



Components and reusable code are important aspects to developing applications for any platform, including Flash Lite 1.1. Components speed up development and help make coding tasks easier. Even though Flash Lite 1.1 is not as robust for application development compared to higher versions, it is still useful to have Flash Lite 1.1 components and is important to consider techniques for developing them.

Contents

- [1 Download files](#)
- [2 Understanding Flash Lite 1.1 procedures](#)
- [3 Emulating function behaviors](#)
- [4 Setting up a common library](#)
- [5 Configure the slideshowlite component](#)
- [6 Setting up custom procedures](#)
- [7 Guidelines for organizing a Flash Lite 1.1 component](#)

Download files

Download the file `Flash_Lite_1.1_components.zip` [1] containing the `Flash Lite 1.1 components.fla` and `basicsslideshow.fla`. The `Flash Lite 1.1 components.fla` contains the reusable `slideshowlite` component movie clip and the `basicsslideshow.fla` is an example of how to use it to create a slide show.

Understanding Flash Lite 1.1 procedures

Components need an abstraction layer so that you or a user can build code around the component without the need to add or alter code within the component. In order to have an abstraction layer you need some way to create custom functions or procedures for your Flash Lite 1.1 component.

While Flash Lite 1.1 ActionScript does not support functions in the conventional sense, it does support a form of procedure. A Flash Lite procedure is code stored in a frame with a label representing the name of the procedure. Flash Lite can execute this code without changing the position of the playhead.

To execute a procedure you use the `call()` ActionScript command. The argument to the `call()` command is a label of the frame containing the procedure or a target path to a different movieclip with the label of the procedure. Use a `?:?` character to append the label on the end of the target path.

```
/*
call a procedure in the current timeline.
?procedurename? is the label of the frame in the current timeline containing the procedure code
*/
call(?procedurename?); // execute code in the labeled frame without moving the playhead

/*
call a procedure stored within a frame of movie clip in another timeline, note the use of the : t
*/
call(?/component:procedurename?); // target path to a label in a different timeline
```

Build_a_reusable_Flash_Lite_1.1_component

Another way to execute a procedure is with the `tellTarget()` action. The `tellTarget()` command is similar to a `with()` statement in that it scopes `call()` commands, navigation commands and variable declarations to the movie clip specified in the target path. The `tellTarget()` command is a convenient way to set variables for and get variables from the procedure call without managing cumbersome slash syntax target paths.

```
/*
tellTarget() scopes the call() command to the component movie clip and the call() command executes
*/
tellTarget(?/component?){
    variable = ?value?; // variable in component movieclip
    call(?procedurename?); // execute a procedure in the component movieclip
    gotoAndPlay(?start?); // play a frame labeled start in the component movieclip
}
```

Emulating function behaviors

You can use naming conventions to emulate typical function features like passing and returning values from a procedure. For example to set or get values from a procedure you might consider using the procedure name as the input or return variable.

```
// passing a value to procedure
hideSlide = ?/slide1?; // target path of movie clip slide to hide
call(?hideSlide?); // the hideSlide procedure expects a value for the variable hideSlide

// procedure code, expects the variable ?hideSlide?
setProperty(hideSlide,_visible,0);
setProperty(hideSlide,_alpha,0);

// returning value from procedure
call(?getCurrentSlideNum?);
currentSlideNum = getCurrentSlideNum; // assign return value to custom variable

// procedure code, assigns value to variable with same name as procedure
getCurrentSlideNum = (curSlide > numSlides) ? numSlides : curSlide;
```

Another useful feature that you can emulate in Flash Lite 1.1 is a callback. A callback event executes when the component's state changes. In the example `slideshowlite` component, there is a callback event that executes when the slide show is complete, called `?onSlideShowComplete?`. To set up a callback procedure in your component, create a variable containing the target path of the custom procedure to call. Pass this variable to a `call()` command in your component.

```
// set up callback procedure target path in configuration code
onSlideShowComplete = ?/:mycustomprocedure?;

// code in frame loop of component, executes when the slide show is complete
// only execute when callback is defined
if(onSlideShowComplete ne ??){
    // execute whatever procedure assigned to this variable
    call(onSlideShowComplete);
}
```

Setting up a common library

Common libraries are a convenient way to save assets that you may want to reuse in other FLAs. You can easily access these assets by selecting the library from the common libraries sub menu of the window menu, and drag the asset from the resulting library window to your FLA library window.

Create a new FLA save as ?Flash Lite 1.1 components?. This FLA will be your repository of reusable components and code libraries for building Flash Lite 1.1 applications. Add other components and code libraries to this FLA as you create them. By saving this FLA into the common libraries folder of the Flash CS3 application, you can easily access these assets for future Flash Lite projects.

After you complete development of the component, copy the Flash Lite 1.1 components.fla into the libraries folder of the Configuration folder. Read ?Working with Common libraries? in Flash help for more information on how to access these folders.

On Windows, the path is

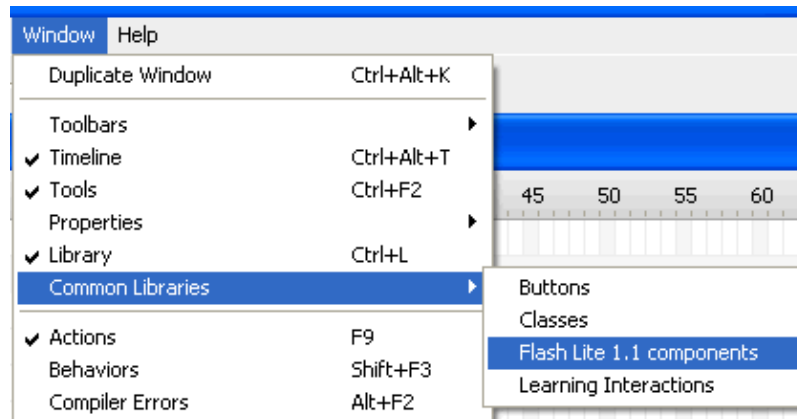
C:\Documents and Settings*username*\Local Settings\Application Data\Adobe\Flash CS3*language*\Configuration\Libraries\

On Mac OS, the path is

Hard Disk/Users/*username*/Library/Application Support/Adobe/Flash CS3/*language*/Configuration/Libraries/

Replace *username* with your user name and *language* with ?en? for English or your language preference.

The FLA will appear as an option in the common libraries sub menu of the Window menu.



Configure the slideshowlite component

1. Download the example zip archive containing the basicslideshow.fla and the Flash Lite 1.1 components.fla.
2. Place the Flash Lite 1.1 component.fla into the configuration/libraries folder (see above).
3. Open the basicslideshow.fla
4. Open the Common Libraries sub menu from the Windows menu and select ?Flash Lite 1.1 components?.
5. Drag ?slideshowlite component? movieclip asset from library window of Flash Lite 1.1 components Common Library to basicslideshow.fla library.

Build_a_reusable_Flash_Lite_1.1_component

6. Configure the first frame of the basicslideshow.fla with the following code.

```
// scope all variables and calls to the slideshow component
tellTarget("/slideshow"){
  /*
  REQUIRED
  reset all slide show variables
  */
  call("newSlideShow");

  /*
  REQUIRED
  set target paths for movie clip slides
  array must be in format s1...sx
  */
  s1 = "/slideshow1/red";
  s2 = "/slideshow1/blue";
  s3 = "/slideshow1/green";
  s4 = "/slideshow1/black";
  numSlides = 4; // total number of slides

  call("hideAllSlides"); // hide all the slides in the slide show

  /*
  OPTIONAL
  slide show property variables
  */
  slideDuration = 1000; // duration of each slide in milliseconds
  repeat = 2; // number of repeats (including first play through)
  autorun = 0; // 1 = automatically start playing

  /*
  OPTIONAL
  call back procedure
  */
  onSlideShowEnd = "[:showEndSlide]; // target path of procedure to execute at end of slide show
  onSlideChange = "[:updateSlideShowProgress]; // target path of procedure to execute on slide change
}
}
```

Add playback control procedure calls to the offscreen button.

```
on (keyPress "<Enter>") {
  tellTarget("/slideshow"){call("togglePlayPause");} // play or pause slide show
}

on(keyPress "<Left>"){
  // go to previous slide, while slide show is paused
  tellTarget("/slideshow"){call("showPrevSlide");}
}

on(keyPress "<Right>"){
  // go to next slide, while slide show is paused
  tellTarget("/slideshow"){call("showNextSlide");}
}
```

Setting up custom procedures

You can use the core procedures of the `slideshowlite` component to build extra features such as a slide progress display and displaying directions at the beginning and end of the slide show. In the `?My Procedures?` layer of the `basicslideshow.fla`, you will find two custom procedures that enhance the behavior of the slide show

The `?showEndSlide?` procedure displays the instructions slide after the slide show is finished playing, including all repeats. The `slideshowlite` component invokes the `?showEndSlide?` from the `onSlideShowComplete` callback event. This procedure gets the slide number of the last slide in the show, builds a target path for the slide movie clip associated with this slide number and hides this last slide. Then it shows the instructions slide. The procedure also clears the slide show progress text box.

```
/*
PROCEDURE
showEndSlide

invoked by onSlideShowEnd callback event in slideshowlite component
*/

tellTarget("/slideshow"){
  call("getCurrentSlideNum"); // call procedure
  currentSlideNum = getCurrentSlideNum; // get return value
  hideSlide = eval("s" add currentSlideNum); // get target path
  call("hideSlide"); // hide current slide

  showSlide = "/instructionslide";
  call("showSlide"); // show a final slide
}

showprogress_tb = ""; // clear slide show progress text box
```

The `slideshowlite` component invokes the `?updateSlideShowProgress?` procedure from the `onSlideChange` callback event. This procedure gets the current slide number and the total number of slides and prints a message, `?x of x?`, into a text box.

```
/*
PROCEDURE
updateSlideShowProgress

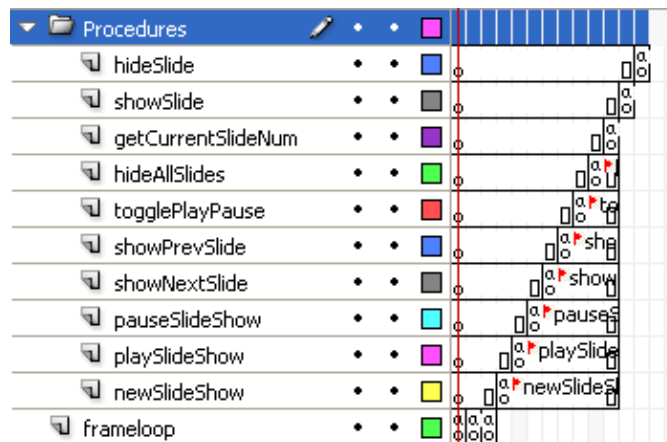
show x of total slide indicator
invoked by onSlideChange callback event in slideshowlite component
*/
call("/slideshow:getCurrentSlideNum");
currentSlideNum = eval("/slideshow:getCurrentSlideNum");
showprogress_tb = currentSlideNum add " of " add eval("/slideshow:numSlides");
```

Guidelines for organizing a Flash Lite 1.1 component

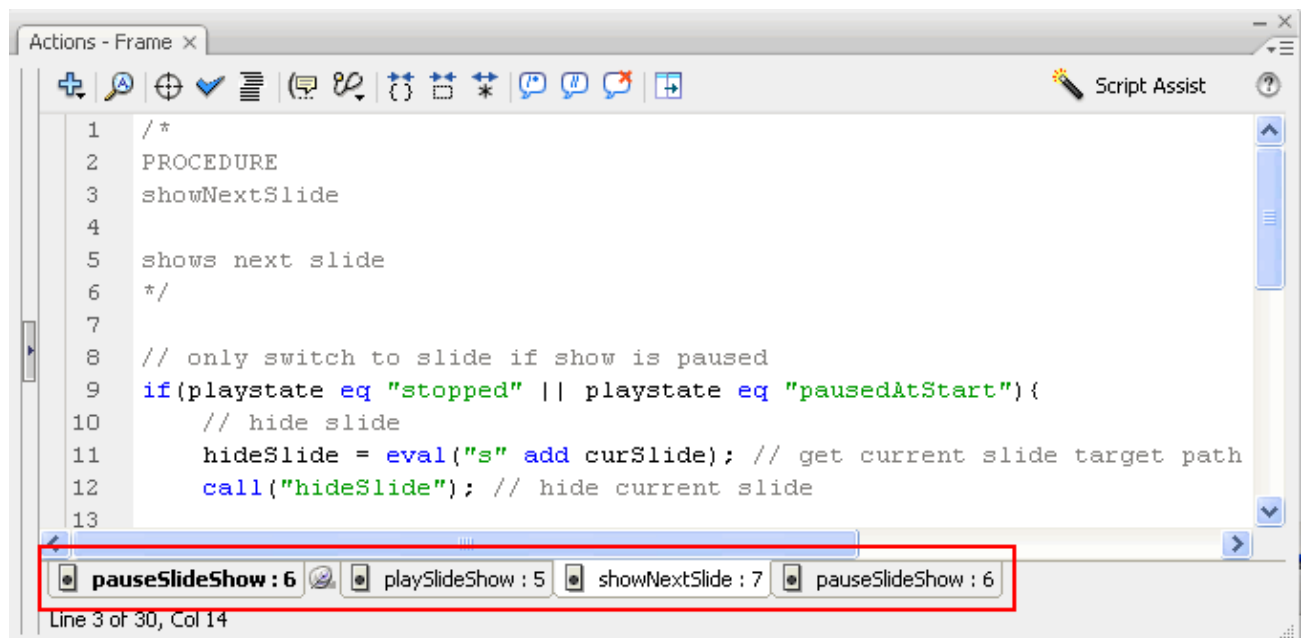
- Build your component within a movie clip. The movie clip acts as a container to hold all code and assets for the component. This makes it easy to drag the component from one FLA library window, or from a common library (see below) into your FLA library window.
- Use Flash Lite 1.1 procedures to make your code modular and to support an abstraction layer between your component code and custom code outside of the component.

Build_a_reusable_Flash_Lite_1.1_component

- Place each procedure in its own layer.



- Position each procedure in a unique frame that is not shared by any other procedure.
- Use the same procedure name for both its layer and its label. The Flash ActionScript window displays the selected layer name in the tab at the bottom of the window. You can ?pin? tabs to make more than one windows of code accessible. Flash will name each tab according to the name of the layer, which is also the name of your procedure. You can easily switch between procedures just by selecting a tab.



- Place procedures in frames that are not normally accessible to the playhead, so that Flash Lite does not inadvertently execute procedures.
- Place all procedure layers within a layer folder so you can expand and collapse the timeline view to conserve screen space.
- Add a stop() command in the first frame of the component to prevent the Flash Lite playhead from inadvertently executing other code within the component movie clip. Place other code in subsequent frames.
- If your component contains a frame loop that can either ?autorun? or wait for playback, then use an if/then condition in the first frame to either stop or play the frame loop depending upon the value of the ?autorun? configuration variable.

Build_a_reusable_Flash_Lite_1.1_component

- Include documentation or comments about the component in the first frame.
- Plan out the code and organization so that you have an easy to use abstraction layer between your Flash Lite project code and the component code.
- Document your abstraction layer carefully so you don't have to go back through your component code to understand how to use it.
- Design your component so that you or a user can configure its behavior from code outside of the component, using documented configuration variables and procedure calls.
- Design your component control/playback procedures so that a user can easily set up code for an offscreen button.

Hp3 00:36, 5 April 2008 (EEST)