

by Giorgio Natili



- www.mxml.it

Created
25 February 2008

Requirements

Prerequisite knowledge	Required products	Sample files	User level
Basic knowledge of building Flash applications.	Adobe AIR Adobe Animate CC	Download sudoku_source.zip (1706 KB) Download SudokuPuzzle.air (2001 KB) Download Sudoku_as3.zip (3117 KB)	Intermediate

The introduction of Adobe AIR opens many doors for developers who have been using technologies such as Adobe Flex, Adobe Flash, HTML, and JavaScript and now want to create desktop applications using languages that they are familiar with. The aim of this article is to demonstrate how easily and quickly existing browser-based Flash applications can be migrated to AIR. In fact, when you want to migrate an ActionScript 3.0 application that follows best practices, you don't need to modify the existing code at all—but you can update it to improve the desktop experience powered by AIR.

Note: The example used in this article is a browser-based Sudoku game originally created by Bob Sander-Cederlof. The game is written entirely in ActionScript 3 and can be played at: www.txbobsc.com/misc/sdk/. Bob graciously provided the source code so I could build the game on Adobe AIR. The source code is included here for reference. Figure 1 shows the original game and the AIR version.

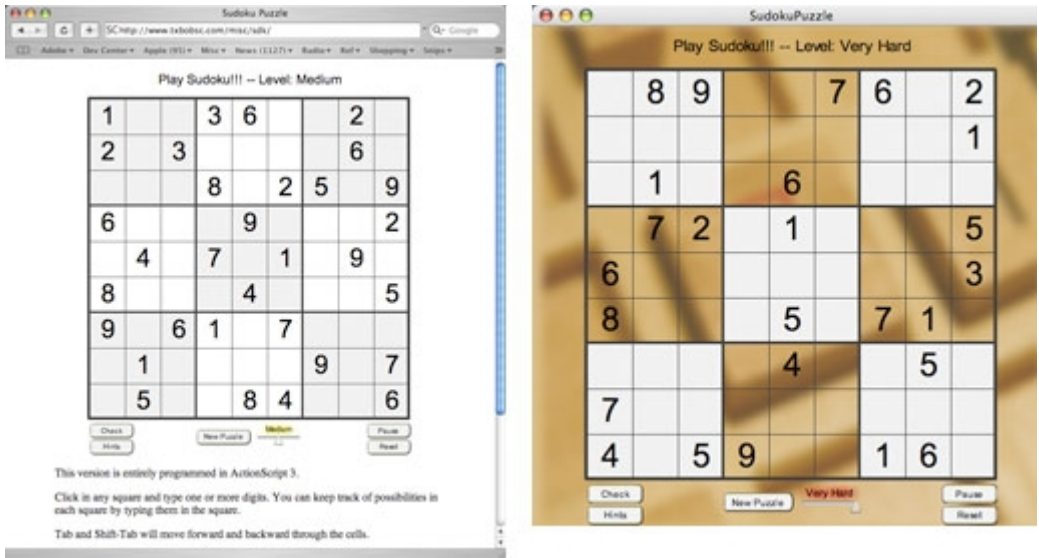


Figure 1. The Sudoku application in the browser and on the desktop.

Exploring the original Flash application

The Flash application that I chose to migrate is an online Sudoku game. The application loads new puzzles from an XML file or from external resources. You can select the level of difficulty using a slider control. By clicking the Pause button you can change the source of the puzzles and the size of the hints, while the Check button reveals any errors in your potential solution (see Figure 1).

The original code is very clean. The developer followed best practices when building the Flash application. Each functional component is contained in a class file, and the classes are loosely coupled, so I didn't need to read and understand all the code to start building the application on AIR.

Note: When classes are loosely coupled, the developer has minimized the dependencies between classes, and as a result, reduced the risk that a change in one module will force a change in another module. The loose coupling of classes in the original Flash application made it even easier to migrate the application to AIR.

I made only one change to the structure of the original application: I removed all the code from the first frame of the SudokuPuzzle.fla contained in the Sudoku_as3.zip and put it in the document class of the AIR application, `Sudoku.Main` (see Figure 2). I made this change in order to keep the code outside of the FLA file, because I prefer to maintain a high level of separation between the business logic of an application and its GUI, and because I have implemented some events to enhance communication between classes (I always define the listener for these events as `private` members).

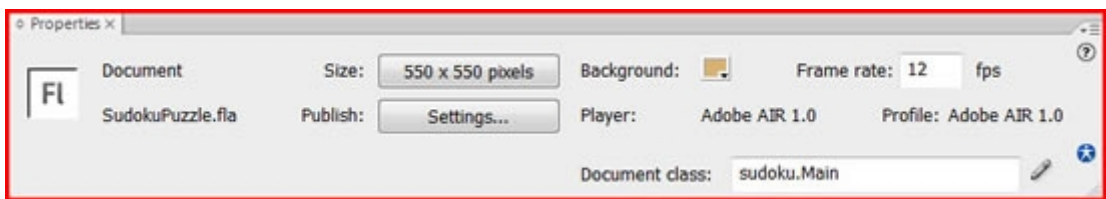


Figure 2. Definition of the document class.

Note: All the changes I made to the code to convert the Flash application to AIR are marked as follows in the sample code that accompanies this article:

The structure of the original application is quite simple—each class represents the UI elements or the logic used to create the puzzles at runtime (see Figure 3).

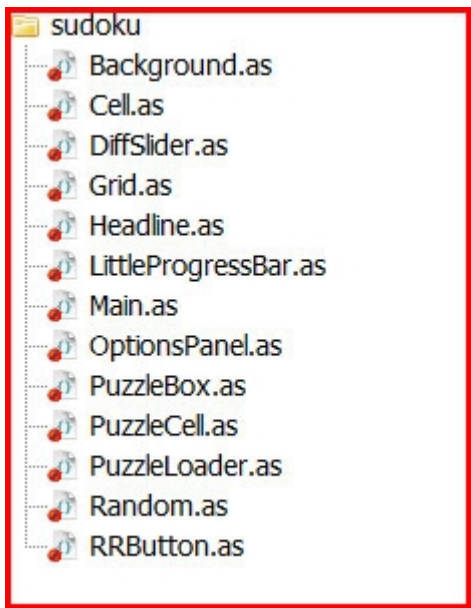


Figure 3. Application structure and classes.

As I converted the application to AIR, I wanted to improve the desktop experience by adding the following capabilities:

- Online / offline management
- Local storage of new puzzles

To accomplish this, I needed to work on two classes: `Main.as` and `PuzzleLoader.as`.

In the original application the options panel is used to set the new puzzle download preferences (see Figure 4). This is the option I'll update in AIR for an online and offline game experience.



SUDOKU ON ADOBE AIR: MIGRATING A FLASH APPLICATION TO THE DESKTOP

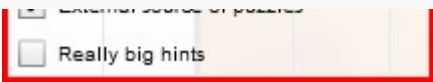


Figure 4. The original Options panel.

Adding desktop functionality

When you start to migrate an existing Flash application to AIR, you have to ensure the application can run without an Internet connection. For this application, I will add two simple features to enhance the desktop experience of this application:

- Handling online/offline status
- Updating the stored XML file

The first feature is used by the application to disable online updates when there is no Internet connection. The second feature is used to update the XML file that stores the puzzle data loaded from the web. This allows the user to run the application when not online. I imported the AIR classes needed to handle the online / offline status into the Main.as file. In the FLA file, I embedded the library of the Service Monitor Shim component as well as an instance of the Service Monitor Shim component from the components panel (Ctrl+F7 or Windows > Component) in order to make the `URLMonitor` class working properly.

The constructor of the application defines a URL monitor that I use to enable or disable the External source of puzzle option defined in the option panel

```
monitor = new URLMonitor(new URLRequest(ON_LINE_OFF_LINE));
monitor.addEventListener(StatusEvent.STATUS, handleOnLineOffLineStatus);
monitor.start();
```

The monitor instance uses the `ON_LINE_OFF_LINE` constant to check the status, the value stored in this constant is www.adobe.com (we can safely assume that this URL won't disappear anytime soon).

The listener defined for the `StatusEvent.STATUS` event controls the value of the application `puzzleSource` property (1 = puzzles stored locally in an XML file, 2 = JavaScript, 3 = puzzles located online on special sites). It also handles the status of the checkbox used to change the loading preference:

```
if(!monitor.available){
    if (puzzleSource == 3) {
        puzzleSource = 1;
    }
    optionsPanel.setOption("External", false);
}
```

In the constructor, I also added the following code to copy the sudoku.xml file contained in the AIR installer to the local storage directory of the application:

```
var storedFile:File = File.applicationStorageDirectory.resolvePath(STORED_FILE_NAME);
if(!storedFile.exists){
    var file:File = File.applicationDirectory.resolvePath(LOCAL_FILE_NAME);
    if (file.exists) {
        puzzleData = new FileStream()
        puzzleData.addEventListener(Event.COMPLETE, completeDataCopy);
        puzzleData.openAsync(file, FileMode.READ);
    }
}else {
    initGame();
}
```

The application copies the file only the first time the application is launched. This avoids the security restrictions that would apply when updating the XML file.

The `File.applicationStorageDirectory.resolvePath` instruction tells the AIR runtime to get a reference to the XML file in the local storage folder of the application. If the file doesn't exist, I create a new `FileStream` object and use it to copy the file to this location.

The `FileStream` object is opened asynchronously, which lets the application run as the file copy completes. The listener defined for the `Event.COMPLETE` event reads the XML data stored in the sudoku.xml file and writes it to the `FileStream` object already opened:

```
var xdata:XML = XML(puzzleData.readUTFBytes(puzzleData.bytesAvailable));
var folder:File = File.applicationStorageDirectory;
var file:File = folder.resolvePath(STORED_FILE_NAME);
var newFile:FileStream = new FileStream();
newFile.open(file, FileMode.WRITE);
newFile.writeUTFBytes(xdata);
newFile.close();
```

In the `onPauseButton` event the code handles the online/offline status by enabling or disabling the checkbox in the Options panel:

```
optionCB1.enabled = monitor.available;
```

To update the XML file that contains the puzzles I have created a custom event that is dispatched from the `PuzzleLoader` class.

The event handler receives and stores in its properties the new puzzle XML data from the web and the level of this new puzzle. The event reaches the `Main` class and then the `onPuzzleUpdate` method registered as a listener for the `UpdateStoredPuzzlesEvent.UPDATE` event handles the data and performs the update of the local file.

Puzzles are stored in the XML file according to their difficulty level (easy, medium, hard, or very hard).

The custom event is defined in the `UpdateStoredPuzzlesEvent` class and is dispatched from the `PuzzleLoader` class when a new puzzle is loaded from the remote web site:

```
dispatchEvent(new UpdateStoredPuzzlesEvent(_level, "<key>" + dotPuzz + "</key>"));
```

I have defined a listener for this event in the `Main` class:

```
html.addEventListener(UpdateStoredPuzzlesEvent.UPDATE, onPuzzleUpdate);
```

The `onPuzzleUpdate` listener receives the data and opens the `FileStream` object aysynchronously:

```
currentXML = e.puzzle;
currentLevel = e.level - 1;
var folder:File = File.applicationStorageDirectory;
var file:File = folder.resolvePath(STORED_FILE_NAME);
updateStream = new FileStream();
// Define the listener for the FileStream
updateStream.addEventListener(Event.COMPLETE, onPuzzleFileUpdate);
//Open the file
updateStream.openAsync(file, FileMode.UPDATE);
```

The listener doesn' t replace the XML file. Instead, it clears the data for the specified level of difficulty, and replaces it with the updated data.



SUDOKU ON ADOBE AIR: MIGRATING A FLASH APPLICATION TO THE DESKTOP

```
updateStream.writeMultiByte(xdata, "utf-8");
updateStream.close();
```

I also made a small change to the code that loads a new puzzle in the `PuzzleLoader` class. If you run the application on a Windows PC (Vista or XP) you can load the XML file with an instance of the XML class that reads the data from the application storage folder using the `File.nativePath` property; but when you run the application on a Mac this fails silently.

If you use the `File.url` property it works without any exception, but I decided to use a `FileStream` object because in my experience with Java when you depend on the operating system too closely your application may encounter portability issues.

To avoid such issues, I use a `FileStream` object to read the data handling in the `Event.COMPLETE` and `Event.PROGRESS` events.

Creating the AIR application file

Adobe AIR update for Flash CS3 Professional makes it easy to build and test a Flash application on AIR. After installing the update, you'll find the Air Service Monitor component in the Components panel (see Figure 5).

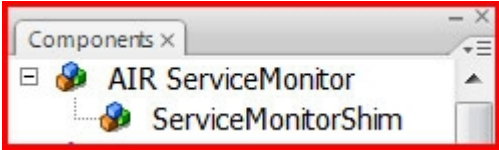


Figure 5. The AIR Service Monitor component is installed with the Adobe AIR update for Flash CS3 Professional.

You'll also find two new menu options: Commands > AIR - Application and Installer Settings and Commands > AIR – Create AIR File. Both of these items facilitate the creation of the AIR file.

Select Commands > Air - Application and Installer Settings to update applications settings, including the file name, version number, description, ID, copyright, window style, and so on (see Figure 6).

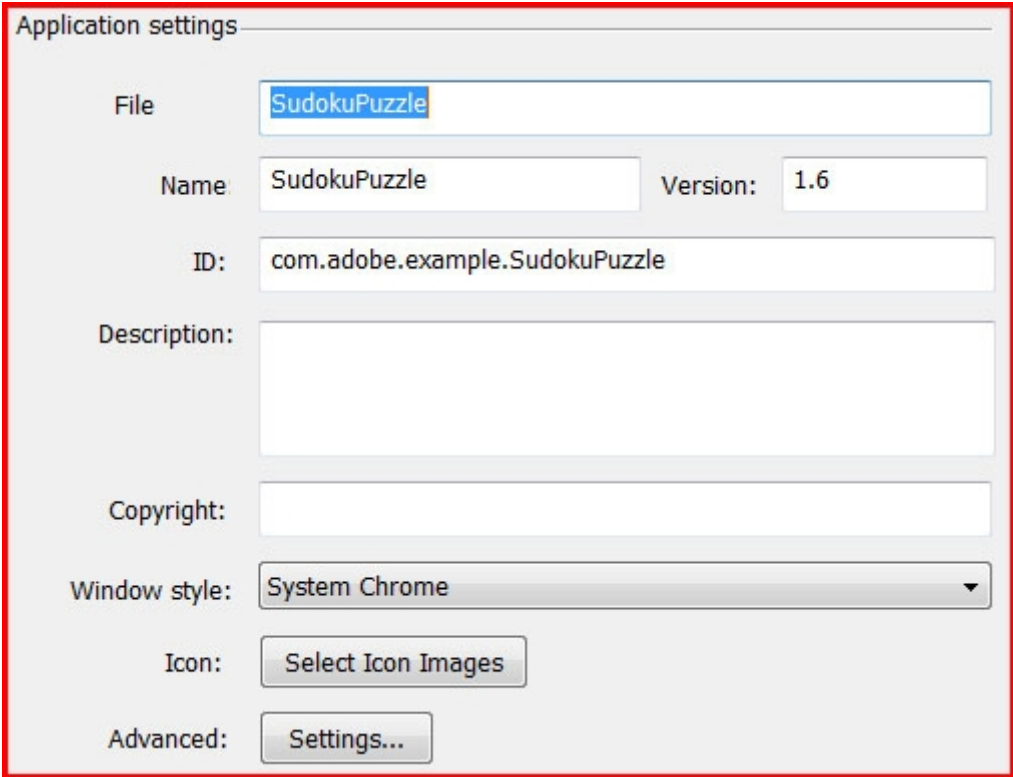


Figure 6. AIR Application settings

The Application and Installer Settings item looks like a classic Flash component that is added when you install the Adobe AIR Update for Flash CS3 Professional; you need to embed it in your library when you are planning to use the network capabilities of AIR (represented by the classes `ServiceMonitor`, `SocketMonitor`, and `URLMonitor`).

Keep the end user in mind as you consider these fields. When installing a desktop application many users want to know as much about the application as possible before proceeding. The version field is particularly important. When a user installs an AIR application the installer will prompt to replace one version with another, for example, version 1.0 with version 1.1. Your users should be aware of such a version change when an application is about to be replaced.

To customize the icons used by your application, click Select Icon Images and use the dialog box (see Figure 7) to choose the icon files (in PNG format) that you want to use. You can specify icons of various sizes to help users recognize the application easily, whether it is on the desktop, the program menu, or file explorer.

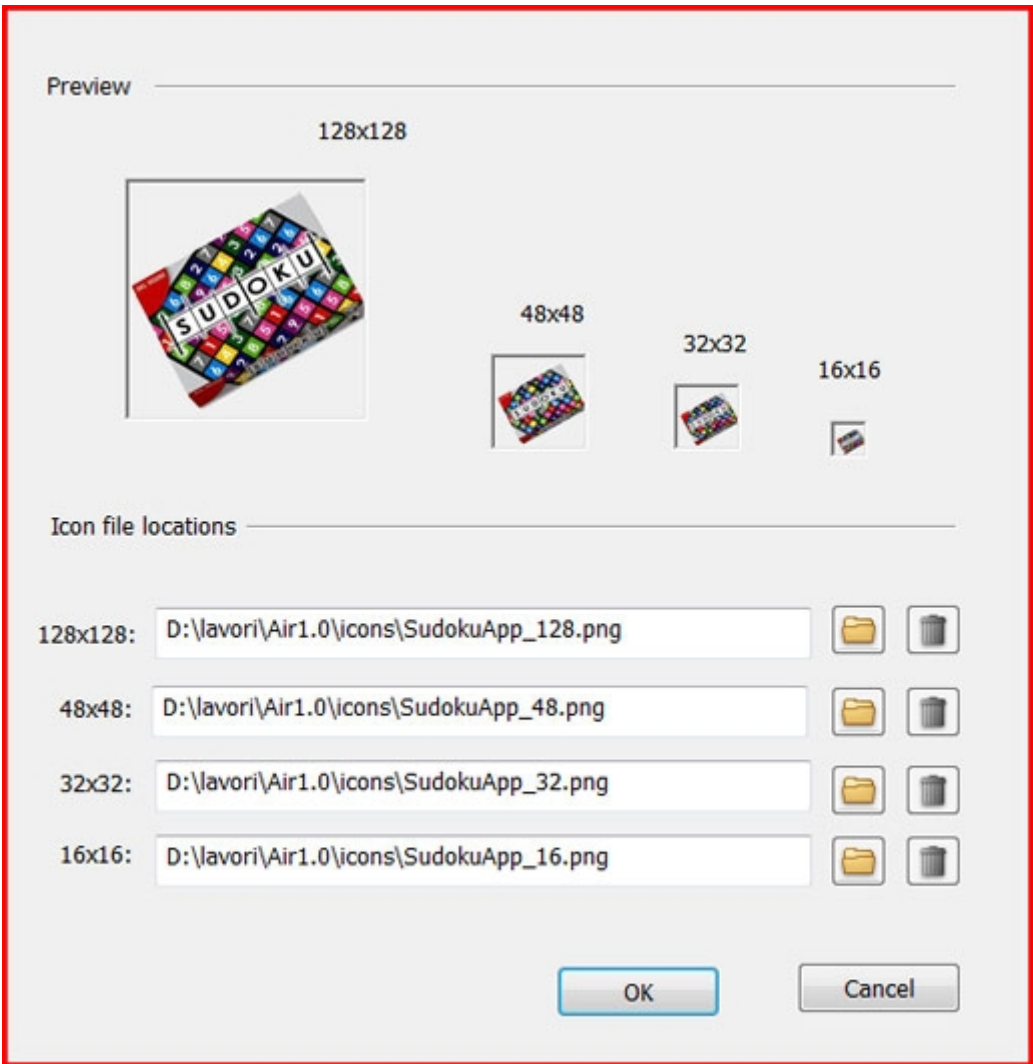


Figure 7. The Icon images dialog box.

Note: In a desktop environment, icons are very important. An icon is part of your brand and helps distinguish your application from the other applications available. Whenever possible, add a well designed, easy-to-remember, and distinct icon when you are migrating your browser-based application to the desktop.

Click Advanced: Settings to set the initial size of the application window, the x and y position of the application on the screen, the install folder, the program menu short cut, the Custom Update UI option, and other options.

The Custom Update UI option determines what happens when a user installs an AIR file for an application that is already installed. By default, AIR displays a dialog box that allows the user to update the installed version with the new version in the AIR file. If you want the application to have complete control over its updates select this option.



SUDOKU ON ADOBE AIR: MIGRATING A FLASH APPLICATION TO THE DESKTOP

You don't need to buy a certificate when you start to develop an AIR application, because the Adobe AIR update for Flash CS3 Professional allows you to create a self-signed digital certificate for your testing cycle. However, when you are ready to distribute your application you will need to sign up with a digital certificate authority. The two main certificate authorities for Adobe AIR applications are Verisign and Thawte. For more details on signing your application, see Todd Prekaski's article, [Digitally signing Adobe AIR applications](#).

Note: You can also prepare an AIR package that will be signed later, however the resultant AIR file can't be run on a user's system. You can use the AIR Developer Tool (ADT) to package and sign the application. ADT is a Java program that can be run from the command line or with a build tool such as Ant. The Adobe AIR SDK includes command-line scripts that execute the Java program for you so you can package and sign an AIR file in a single step using the ADT `-package` command. This process can be very useful when the certificates for signing are installed on another PC or when you create an application for which the final customer will provide the certificates needed to sign it.

It is important that you sign your application—in fact, you won't be able to package your application and distribute it using Adobe tools if it is not signed. Consider what would happen if users could not verify the integrity of code published on the Internet. They would not know if any application they were about to install was from someone trying to create a backdoor to perform any number of potentially malicious activities on their systems.

The Included files section of the AIR - Application and Installer Settings dialog box is used to add files to your AIR installer (see Figure 8). By default, this box includes the SWF file that contains your application and the XML file that describes your application to the AIR runtime. If your AIR app requires any additional files to run, include them explicitly by clicking the plus button.

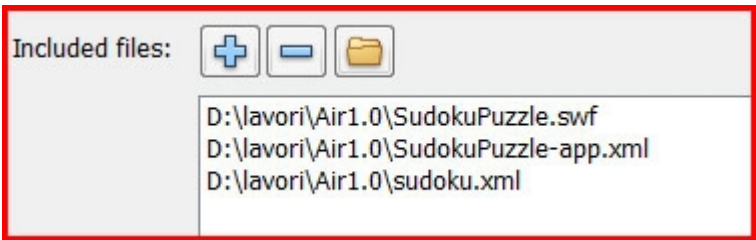


Figure 8. The Included files list.

Now that you have all the necessary files for you AIR application, you are ready to create your AIR file and distribute your application. To do this, click Publish Air File on the Air - Application and Installer Settings dialog box or select Commands > Air – Create Air File.

You can find your packaged AIR file in the same directory that contains the dependent files of your application.

Where to go from here

With Adobe AIR update for Flash CS3 Professional installed on your PC, you can test your application by pressing CTRL + Enter or selecting Control > Test Movie from the menu bar. In addition to these tests, I recommend performing additional tests after installing the application on your application's target operating systems.

As you can see, very little code was needed to convert a Flash application to AIR and the added code is quite simple. If you wanted to, you could enhance this application with more functionality. For example, you could use local variables to resume the status of the last puzzle the user played (using `EncryptedLocalStore`), add drag-and-drop functionality to enable the user to load a new puzzle by dragging a puzzle URL onto the application, or use local SQL support or a text file to keep track of the user's running score. Perhaps you can think of even more things to do with Sudoku on AIR.

For more information about Adobe AIR, visit the [product page](#).

For more inspiration, check out the sample applications in the Adobe AIR Developer Center for [Flash](#), [Flex](#), and [HTML/Ajax](#), as well as the [apps showcase](#).

To start building Flash apps on Adobe AIR go to the [Getting Started](#) section of the Adobe AIR Developer Center for Flash.

