Writing Persistant Data to the Device

In this unit, you will learn how to persist data on a mobile device.

Objectives

After completing this unit, you should be able to:

- ►Persist ("store") data on the phone using the `SharedObject` class
- ►Store complex data in shared objects
- ►Read data from shared objects and use the data in your application
- ►Use event broadcasting to improve code structure
- ►Send XML data from the Flash Lite player to be persisted on the server

Persisting Data in a Flash Lite Application

There are two ways to persist, or store, data from Flash Lite applications:

- **On the client**: Flash Lite applications can store data on the user's device as shared objects. These are similar to HTTP cookies, but more powerful as the may store complex data structures.
- **On the server**: Flash Lite may use `XML`, `XMLSocket`, or `LoadVars` objects to pass data to a server. The server receives and persists the data, likely using SQL with an RDBMS of some sort.

Introducing local shared objects

Local shared objects are the Flash Lite equivalent of browser cookies, but are more powerful, because they may store structured, typed data, not just flat name=value pairs.

Local shared object characteristics

Local shared objects have the following characteristics:

- ▶They are stored in a device and platform specific location
- ▶The device file extension is `.sol`
- ▶Stored or retrieved data is held within an instance of the `Object` class
  - ▪each `Object` property may be a different type (`Array`, `String`, `Date`, etc.). However, a `Function` cannot be stored.
  - ▪applications may use multiple local shared object
- ▶Stored values are written to the device when:
  - ▪the SWF unloads from the Flash Lite player, or
  - ▪the data is explicitly flushed to the device (the best practice)
- ▶Stored values are read from the device when a named object is retrieved
- ▶By default, up to 4kb in data may be stored per SWF
- ▶Each device is limited to 256kb maximum for all SWF files
  - ▪Actual capacity varies by device

Local shared object characteristics

- Local shared objects may only be retrieved by the SWF application which created them in a prior session.
  - Unlike the browser/desktop Flash player, you cannot share data between mobile SWF applications, nor different versions of the same SWF, using local shared objects.
- The device system and user may remove local shared objects, so you cannot assume one previously created still exists.

Verifying a device supports shared objects

To determine whether a device in which the SWF is running supports SO capability, refer to the `hasSharedObjects` property of the `System.capabilities` object, which will contain a `Boolean` `true` if shared objects are supported, and `false` if not.

```
if(System.capabilities.hasSharedObjects)
{
 // create and use local shared objects
}
else
{
 // persist data to the server
}
```

Creating shared objects

- The static `getLocal(name:String)` method of the `SharedObject` class creates a new `.sol` file if one does not already exist on the file system with the specified name.
- If one does already exist by that name, `getLocal(name:String)` retrieves its local data.

The following ActionScript code creates a shared object:

```
var data_so:SharedObject = SharedObject.getLocal("userData");
```

Using SharedObject methods

The following table summarizes the core methods used with shared objects:

Method	Description
`getLocal(name:String)`	Retrieves an existing SO or creates it
`flush():Object`	Writes an SO file
clear():Void	Purge the SO from the file system and memory
getSize():Number	Return current size
getMaxSize():Number	Return total space
addListener()	Designate the event handler to call when data from the named object has loaded.

Storing data in a shared object

- ▸A local shared object, once created or retrieved using the `getLocal()` method, will have a property called `data`.
- ▸The `data` property refers to the `Object` stored to disk. All data to be stored must be assigned as a property of the `data` object. If the `data` object itself is overwritten, no data will be stored.

```
var data_so:SharedObject = SharedObject.getLocal("userData");

data_so.data.firstName = "Siggy";
data_so.data.lastName = "Moonstar";

// overwrites the object, preventing storage
data_so.data = "Siggy"; // don't do this!
```

Using stored data, once loaded

- ▶Reading and writing data on a handheld device can be relatively slow.
- ▶To ensure data is available when requested, Flash Lite 2.0 requires a listener (an "event handler") be created.
  - The player invokes the listener when the device has loaded the shared object's data.
  - Do not interact with shared object data until the listener has fired.

Listener Code Example

```
function onDataLoad(data_so:SharedObject) {
 if (data_so.getSize() == 0){
  // If size is 0, new object, so initialize
  data_so.data.name = "[New User]";
  data_so.data.email = "[Add Email Here]";
 }
 // display the loaded, or initialized, data
 // in the corresponding TextFields
 name_tf.text = data_so.data.name;
 email_tf.text = data_so.data.name;
}


// create or retrieve the shared object
var data_so:SharedObject = SharedObject.getLocal("userData");

// assign the listener created above
SharedObject.addListener("data_so", onDataLoad);
```

Storing complex data

- ▶To store complex data structures in a shared object, that object must be instantiated within the shared object.
- ▶The following code creates an `Array` inside a shared object and adds data to it.

```
if(data_so.data.songList == undefined)
{
 data_so.data.songList = new Array();
}
data_so.data.songList.push(songName_tf);
```

Note: You cannot simply assign an existing complex object (Array or otherwise) to the shared object, as only a reference to it would be placed in the shared object, not the object itself and its data.

Using the flush() method

- ▸Shared objects are automatically written to the device when the SWF is removed from the Flash Lite player.
- ▸However, this is not guaranteed behavior (for example, the battery dies). The best practice is to immediately write data to the file system as it's added, using the `flush()` method.

The syntax is:

```
mySharedObject.flush();
```

Note: the `flush()` *method accepts an optional argument, specifying the minimum size, in bytes, for which the* `.sol` *file should be created. This may help in capturing the necessary anticipated storage volume on the device, even if it's not all to be used yet.*

Walkthrough 1: Persisting Data

In this walkthrough, you will perform the following tasks:

- ▶Create a shared object using the `SharedObject` class
- ▶Store the user’s name as persistent data
- ▶Display the `SharedObject` data

Posting XML data to be persisted on the server

- ▸Local shared object data can be deleted by the device user. If data must be retained permanently, it should be posted to a server and stored in an RDBMS.
- ▸One approach to persisting data on a server is to send XML from Flash Lite back to the server.

Sending XML data

- ▶The `send()` method of the `XML` class encodes the current object's data as an XML string, and sends it to the specified URL through an HTTP post operation.
- ▶Take the following steps to send XML data:

1.Create an XML object containing the XML formatted data to be sent

```
var my_xml:XML = new XML("<highscore><name>Ernie</
  name><score>13045</score></highscore>");
```

2.set the content type to text/xml

```
my_xml.contentType = "text/xml";
```

3.send the XML data to the chosen URL

```
my_xml.send("http://www.games.com/input.cfm");
```

Note: if you wish to send XML data and load an XML response in the same operation, implement the `sendAndLoad(url,result)` *method instead of* `send(url)`

Walkthrough 2: Passing XML Data Out of a Flash Lite Application

In this walkthrough, you will perform the following tasks:

- ►Send Data out of a Flash Lite Application

Using stored data, once loaded

- Events are signal notifying that either the User or System has done something.
  - Events are handled when the System calls a specified function, passing it information describing what happened (the event object).
  - You can create and dispatch your own events.

Understanding the need for event dispatching

- ►A custom class should not know the internal names of any property or method of another class.
- ►Classes should be independent ("loosely coupled" black boxes of code).
- ►If one ActionScript class uses a path keyword (`_parent` or `_root`) to interact with a property or method of another class, you've created a dependency. Renaming something within the target class could have side effects on the class referring to it. Code maintenance becomes challenging.
- ►In larger applications, in which code may be maintained over time, this problem can be avoided by having one custom class broadcast ("dispatch") events to be handled, just as built in classes dispatch events when they've done something potentially interesting. Other classes can then choose to listen for and handle the event, or not, as needed.

Note: event dispatching is an advanced technique, which benefits from careful up-front architectural planning. It is best used in more involved applications which are likely to be updated over time.

Using mx.events.EventDispatcher

The `EventDispatcher` class allows events to be broadcast from custom written code. Take the following steps to broadcast a custom event.

1. import the `mx.events.EventDispatcher` class

```
import mx.events.EventDispatcher
```

2. either implement `dispatchEvent()`, `addEventListener()`, and `removeEventListener()` methods within your class, or simply declare them as properties, to use the inherited methods from `EventDispatcher`

```
public var addEventListener:Function;
public var removeEventListener:Function;
public var dispatchEvent:Function;
```

Note: for more complex behavior, you could override and re-implement these methods; here we simply used the inherited ("derived") behavior from EventDispatcher

Using mx.events.EventDispatcher

3. in your class constructor, pass a reference to the current object (`this`) to the `initialize()` method of `EventDispatcher`. Doing so allows your class to use the methods of this class.

```
import mx.events.EventDispatcher
class MyClass extends MovieClip
{
 public var addEventListener:Function;
 public var removeEventListener:Function;
 public var dispatchEvent:Function;
 public function MyClass()
 {
  EventDispatcher.initialize(this);
 }
}
```

Using mx.events.EventDispatcher

4. where an event should be broadcast, create an event `Object` with `type` and `target` properties, and other information properties as appropriate. The `type` is your name for your event. The `target` is a reference to the object dispatching the event (usually `this`).

```
class MyClass extends MovieClip
{
 ...
 private function onDataLoad():Void
 {
  var eo:Object = new Object();
  eo.type = "dataLoaded";
  eo.target = this;
 }
}
```

Using mx.events.EventDispatcher

5.call the `dispatchEvent(event)` method to dispatch your event object

```
class MyClass extends MovieClip
{
 ...
 private function onDataLoad():Void
 {
  var eo:Object = new Object();
  eo.type = "dataLoaded";
  eo.target = this;

  dispatchEvent(eo);
 }
}
```

6.register one or more listeners for your event

```
addEventListener("dataLoaded", Delegate.create(this,
  showDataScreen));
```

Using mx.events.EventDispatcher

- Once `EventDispatcher` has been initialized and set up for a class, `dispatchEvent()` can be called, and passed an event object, in any relevant method.
- Anything listening for that `type` of event will receive the event object as an argument to its event handler function. The object will let the listener know "who" (`target`) broadcast what (`type`) event, and may also have extra properties filled with information about what happened.

Walkthrough 3: Understanding Event Dispatching

In this walkthrough, you will perform the following tasks:

- ▸Update the Hunt class to dispatch events
- ▸Listen for and react to dispatched events

Summary

- ▸Local shared objects are like a "Flash Cookie", but better than a cookie, because they store structured data
- ▸Invoking the `SharedObject.getLocal(name)` method causes a .sol file for the specified name to be created on the device, or retrieved if it already exists, and the shared object to be created in memory
- ▸Local shared object data must be assigned to properties of the `data` object of the shared object
- ▸The best practice is to `flush()` shared data to disk once it has been assigned to a property of the `data` object
- ▸If a complex object (`Array`, `Date`, etc.) is to be stored, it must be created as a property of the `data` object, then populated with its data
- ▸A shared object can register a listener to be invoked when its data has fully loaded, using its `addListener(so_name, function)` method

Adobe

Summary

- ▶Custom events can be broadcast using the `EventDispatcher` class. To do so, you must
  - import and initialize the `EventDispatcher`
  - implement `addEventListener()`, `removeEventListener()`, and `dispatchEvent()` methods (or simply declare as properties to use inherited behavior from `EventDispatcher`)
  - create event objects, dispatching them with the `dispatchEvent(event)` method
  - register event listeners using `addEventListener()`
- ▶Data can be persisted on a server by using the `send()` method of an `XML` object to send its data to a specified URL

Adobe