline pane.

You will add the navigation code to the Timeline of the `main` screen. The advantage of putting code in the Timeline of the `main` screen is that all the code and the visual elements for one part of the application are all contained and associated with that screen. This structure works well if your code does not interact with any of the other screens in the application. Unfortunately, the Actions panel for a Form-based application does not show all the places in the application that have code attached to them as it did in a normal Flash document so you have to open the Actions panel and then click each screen in the Screen Outline pane to see if it has any associated code.

You can also add this code to the Timeline of the `application` screen. The advantage of placing the code in the Timeline of the `application` screen is that if you have code for multiple screens, all the code is in one place.

3. Click the title bar of the Timeline panel to open it. Rename Layer 1 as *actions*.



Note that you can still organize content within a screen in different layers, just as you did in a normal document.

4. Open the Actions panel. Look at its title bar.

Because you have a layer in the Timeline selected, the focus of the Actions panel has shifted to the first frame in the Timeline; the title of the Actions panel should be Actions – Frame.

5. Create an `onPress` event handler for `logout_btn`.

```
logout_btn.onPress=function():Void{};
```

6. Inside the event handler definition, set the `visible` property of the `logout` screen to *true*. Use an absolute path to the `logout` screen.

```
_root.application.login.logout.visible=true;
```

7. Still inside the event handler definition, set the `visible` property of the `login` screen to *true* and the `visible` property of the `main` screen to *false*. Use absolute paths.

```
logout_btn.onPress=function():Void
{
      root.application.login.logout.visible=true;
      root.application.login.visible=true;
      _root.application.main.visible=false;
};
```

To test this code you need to temporarily change the initial visibilities of the `login` and `main` screens to false and true, respectively, because right now you have no way to get past the `login` screen in the application. You change these values in the next step.

8. Use the Property inspector to change the `visible` parameter of the `login` screen to *false* and the `visible` parameter of the `main` screen to *true*. Test the application.

You should see the `main` screen instead of the `login` screen.

9. Click the logout button.

The navigation buttons should disappear, and you should see the `login` and `logout` screens.

10. Return to the code in the Timeline of the `main` screen and create an `onPress` event handler for `home_btn`.

```
home_btn.onPress=function():Void {};
```

11. Inside the event handler definition, set the `visible` property of the `search` and `saved` screens to *false* and the `home` screen to *true*. Use relative paths.

Keep in mind that inside the event handler, the scope is still the `main` screen.

```
home_btn.onPress=function():Void
{
      search.visible=false;
      saved.visible=false;
      home.visible=true;
};
```

12. After this event handler, create an `onPress` event handler for `search_btn`.

```
search_btn.onPress=function():Void {};
```

13. Inside the event handler definition, set the `visible` properties of the `home` and `saved` screens to *false* and the `search` screen to *true*. Use relative paths.

```
search_btn.onPress=function():Void
{
      search.visible=true;
      saved.visible=false;
      home.visible=false;
};
```

14. After this event handler, create an `onPress` event handler for `saved_btn`.

```
saved_btn.onPress=function():Void {};
```

15. Inside the event handler definition, set the `visible` properties of the `home` and `search` screens to *false* and the `saved` screen to *true*. Use relative paths.

```
saved_btn.onPress=function():Void
{
      search.visible=false;
      saved.visible=true;
      home.visible=false;
};
```

16. After this event handler, create an `onPress` event handler for `calc_btn`.

```
calc_btn.onPress=function():Void {};
```

17. Inside the event handler definition, set the `visible` property of the `calculator` screen to *true*. Use a relative path.

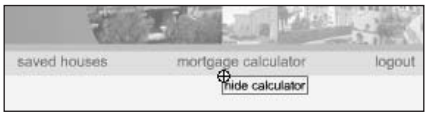
```
calc_btn.onPress=function():Void
{
      calculator.visible=true;
};
```

18. Test the application. Try out all the buttons, except the logout button, and make sure they work.

Don't click the logout button or you will not be able to get back to the `main` screen. Notice that when you click the calculator button and then click another navigation button, the calculator does not disappear. You want the user to be able to use the calculator when they look at other

application pages that include house prices. You do need some way, though, to close the calculator, which is a functionality that is not built into the component. You need to add a button with this functionality to the `calculator` screen.

19. Select the `calculator` screen in the Screen Outline pane. From the navbuttons Library panel, drag out an instance of the `btnHideCalc` button and drop it below the mortgage calculator button in the navigation bar. Give it an instance name of `hide_btn`.



You should have the navbuttons library open from a previous exercise. If you closed it, you can reopen it by selecting File > Import > Open External Library and opening navbuttons.fla located in /fpad2004/mmestate/assets/.

20. Return to the code in the Timeline of the `main` screen and add an `onPress` event handler for `hide_btn`.

Be sure to scope `hide_btn` appropriately. It is located in the `calculator` screen, not in the `main` screen, where the code is located.

```
calculator.hide_btn.onPress=function():Void {};
```

21. Inside the event handler definition, set the `visible` property of the `calculator` screen to *false*. Use a relative path.

```
calculator.hide_btn.onPress=function():Void
{
      calculator.visible=false;
};
```

22. Save the file and test the application. Try out your new hide calculator button and make sure it works.

Click the mortgage calculator button; the calculator should appear. Click the home button; the calculator should still be visible. Click the hide calculator button; the calculator should disappear.

The finished file should resemble: /fpad2004/lesson07/intermediate/mmestate_timelinecode.fla.

Placing Code on a Screen Object

In the last exercise, you added code to the Timeline of a screen. The whole Forms-based application development environment, though, tries to steer you away from this workflow. When you select a screen and then open the Actions panel, any code you enter is added to the screen object and must reside inside an event handler for that screen. This is analogous to how you added code *on* instances of MovieClips in Lesson 4.

To add code to a screen instead of the Timeline, you simply select the screen in the Screen Outline pane and then open the Actions panel. Remember, though, that any code placed here must be inside a screen event handler. The code inside an event handler will be executed whenever the event fires; this can be when the screen loads, is made visible, is clicked, and so on.

In this exercise, you move the script controlling the navigation buttons you created in the Timeline to an event handler *on* the `main` screen instance. In the next exercise, you will move the code to an external class file.

1. Return to mmestate.fla in /fpad2004/mmestate/.

Return to the file you created in the last exercise or open /fpad2004/lesson07/intermediate/mmestate_timelinecode.fla and save it as *mmestate.fla* in /fpad2004/mmestate/.

2. Return to the code in the Actions panel for the first frame of the `main` screen. Select all the code and cut it.

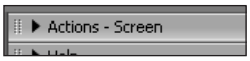
You will move this code to the inside of an event handler *on* the `main` screen instance.

3. Collapse the Timeline panel.

You will no longer use the Timeline to add code to the application.

4. Select the `main` screen in the Screen Outline pane.

The title bar of the Actions panel should change from Actions – Frame to Actions – Screen.



5. Open the Actions panel (if it is collapsed) and create a `reveal` event handler.

```
on(reveal)
{
}
```

6. Paste the code you copied inside the `reveal` event handler.

```
on(reveal)
{
      [all existing code]
}
```

Remember that when placing code directly on an object and not in the Timeline, all code must be inside an event handler, and you must use the `on(event){}` syntax.

7. Save the file and test the application.

Your navigation buttons should work as they did before.

Right now, the code defining the event handlers is executed when the screen is made visible for the first time (which in this case happens to be when the screen is loaded into the Flash Player). The code can also be executed multiple times if the screen's `visible` property is changed with code; the code will be executed anytime the screen's `visible` property is changed from false to true.

8. Return to the code on the `main` screen object and change the `reveal` event to a `load` event.

```
1 on (load)
2 {
3     logout_btn.onPress=function():Void
4     {
```

The code defining the event handlers does not need to be executed every time the screen is made visible; it only needs to be executed once.

9. Save the file and test the application.

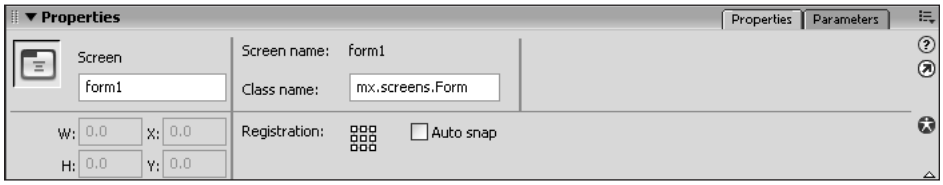
Your navigation buttons should work the same as before.

The finished file should resemble: /fpad2004/lesson07/intermediate/mmestate_objectcode.fla.

Placing Code in a Form Screen Subclass

In the last two exercises you added code directly to the FLA. You might want to separate your code from the FLA, enabling multiple developers to work on the application and also allowing your source code to be placed in a source control system. You can separate the code from the FLA in two ways, both of which you have already learned about. One method is to place the script in an external file and then include that file in the appropriate place in the FLA using `#include`. This method enforces the separation, but it does not improve the architecture, readability, and maintainability of the application. These factors can all be improved by implementing a more object-oriented technique of separating the code from the FLA by placing it in a class file.

In Lesson 6 you leaned how to associate a class file with a MovieClip symbol in the library. The linked class file is automatically associated with the symbol and instantiated whenever an instance of the MovieClip symbol is loaded into the Flash Player. You can make the same type of association between a screen object and a class file. Instead of specifying a linkage to the class file in the library, though (because screens do not appear in the library), you assign the class name in the Property inspector.



By default, a screen is linked to the mx.screens.Form class, which gives it its base functionality as a Form screen. If you specify a new class to associate with a screen, you need to make sure that the class extends the mx.screens.Form class to maintain the functionality of the screen as a Form screen object. In addition to creating a constructor for your class, you can also create an `init()` method, which is automatically invoked by the parent UIObject constructor when the class is instantiated; you do not have to explicitly call the `init()` method. UIObject's constructor calls the `init()` method defined at the lowest subclass (in this case your mx.screens.Form subclass). If you create an `init()` method, you should call the parent class's `init()` method using `super.init()` from the form screen subclass's `init()` method to ensure that all the form screen's base classes finish initializing (and are in usable states).

In this exercise, you move the navigation code to an external class file that subclasses the mx.screens.Form class and then associate that class with the `main` screen.

1. Create a new ActionScript file and save it as *MmestateMain.as* in /fpad2004/mmestate/classes/.

Make sure you use the appropriate capitalization.

2. Inside the file, define a class called *MmestateMain*.

```
class MmestateMain
{
}
```

You saved the class in the classes folder, which is a classpath so you do not need to specify a package for the class.

3. Make the class extend the mx.screens.Form class.

```
class MmestateMain extends mx.screens.Form
```

The class must extend the mx.screens.Form class to maintain the functionality of the screen as a Form screen object.

4. Inside the class definition, create the skeleton code for a class constructor with no parameters. Save the file.

```
function MmestateMain()
{
}
```

5. After the constructor, define a *public* method called `init` with no parameters and no return value. Save the file.

If an `init()` method exists for a screen subclass, the method is automatically invoked; you do not have to explicitly call it. You will place all the code that should initially be executed inside the `init()` method instead of directly inside the constructor. It is a best practice to keep the amount of code inside the constructor to a minimum.

```
public function init():Void
{
}
```

6. Inside the `init()` method, call the parent class's `init()` method.

```
super.init();
```

7. Click the Check Syntax button. Save the file.

You should not get any errors.

It is a good idea to always check a class file for errors before you try and link it to a screen. If you attempt to link a class that has errors, the linkage to the screen will fail with a message stating that no such class exists.

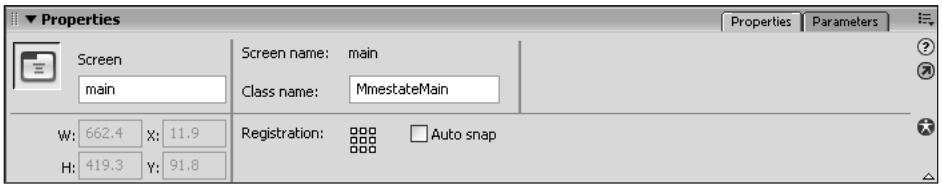
8. Return to mmestate.fla in /fpad2004/mmestate/.

Return to the file you created in the last exercise or open /fpad2004/lesson07/intermediate/mmestate_objectcode.fla and save it as *mmestate.fla* in /fpad2004/mmestate/.

9. Select the `main` screen and then open the Property inspector. Click the Properties tab and look at the name of the class associated with the screen.

You should see mx.screens.Form.

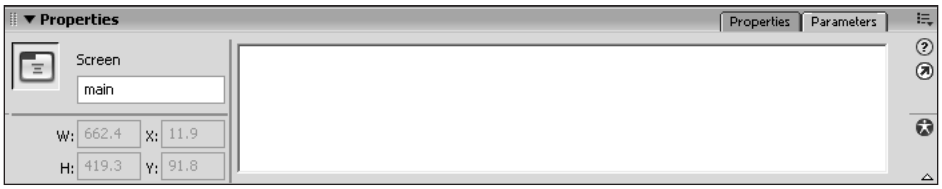
10. Change the name of the class linked to the `main` screen to *MmestateMain* and press enter.



Note *If you did not add the ./classes classpath in Lesson 5, "Creating Classes," this step will fail and you will see a message stating that this class does not exist. Instructions for setting the classpath can be found in the "Setting Classpaths" exercise in Lesson 5.*

11. Click the Parameters tab of the Property inspector.

Unfortunately, when you assign a new class to a screen, the screen parameters no longer appear in the Property inspector. If you want to assign values, you now must do it via code.



12. Open the Actions panel for the `main` screen and select all the code inside the `load` event handler; cut the code to the Clipboard.

You will place this code in the external class file.

13. Delete the remaining lines of code and save the file.

Delete the lines of code defining the load event handler.

14. Return to the MmestateMain class definition and paste the code inside the `init()` method.

```
 9      public function init():Void
10      {
11          super.init();
12          logout_btn.onPress=function():Void
13          {
14              _root.application.login.logout.visible=true;
15              _root.application.login.visible=true;
```

You can highlight the code and press the Tab key to indent it appropriately.

15. Above the constructor, define each of the `main` screen Button instances as *private* properties of type *Button* for the class.

```
private var logout_btn:Button, home_btn:Button, search_btn:Button,
saved_btn:Button, calc_btn:Button;
```

You can make these separate variable declarations or combine them into one, as shown here. You are not defining the `hide_btn` Button because it is not a property of the main screen; it is a property of the calculator screen.

Note *To manipulate these Buttons in other class files, you would make them public.*

16. Define the `calculator` screen as a *private* property of type *mx.screens.Form*.

```
private var calculator:mx.screens.Form;
```

17. Inside each of the event handlers in the `init()` method (except for `calculator.hide_btn`), prefix each of the relative screen references with `_parent`.

There should be ten references.

```
 9     public function init():Void
10     {
11         super.init();
12         logout_btn.onPress=function():Void
13         {
14             _root.application.login.logout.visible=true;
15             _root.application.login.visible=true;
16             _root.application.main.visible=false;
17         };
18         home_btn.onPress=function():Void
19         {
20             _parent.search.visible=false;
21             _parent.saved.visible=false;
22             _parent.home.visible=true;
```

Inside the constructor you are assigning a function literal to be the value of a class property. The code inside the function literal (inside the event handler definition) is scoped to the property itself (for example, `logout_btn`), not to the class instance (the `main` screen in this case). You can change the references to absolute references, or you can give them the proper scope by prefixing each of them with `_parent`, which refers to the class instance they reside in—in this case, the `main` screen.

18. Inside the `calculator.hide_btn` event handler, change `calculator` to `_parent`. Save the file.

```
calculator.hide_btn.onPress=function():Void
{
      _parent.visible=false;
};
```

19. Click the Check Syntax button.

You should get only one error: There is no property with the name 'hide_btn'. This is because you defined calculator as type mx.screens.Form, and this class definition does not have a property called `hide_btn`. The rigorous way to fix this error is to create a new class file, MmestateCalc, which defines a public property called `hide_btn`, to link this class to the `calculator` screen, and then define `calculator` as type MmestateCalc instead of mx.screens.Form. A quicker, less rigorous solution is to change the type of the `calculator` screen from mx.screens.Form to a general Object. Because the latter solution is faster and because you do not need to place any other code in a class for the `calculator` screen, you will use this solution in the next step. Feel free, though, to define and link a class instead.

20. Change the type of the `calculator` property from mx.screens.Form to *Object*. Click the Check Syntax button. Save the file.

```
private var calculator:Object;
```

You should no longer get any errors.

21. Return to mmestate.fla and test the application. Click the various buttons in the navigation bar.

Your navigation should work exactly as it did before. The only difference in your application is that now your code is contained in an external class file associated with a particular screen.

22. Return to mmestate.fla. In the Property inspector, change the `visible` parameter for the `login` screen to *true*. Save the file.

You need to change back the initial screen visibilities so that the `login` screen is initially visible and the `main` screen is not. You build the button functionality to advance from the `login` screen to the `main` screen in the next lesson.

The finished file should resemble: /fpad2004/lesson07/intermediate/mmestate_classcode.fla.

23. Return to MmestateMain.as. Inside the `init()` method, set the `visible` property of the screen instance to *true*. Save the file.

```
this.visible=false;
```

Because you have linked the `main` screen to a class, you must set its visibility with code. If you test the application now, you should only see the login screen.

The finished file should resemble: /fpad2004/lesson07/intermediate/classes/MmestateMain_classcode.as.

What You Have Learned

In this lesson, you have:

- Created a Form-based application using screens (pages 305–312)
- Set the visibility of a screen in the authoring environment (pages 312–314)
- Set the initial runtime visibility of a screen (pages 315–316)
- Architected an application with screens (pages 316–322)
- Became familiar with the Form screen API (pages 322–323)
- Added code in the Timeline of a screen (pages 324–328)
- Added code on a screen object (pages 329–330)
- Added code in a class associated with a screen (pages 330–335)