

---

# **S60 Platform: How to Prototype Applications with Flash**

**Version 1.0**  
July 12, 2007

**S60** platform

## Legal notice

Copyright © 2007 Nokia Corporation. All rights reserved.

Nokia and Nokia Connecting People are registered trademarks of Nokia Corporation. Java and all Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. Other product and company names mentioned herein may be trademarks or trade names of their respective owners.

### Disclaimer

The information in this document is provided “as is,” with no warranties whatsoever, including any warranty of merchantability, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. This document is provided for informational purposes only.

Nokia Corporation disclaims all liability, including liability for infringement of any proprietary rights, relating to implementation of information presented in this document. Nokia Corporation does not warrant or represent that such use will not infringe such rights.

Nokia Corporation retains the right to make changes to this specification at any time, without notice.

### License

A license is hereby granted to download and print a copy of this specification for personal use only. No other license to any other intellectual property rights is granted herein.

## Contents

<b>1.</b>	<b>Introduction .....</b>	<b>5</b>
<b>2.</b>	<b>Prototyping in UI design.....</b>	<b>6</b>
2.1	Adobe Flash in UI prototyping.....	6
2.2	Process flow .....	6
2.3	Utilization of prototypes.....	7
2.4	Summary .....	8
<b>3.</b>	<b>Practical issues of developing with Adobe Flash Lite.....</b>	<b>9</b>
3.1	Flash prototyping project considerations .....	9
3.1.1	Required skills .....	9
3.1.2	Tools .....	9
3.2	Creating a flexible prototyping environment.....	11
3.2.1	Development for reuse .....	11
3.2.2	Multiperson projects .....	11
3.3	Creating a Flash prototype.....	12
3.3.1	Implementation issues.....	12
3.3.2	Debugging methods .....	15
3.3.3	Testing of prototypes .....	16
3.3.4	Small memory optimization .....	17
3.4	Extending the capabilities of Flash .....	18
3.4.1	fscommand().....	18
3.4.2	Creating external supporting code .....	19
3.5	Limitations of Flash .....	21
3.6	Summary .....	21
<b>4.</b>	<b>Case study: UI prototyping at SYSOPENDIGIA.....</b>	<b>23</b>
4.1	Summary .....	24
<b>5.</b>	<b>Summary .....</b>	<b>25</b>
<b>6.</b>	<b>Terms and abbreviations.....</b>	<b>26</b>
<b>7.</b>	<b>References .....</b>	<b>27</b>
<b>Appendix A.</b>	<b>An XML loader class .....</b>	<b>28</b>
	<b>Evaluate this resource.....</b>	<b>31</b>
	<b>Poster page.....</b>	<b>32</b>

## Change history

July 12, 2007	Version 1.0	Initial document release

---

## 1. Introduction

Up until recent years, user interface design projects have involved only UI design and usability testing tasks. The user interface design has been verified by paper prototypes, which are simple to create but lack the level of detail and complexity of interaction mechanisms. However, at the same time rapid prototyping tools have become a potential tool to enhance the design process. With rapid prototyping tools, more visually compelling and highly interactive prototypes can be created, which in turn has made UI prototypes available also for other purposes than usability testing.

Typically UI design, prototyping, and graphics design are done in parallel in a project, that is, the prototype and the UI design are ready almost at the same time at the end of the project. Because the prototype needs to be created quickly (typically in 3–4 weeks), its correctness needs to be verified easily, and changes must be done rapidly, prototype development imposes some special requirements for the prototyping tool. This is even more important when prototyping in a mobile environment, because the processing capacity is much more limited than in the desktop environment.

There are several UI prototyping technologies that are suitable for developing mobile prototypes:

- *Microsoft PowerPoint®*: a tool for creating visual slideshows of the UI very quickly.
- *Adobe Flash*: a tool for creating visually appealing multimedia applications [1]. The tool is widely used in the industry and it enables creating prototypes for mobile devices
- *Python*: a programming language for rapid application development [2]. The language can also be used for prototyping in mobile devices.
- *Java, C++*: powerful programming languages that are typically used for application development. Naturally, they are also suitable for UI prototype development, but the development effort is big.

This document focuses on UI prototyping with Adobe Flash, due to its widespread utilization and capabilities. Originally, the tool was developed for PCs, but currently there is also a version — Flash Lite [3] — which is used to create applications for mobile devices.

In principle the development methods are the same in the PC and mobile versions of the tool, but when developing for a mobile platform, some special considerations need to be taken into account. In this document the focus is on applying Adobe Flash Lite in developing UI prototypes for S60 mobile devices. Even though the focus is on the S60 platform, most of the techniques in this document are applicable to other mobile platforms, as well.

---

## 2. Prototyping in UI design

### 2.1 Adobe Flash in UI prototyping

Adobe Flash is a tool for creating rich multimedia applications for PC and mobile environments. It provides support for a wide range of graphics, video, and audio formats, and deployment of the formats is very easy with the tool (it should be noted that SVG is not supported, however). It makes the tool a potential choice for creating UI prototypes. Naturally, the tool can be used to create product quality applications by using more effort in the development.

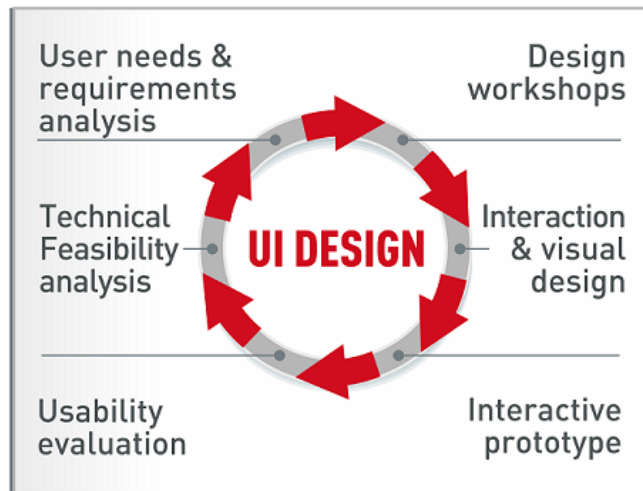
There is also a Flash Lite version of the tool (currently at version 2.1, version 3.0 is just behind the corner [4]) which is targeted for mobile devices. Currently there are a wide range of mobile devices supporting Flash Lite, including S60 devices. Support for S60 devices and the wide customer base of existing S60 devices make the tool very attractive for creating UI prototypes. In principle, Flash Lite is similar to the desktop versions, but in practice there are some differences that need to be taken into account in the development. For example, some functionality might not be available in the Lite versions, and vice versa. Flash Lite has some functionality that the desktop version of Flash does not support. Therefore, developers switching from the Flash world to Flash Lite need to familiarize themselves with the possibilities provided and restrictions set by the tool.

One advantage of Flash is that the development methods are similar regardless of the target platform where the prototype is run. For example, the same code can be used when developing to a PC and when developing to an S60 device. This enables running S60 prototypes in a PC in cases where the processing power of available devices is not sufficient, or the devices do not have suitable interaction mechanisms that the UI design would require (simulating these interaction mechanisms is much easier in a PC than in a mobile device). Also, a prototype for S60 devices can be easily distributed to parties interested in the UI design, since they do not necessarily need a suitable S60 device to run the prototype.

### 2.2 Process flow

Application UI design is done early in the software development process, typically after analyzing the business need and creating the high-level requirements. Quality of the UI design greatly affects the acceptance of the application by end users. For this reason, UI design needs to be conducted carefully, and enough time needs to be reserved for it.

There are several user-centered design processes that can be utilized in UI design. When considering UI prototyping, the working methods are quite similar, and the exact details of the applied process are not relevant.



**Figure 1: A typical UI design process**

Prototyping is usually carried out at the UI design phase (Figure 1). Typically UI design is done in iterations which consist of interaction design, usability testing, graphics design, and prototyping. The tasks are done in parallel and each of the development tracks get input and give feedback to other tracks. For example, the interaction designer might notice that a design idea is not working when it is implemented in a prototype running in a device (thus saving time from testing it with end users), or the contrast in a graphics icon is not sufficient on the display of a mobile device. It is notable that typically formal UI specifications are not created in the first stages of the design, thus emphasizing the need for informal communication. In larger projects there might be a need to evaluate the design iteratively several times with end users.

There is usually quite limited time reserved for the UI design activities, for example 2–4 calendar weeks. During this time the customer is usually very eager to see how the design proceeds and give comments and feedback about the desired outcome. This requires that there should be intermediate releases of the UI design and the prototype during the project, and the customer should be able to run the prototypes without special arrangements. In practice this means that the customer needs to be able to run the prototypes in his/her PC without special software.

When the prototype (and UI design) are ready, usability tests are arranged. During the usability tests the prototype might need to be corrected quickly so that the errors discovered in a test can be corrected to the next test session. In addition, after the tests the prototype might need to be fixed to include major design changes, and to keep it aligned with the UI design. In conducting usability tests for the mobile environment, including S60, it is important to run the prototype in a real device because the test results are more accurate in the real environment of the application.

## 2.3 Utilization of prototypes

UI prototypes can be used for various purposes:

- *Usability tests:* A natural, and the most common, purpose of prototypes is to use them in testing the usability of a UI design. In usability tests the prototype should give end users the feeling that they are using the final product. This requires that exceptional use cases and interaction flows are implemented, too.
- *Communicating design ideas:* Design ideas can be easily communicated to the customer by showing the prototype and discussing its functionality. It has been noted that customers are more eager to comment on a highly visual UI

prototype than a lengthy UI specification, thus making it easier for the customer to participate in the UI design process. In addition, customers might not really know what they want before seeing a tangible interactive prototype.

- *Demonstration purposes:* In many cases a UI prototype is used to raise more funding for a product concept. With a UI prototype it is much easier to convince the managers responsible for the budget about the innovative nature of the concept than with a standard PowerPoint® presentation. The same prototype used in usability tests can also be used to demonstrate the concept later on.
- *Verifying technical feasibility:* One special domain where prototyping can be used is to verify the feasibility of a certain technical issue in the application. This type of prototyping usually requires programming with Symbian C++ in S60 devices, and is outside the scope of this document.

## 2.4 Summary

When using prototyping in UI design projects, special considerations need to be given to the following issues:

- Prototype development should bring visible results quickly.
- The prototype developer should be able to create prototypes from informal and brief (and possibly ambiguous) designs.
- Prototypes should run without the need to install special software.

Flash Lite is a potential tool in developing UI prototypes for S60 devices. With the tool, complex and appealing prototypes can be created with a relatively small effort.



## 3. Practical issues of developing with Adobe Flash Lite

This chapter discusses developing UI prototypes with Adobe Flash. The emphasis is on introducing the most fundamental issues that are common in almost every Flash UI prototyping project and issues that need special consideration.

### 3.1 Flash prototyping project considerations

#### 3.1.1 Required skills

Prototype development with Flash can basically be done in two ways:

- Creating an application framework and basic interaction with frames, scenes, and tweening.
- For complex interaction mechanisms or advanced animation (for example, physics modeling) ActionScript is needed. ActionScript must be used if the prototype contains dynamic content which is changed according to the input given by the user.

ActionScript is essentially an object-oriented programming language. To fully utilize the capabilities of the language, a prototype developer should have enough software development skills. Also, prototypes need to be created in a quick pace, so the developer should have skills to quickly design an application that does not become too flaky when new features of the developed UI are added.

On the other hand, UI prototyping projects do not usually have formal specifications that could be used as a basis for developing the prototype. The prototype developer should have a basic understanding of the interaction mechanisms of the S60 platform and insight into usability design paradigms so that UI designers do not need to explain every detail of the UI early in the design work.

As the name suggests, UI prototyping requires basic skills in using graphics tools. Even though a graphics designer usually creates the graphics for the prototype, there are situations where small fixes to the graphics are needed. For example, the color range of an icon might need to be changed from blue to red, and the graphics designer might currently be involved in another project. The project proceeds better if the prototype developer is able to make this change without the help of the graphics designer.

#### 3.1.2 Tools

To use Flash efficiently, certain content development kits (CDKs) [5] and/or device profiles might be required. They can be downloaded free of charge from Adobe's Web site. Content development kits provide documentation, examples, and general UI elements that can be utilized in projects.

Mobile device profiles are used to test the Flash application in a PC. The device profiles provide mock-up device emulations for various S60 device models (among others) that can be used to test how the application would run in an actual device without installing it to the device itself (see Figure 2 for an example).



**Figure 2: An example mock-up device in the Flash Lite emulator**

Device profiles define the hardware properties of each supported S60 model, so in principle the device profiles should help to create applications that run in S60 devices correctly. However, a simulation with a device profile is not 100% accurate so the application needs to be tested extensively in a real device as well. One problem with the device profiles is that they tend to run out of memory more easily than real devices. Therefore, there should be real S60 devices available for testing during the prototype development. The device models used should be the same as in the final use of the prototype (for example, the models used in usability tests) because there are subtle differences between S60 device models, and unexpected results may occur in some models.



**Tip:** Device profiles are simple XML files that can be modified if an existing profile does not seem to work perfectly or does not fully suit the project needs. For example, customizing a profile for a non-standard screen resolution is a trivial task. Device profile XML files are located in the folder `%ProgramFiles%\Macromedia\Flash 8\en\Configuration\Mobile\Devices`.

A graphics tools should be installed because the prototype development requires manipulating graphics. Adobe bundles Flash in a Creative Suite application kit that also includes Adobe Illustrator and Adobe Photoshop graphics tools [6]. Flash can utilize only Illustrator's vector format perfectly, so in practice Illustrator needs to be used in drawing vector graphics. For bitmap graphics there are also other tools than Photoshop, many of which are also free of charge.

If native code is created (as described in Section 3.4.2, "Creating external supporting code"), a suitable S60 software development kit (SDK) needs to be installed, which can be freely downloaded from Forum Nokia [7]. It should be noted that the SDK used to create the native application should match the hardware device intended to run the application.

## 3.2 Creating a flexible prototyping environment

### 3.2.1 Development for reuse

S60 prototyping projects usually require user interface elements that are the same in almost all prototypes, for example lists, menus, and notifications. To speed up the development work, a UI component library should be created. In Flash, UI components can be created with two methods: creating a real Flash component or composing a collection of movie clips which define a UI component.

Creating a component library with either method requires judging the needed effort against the usefulness of the solution. When developing a component library, the following issues should be considered:

- Usually the component library cannot be used as such in every project and it requires slight modifications.
- Making changes to movie clips is easier and possibilities for variation are bigger.
- Real Flash components hide the implementation of the component and the developer does not need to know how it is implemented.

A good guideline in creating a component library would be to first make a collection of user interface elements from movie clips and develop their implementation according to the project needs. When a certain UI element seems to be working properly without the need for major rework, it can be modified as a real Flash component.

There is a similar logic in every S60 Flash application that is not directly visible to the user, for example changing between views, using timers, or managing key presses. This logic can be implemented in an application framework class library (see Section 3.3.1.2, “Object-oriented programming,” for a discussion of classes). It has been noted that this kind of framework library significantly speeds up the development and produces higher quality prototypes. However, the tradeoff is that creating the library takes quite a lot of effort and requires that the framework users have knowledge of software development.

### 3.2.2 Multiperson projects

Usually UI prototypes are of such size that they are the easiest to do by one prototype developer. However, if a prototype is very large, there is a tight deadline to meet, or several prototypes or different variants are created, several developers can work in the same project. In such case, special considerations are needed to keep the work effective and to synchronize the work:

- *Divide the work into smaller entities.* Work should be divided into smaller pieces that have as few interconnections as possible. Usually there is one focus area in the prototype (for example, Settings application) and supporting functionality (Application Grid to start Settings, making calls to test the settings, and so on), thus defining logical boundaries for the division.
- *Keep the focus area solid.* When several persons work with the same prototype, it is recommended that the focus area of the prototype is handled by one developer and the supporting functionality is left to others.
- *Freeze the data model of the prototype.* This ensures that all parties involved in the development use the same data. After a sufficiently detailed data

model is created, it should be frozen; that is, it cannot be modified without common agreement between the developers.

- *Invest time in design.* Enough time should be arranged to design the data model and the division into smaller entities because changes made to the data model need to be propagated to every entity, which is very error prone.
- *Integrate often.* The work should be integrated often to keep the amount of errors to a minimum. For example, weekly integrations are a minimum in a one-month project.

Usually there are quite a lot of symbols in the prototypes that are reusable by all developers in the project. This is especially true with several variants of a prototype, since typically only a small portion of the concept changes between variants. Flash provides a mechanism for creating a shared library of symbols that can be used in several Flash applications. For example, views related to making calls can be the same when testing different concepts of a phonebook. Thus, the view symbols related to making calls can be placed in an external `.fla` file and those symbols can be imported to the prototypes implementing the variants. An `.fla` file can be opened as a shared library from *File > Import > Open External Library*.



**Caution:** When using a symbol from a shared library, the shared symbol must have the *Always update before publishing* feature on in its *Properties* settings. Without setting the feature on, updates to the original symbol are not taken into use. Also, editing must be done only to the original symbol.

### 3.3 Creating a Flash prototype

#### 3.3.1 Implementation issues

##### 3.3.1.1 Data model

UI prototypes usually require generating test (or demo) content for the prototype. To make the data easily modifiable, it is best placed in a file external to the prototype. This solution allows other persons than the prototype developer (for example, a usability designer) to modify the data. In usability tests there are situations where the test data needs to be slightly changed, for example, by correcting typos or changing names. When you are using external data, it is easier to modify the data because the prototype does not need to be recompiled.

In Flash, an easy way to use external data is to put the data into an XML file and read it into the prototype application when the application is started. See Appendix A, “An XML loader class,” for an example of an XML loader which loads contacts data into a Flash application.

In general, arrays are quite handy data structures for collecting data. Arrays store objects, so they are not bound to any single data type. For example:

```
var myArr: Array = new Array;
myArr.push("A string");
myArr.push(new Thong());
myArr.push(42);
trace(myArr); // outputs "A string", [object Object], 42
```

#### Example 1: Simple use of arrays

Example 1 assumes that the class `Thong` is defined.

In addition, arrays themselves are normal objects, and their behavior can be extended with custom methods if needed.

Objects can be used as associative arrays (or map-like data structures), which makes it easy to refer to array indexes with keys in string format. For example:

```
var assocArr: Object = new Object;
assocArr["take"] = "that";
assocArr["squirrel"] = "banana"
trace(assocArr["take"]); // outputs "that"
```

### Example 2: Use of associative arrays

Array traversal in associative arrays is implemented using `for(x in y)` statements, which traverse through the properties of an object:

```
for (var x in assocArr)
{
    trace("assocArr[" + x + "] = " + assocArr[x]);
}
```



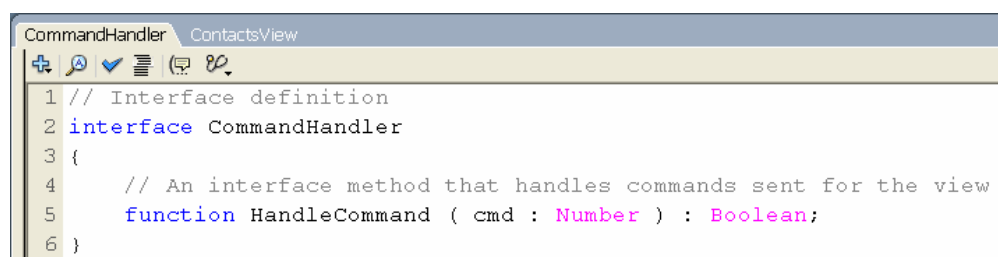
**Tip:** The shared library mechanism discussed in Section 3.2.2, “Multiperson projects,” can also be used to store the common data model of the application in a movie clip symbol and access it from other `.fla` files.

#### 3.3.1.2 Object-oriented programming

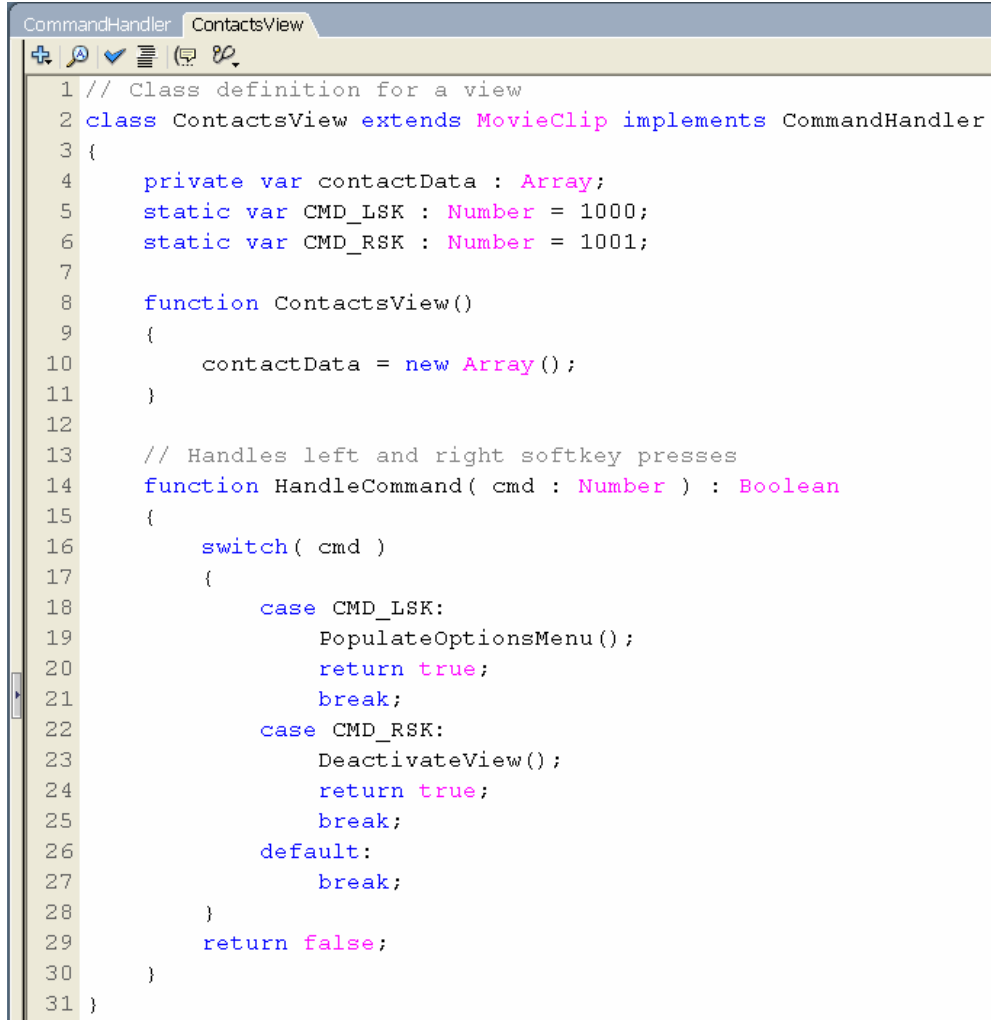
In ActionScript, as in any OO-based programming language, the application code can be placed in a class. It is recommended to use classes instead of placing code inside a movie clip whenever the code could be reused in other movie clips, or the behavior could be extended in other movie clips. This makes it possible to extend or modify the behavior in the “base” movie clip and still make the change automatically visible in the other movie clips.

As in Java, the source file name must match the class name. That is, `SomeClass` goes into a file named `SomeClass.as`. A single `.as` file can contain only one class definition and any number of other statements (for example, global functions). ActionScript uses the same inheritance rules as Java, that is, it does not support multiple inheritance nor does it have abstract classes. In ActionScript it is possible to derive from a single class and implement multiple interfaces.

For examples of using classes, see Figure 3 and Figure 4.



**Figure 3: An interface**



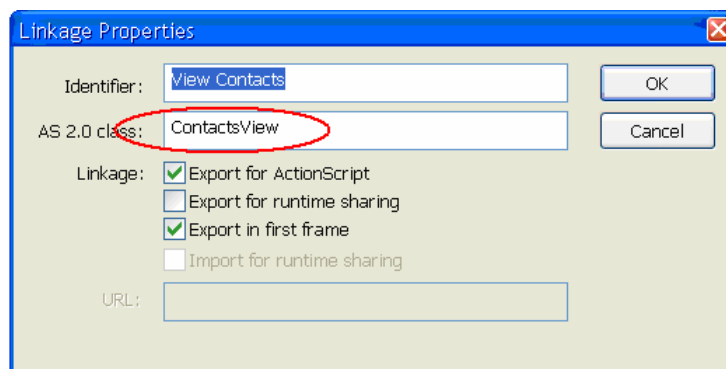
```

1 // Class definition for a view
2 class ContactsView extends MovieClip implements CommandHandler
3 {
4     private var contactData : Array;
5     static var CMD_LSK : Number = 1000;
6     static var CMD_RSK : Number = 1001;
7
8     function ContactsView()
9     {
10         contactData = new Array();
11     }
12
13     // Handles left and right softkey presses
14     function HandleCommand( cmd : Number ) : Boolean
15     {
16         switch( cmd )
17         {
18             case CMD_LSK:
19                 PopulateOptionsMenu();
20                 return true;
21                 break;
22             case CMD_RSK:
23                 DeactivateView();
24                 return true;
25                 break;
26             default:
27                 break;
28         }
29         return false;
30     }
31 }

```

**Figure 4: A class implementing an interface and inheriting from another class**

In the Flash editor, movie clips that use classes need to be explicitly linked to a class (Figure 5). Failing to make the linkage to the class makes it impossible to use the code of the class in the movie clip, or access the code outside the movie clip. When duplicating movie clips, remember to re-set the linkage to the correct class since duplication does not automatically set the linkage parameter.



**Figure 5: Setting linkage of a symbol to an ActionScript class**

### 3.3.2 Debugging methods

Flash IDE has a debugger that can be used in debugging Flash applications. Unfortunately, the debugger is not available before Flash Lite 3.0. For this reason, alternative debugging methods need to be used in Flash Lite 2.x (and Lite 1.1) applications.

Using traces is the simplest (but quite powerful) method to get information about the run-time status of a Flash application. By placing trace information in the code, a good view of the internals of the running application can be obtained. For example, a call history of the application can be created by placing a simple trace statement at the beginning of each method:

```
function MyFunction ( param : String )
{
    trace("MyFunction() called with parameters: " + param);
    // code starts here
}
```

#### Example 3: Use of trace

It is not possible to see traces when running a Flash application in a mobile device because there is no debugging environment available. However, it is possible to set up a simple mechanism that prints out debugging information in a similar manner as traces. The idea is to create a simple movie clip that contains a large text field (even the size of the whole screen). Traces can be simulated by adding different methods to the movie clip which add text to the text field. In the following example it is assumed that there is a multiline `TextField` symbol named `debugText` in a movie clip displaying debug text.

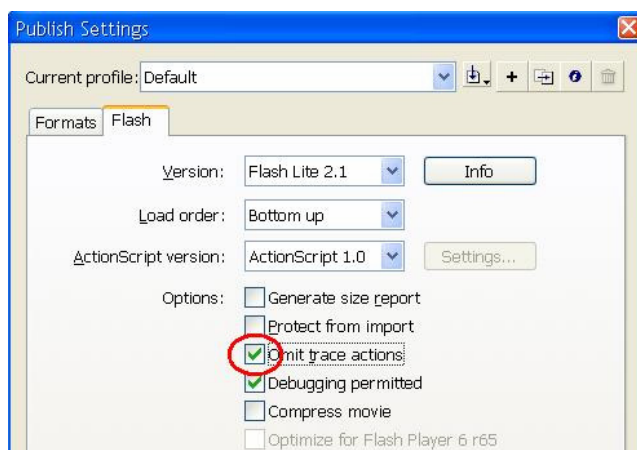
```
function DoTrace(obj)
{
    if (obj instanceof Object)
    {
        debugText.text += "Object:\n";
        for (var x in obj)
        {
            debugText.text += "  " + x + " = " + obj[x] + "\n";
        }
    }
    else
    {
        debugText.text += obj + "\n";
    }
    debugText.scroll = debugText.maxscroll;
}
```

#### Example 4: An example of generating trace information in a Flash Lite enabled device

It should be noted that in the examples above there is no scrolling of text implemented, or any other complex functionality. Also, a real “debugger” should have a mechanism to show and hide the debug window, for example, by pressing a certain key.



**Tip:** When publishing a version of the application which is intended to run on an S60 device, remember to omit traces from the compiled version in the *Publish Settings* (Figure 6). This will save some memory and improve the performance slightly since traces are not included in the code.



**Figure 6: Omitting traces from a published Flash presentation**

### 3.3.3 Testing of prototypes

UI prototypes are prototypes, as the name implies, and they do not need to possess quality that is in industry strength applications. Still, UI prototypes should be formally tested.

Flash UI prototypes are usually created very quickly and the underlying design might not be bullet-proof; thus the application code might become spaghetti-style after a while, and the risk of introducing strange bugs when adding new features increases. UI prototypes, however, have intrinsic quality requirements: prototypes are used in 'real-world' cases (for example, usability tests, product idea show-casing) where their behavior should mimic the real application. Therefore, the application should work reliably with any user interaction or data.

To assure that a sufficient quality level is met, systematic testing of prototypes is needed — similar to the final product but in a smaller extent. It is recommended to use an external testing engineer, if possible. This ensures that the test cases have a good coverage and the tester is not too familiar with the concept (persons involved in the development tend to have a bias to test only happy-day scenarios).

Prototypes should be tested throughout the development process (despite the development cycle being very short). This ensures that anomalies are found early and there are no big surprises at the end of the project. Basically there should be some intermediate releases (at least) weekly, and the releases should be tested independently.

As debugging, testing should be done in the intended target device, too. There are certain differences between the Flash player in a PC and the Flash Lite player of the mobile device. In mobile devices the amount of available memory is limited, which can quite easily cause lack of memory to run applications properly.





**Tip:** By decreasing the size of the heap in a device profile XML file, the lack of memory can be simulated in a PC. See Section 3.1.2, “Tools,” for more information about device profiles

In addition, Flash players behave a little differently in different device models, so there might be some unexpected results in some models even though the same code runs nicely in others.



**Tip:** As a rule of thumb of testing, test early, test often. And test in the target device, too.

### 3.3.4 Small memory optimization

In mobile devices there is a limited amount of memory allocated for the use of the Flash Lite player. Players run in a sandbox that defines the maximum amount of memory for run-time use of the player (typically 1–2 MB). To prevent running out of run-time memory, special actions should be taken during the prototype development.

In Flash the developer does not need and cannot explicitly release the resources s/he has used, since the player handles releasing unused memory for the developer. To indicate that some memory allocations are not needed anymore, the object can be deleted or assigned a `null` value. Note that `delete` does not have an effect in local variables defined with the `var` keyword.

Unfortunately, deleting an object does not trigger garbage collection in the player because the process is run automatically in 60 second intervals, and it cannot be forced. (In fact, garbage collection is also triggered when memory consumption of the application increases suddenly over 20 %.)

To lower the amount of code and data in the run-time memory at a certain time, the application can be divided into several independent `.swf` files. For example, each view of the application can be placed in a separate `.swf` file, and when switching between views, a new `.swf` file is loaded into memory.

```
class ViewChanger extends MovieClip
{
    private var currentView : MovieClip;

    function SwitchView( newView : String)
    {
        this.loadMovie( newView, "currentView");
    }
}
```

#### Example 5: A technique to replace a view with another

In the previous example, a `ViewChanger` class is created which has a method `SwitchView()` that is used to load a new `.swf` containing the view code. By replacing a movie clip with another, the contents of the movie clip are flushed from the memory. This technique ensures that only the necessary amount of data is kept in the memory.

When dividing the code into several `.swf` files the data model needs to be designed so that the shared data (and code) between `.swf` files is accessible at all times. The common code and data can be placed in a “parent” `.swf` that loads the other `.swf` files when needed. For example, the parent `.swf` could have code for menus, general notifications, and loading of external data. (See

Section 3.2.2, “Multiperson projects,” for information about dividing a prototype into several `.fla` files).

Use vector graphics whenever possible because it produces usually smaller file sizes and requires less run-time memory. If you need to use bitmap graphics, try different options for images when storing and test whether different options produce acceptable image quality. For example, for images with large continuous areas with the same color, you could try saving them in `.png` format, or you can try to lower the bitmap color depth (in this case lowering the color depth may not cause any noticeable differences).

If you use bitmap images in the stage, you can try to reduce the output file size by converting the images to vector graphics from the menu *Modify > Bitmap > Trace Bitmap*. Try different options here and test which one produces the best results.

You can test the published file size by activating the *Generate Size Report* option in the publish settings. This produces a listing of the generated `.swf` file with detailed information of the memory consumption of different elements in the file. When running out of run-time memory, start optimizing the code in elements that contribute the most to the size of the compiled code.

## 3.4 Extending the capabilities of Flash

In general, Flash is intended to be a platform- and device-independent environment for executing multimedia applications. For this reason the mechanisms for accessing platform-specific functionality are limited. For example, in Flash Lite only the standard ITU-T keyboard keys are supported and it is not possible to capture S60-specific keys.

Fortunately, there are some mechanisms that can be used to extend the capabilities of the standard Flash player.

### 3.4.1 `fscommand()`

ActionScript has the `fscommand()` and `fscommand2()` commands to interact with the basic functionality of a mobile device (only `fscommand()` is used in this discussion because their behavior is almost similar). These commands are used, for example, to create a phone call, send messages, browse the Web, launch external applications, and read different parameters of the device (for example, battery status, signal strength, available memory). For example, the Contacts application can be launched with the following command:

```
fscommand("Launch", "phonebook.exe"); // 3rd Edition devices
// 2nd Edition devices
fscommand("Launch",
    "z:\\system\\apps\\phonebook\\phonebook.app");
```



**Tip:** It should be noted that in S60 3rd Edition devices the absolute path cannot be used since the binaries are always located in the same folder. In 2nd Edition devices, the absolute path of the application needs to be used.

One problem with `fscommand()` is that from the Flash perspective it may provide access to potentially unsafe operations, for example, making a call without the user knowing this. For this reason the Flash Lite player requests the user a confirmation for the operation that is considered potentially unsafe. In many

cases this is not very user-friendly, since it distracts the user from the normal interaction flow, for example when making a call.

`fscommand()` is available only in the Flash Lite versions of published applications, and they cannot be used in the desktop Flash versions. To alleviate this problem, it is recommended to place all `fscommand()` commands you use in a wrapper class, and create two implementations for the Flash Lite and Flash versions. The following example demonstrates how to use the `FSCommand` class to create two versions of the class for the two platforms.

```
// class implementation for the S60 version
class FSCommand
{
    static function Launch( parameters : String )
    {
        fscommand("Launch", parameters);
    }

    static function GetFreePlayerMemory() : Number
    {
        return fscommand2("GetFreePlayerMemory");
    }
}

// class implementation for the PC version
class FSCommand
{
    static function Launch( parameters : String )
    {
    }

    static function GetFreePlayerMemory() : Number
    {
        return -1;
    }
}
```

#### Example 6: Two implementations for the `FSCommand` class



**Tip:** In the example the `FSCommand` class is simply implemented in two files. In reality, you should consider creating an interface file that defines all required `fscommand()`s.

When compiling to the other platform, the class implementation must be replaced with the associated file. Now, in the code there needs to be only one type of call to `fscommand()`:

```
FSCommand.Launch("phonebook.exe");
var mem : Number = FSCommand.GetFreePlayerMemory();
```

#### 3.4.2 Creating external supporting code

In Flash you can send and receive data from external entities by using sockets. This mechanism is intended to be used in communicating with servers in the Internet, but this mechanism can also be used to send and receive data in a local socket. In solutions utilizing this mechanism, a Flash application requests the

external application to do some platform-specific operation and return the result to the Flash application.

There are three potential mechanisms to use the sockets: the `getUrl` method, the `LoadVars` class, and the `XMLSocket` class.

A good example of a useful external application is a small utility that captures key presses of the S60-specific keys and forwards the information to the use of a Flash application. In the following example, a schematic flow of the interaction between the Flash and external application is described:

1. The Flash application launches the external application that is used for capturing key presses:

```
fscommand("Launch", "helper.exe");
```

2. After launching the external application, the Flash application requests the external utility to capture key presses. In the request it can of course be specified if only certain key presses are requested to be captured.

```
MakeRequest()
{
    // Example request contents:
    // request.params.requestId = "CaptureKeys";
    // request.params.keyCode1 = 63586; // Send key
    // request.params.keyCode2 = 63587; // End key
    var loader : LoadVars = new LoadVars;
    // Populate loader with request parameters
    for (var x in request.params)
    {
        loader[x] = request.params[x];
    }
    loader.requestId = request.params.requestId;
    // Dispatch HTTP request to the helper software
    loader.sendAndLoad("http://127.0.0.1:10000/",
                      this, "GET");
}
```

3. When a key press is captured, a notification about the specific key press is returned to the Flash application. The Flash application interprets the notification and acts based on the received key press. With `loadVars` the response for the `sendAndLoad()` call is received in the `onData` event handler.

```
function onData(str: String)
{
    // Response to an HTTP request received

    var keyEvents: Array = str.split("&", 100);
    for (var k = 0; k < keyEvents.length; k++)
    {
        var key : Object = ParseKeyEvent(keyEvents[k]);
        // Notify an observer about received key events
        _observer.CapturedKeyPressed(key);
    }
    // Issue new request
    MakeRequest();
}
```

4. When the external application is not used anymore, the Flash application requests it to close by sending a specific request to close it. This is done in a similar fashion as in step 2.

The external application can be written with any language that is capable of using sockets. With Symbian C++, all functionality of S60 devices can be accessed, but the development effort is quite high and may become unacceptable in the scope of the prototype development. In Python, it is also possible to access some functionality of the native platform and it might be a more viable solution to create the external functionality for the prototype. A solution for extending Flash Lite features with Python can be found from [8].

### 3.5 Limitations of Flash

Even though Flash Lite is a very powerful tool for creating UI prototypes, it has some limitations that make it an unsuitable or at least a non-optimal tool in some cases.

Code compiled by Flash Lite generates byte-code that is interpreted by the Flash player. While using an interpreter enables running the same prototype in virtually all Flash-enabled devices, it causes some performance problems. Creating prototypes that use an extensive amount of calculations or algorithms causes prototypes to run slowly even in the latest S60 devices. To overcome the problems, the same considerations are needed as in developing normal embedded software. For example, data structures should be kept simple (and small) so that traversing data is quick and no unnecessary calculations are needed. Also, sloppy coding with unnecessary loops needs to be avoided because the complexity becomes easily very high (this is not visible in a PC because there is a huge amount of more processing power than in an S60 device).

Flash does not provide direct support for creating 3D animations. This is a pity, since the current trend in UI development is to create fancy 2D/3D animations which make the UI more attractive to end users. Naturally, it is possible to create an own 3D engine, but running smooth and fluent animations might not be possible due to the processing power limitations. However, the latest S60 devices with graphics accelerators have eased the problem, and Flash players are expected to utilize the graphics accelerators better in the future. Currently the best solution could be to use other methods and tools that are better suited for creating 3D animations (for example, native Symbian C++ and OpenGL ES [9]).

The level of integration to the native platform, as discussed in Section 3.4, “Extending the capabilities of Flash,” can also be seen as a limitation. With a better integration, even more realistic prototypes could be created with less effort, as the existing functionality in the device could be reused. For example, in many prototypes contact information is needed, and it would be a natural choice to use the contact information of the phonebook in the device. Currently the only possibility to use this data is to create an external application that fetches the data from the phonebook and passes it to the Flash prototype.

### 3.6 Summary

Flash Lite is a potential tool for developing UI prototypes. To make the most of the capabilities of the tool, the following issues should be considered:

- *ActionScript skills are needed.* Flash Lite is a development environment whose fundamentals are easy to learn. To fully deploy the power of the tool, and to create complex and dynamic prototypes, ActionScript skills are needed. In essence, this requires experience in object-oriented programming. Since software development is involved, prototypes should be carefully debugged and tested before releasing them.

- *Create reusable software.* Almost all UI prototyping projects have similar functionality. For this reason, it is recommended to create reusable Flash software, that is, component libraries and class frameworks. When creating reusable software, it should be judged how much effort is used, since the achieved benefits may not justify the effort if too much time is spent on the development.
- *Optimization for small memory devices needed.* Creating UI prototypes with Flash Lite for S60 devices usually requires optimization techniques. The amount of memory and processing power of the current devices require another approach to the development than the development in the desktop environment. Also, limitations in the processing power make it hard or impossible to create Flash Lite prototypes of certain type, for example, prototypes using 3D.
- *Use native code to extend the capabilities of Flash.* Flash Lite has limited possibilities when integrating to the native S60 platform. To better utilize the functionality in S60, native Symbian C++ code (or Python, JavaScript, and so on) needs to be used.

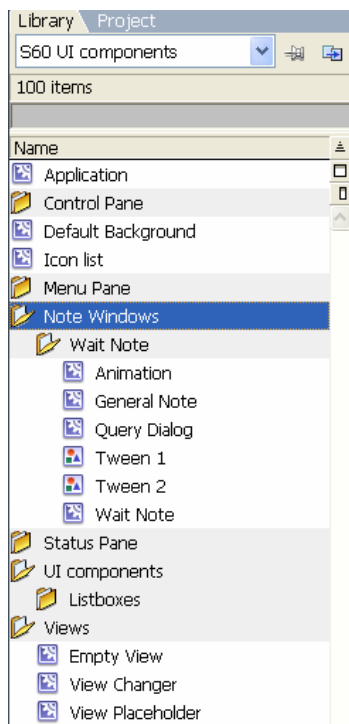
## 4. Case study: UI prototyping at SYSOPENDIGIA

SYSOPENDIGIA's UI design projects are done in multidisciplinary teams consisting of an interaction designer, usability designer, graphics designer, and a prototyping engineer (in larger projects there might several persons from the same profession) [10]. It has been noticed that when conducting UI design projects with participants from several fields, the traditional methods fail to give the optimal solution. For this reason, the Scrum process model [11] is used in most of the projects by SYSOPENDIGIA's user experience group.

The Scrum model emphasizes communication within the team, customer involvement, light management, and avoiding unnecessary documentation. These issues are very important in short UI design projects. The following lists in a nutshell the most important aspects of SYSOPENDIGIA's adaptation of the Scrum model:

- *Project team located in the same premises:* This ensures fluent communication and reduces the need for written documentation.
- *Weekly releases:* The UI design and prototype are released weekly and sent for the customer's approval. A certain set of features is designed and prototyped during a week, and their functionality is also tested systematically.
- *Prioritization of work:* Using a spreadsheet-based plan, the progress of the project can be easily tracked and work for the next weeks planned. In weekly planning meetings the remaining work can be prioritized and unimportant features left to the end of the project. This ensures that the most important features are implemented first and enough time is reserved for them. In the planning meetings the customer gives feedback about the design and can suggest improvement ideas.

A semi-finished UI component library is used in UI prototyping. The component library has UI components that can be reused in prototyping projects, but which need some modifications, for example changing their dimensions or colors. The UI component library is a collection of movie clips representing UI objects ranging from basic views to radio buttons (Figure 7). The component library has a supporting UI framework class library implemented in ActionScript.



**Figure 7: Example screen shot from a Flash UI component library (courtesy of SYSOPENDIGIA)**

The UI component library is also suitable for other platforms than the S60 platform. For example, the UI component library has been successfully used in simulating touch UI concepts for platforms that do not yet have existing hardware. In such cases, the simulation has been done in an existing device with a touch display.

Since it is not possible to capture S60 keys in Flash Lite, SYSOPENDIGIA has written a key capture server to the native S60 platform. It has been implemented with Symbian C++, and it is capable of capturing any set of keys a Flash application requests. The key capture server and the supporting functionality in the UI framework library is an essential tool when developing UI prototypes to S60 devices, since it enables capturing, for example, Application and End keys.

## 4.1 Summary

UI prototyping has been successfully applied in several customer projects with the methods discussed in this chapter. Quality of the designs has improved since UI prototypes can be tested on real S60 devices, which produces more accurate usability test results. Also, with prototyping the needs of end users are easier to incorporate to the UI design.

Above all, customers have been very pleased to see working UI prototypes early on in the project. Customers have also been able to participate better in the design process by giving comments and new design ideas.



---

## 5. Summary

UI prototyping projects are done quickly, and the UI design evolves at the same time as the prototype is being developed. For this reason, it must be possible to create visible results quickly with a prototyping tool and making changes must be easy.

Flash Lite has great potential as a tool for developing UI prototypes for S60 devices. With the tool, complex and visually appealing prototypes can be created with a relatively small effort.

It is possible to create applications without programming skills with Flash Lite. However, to make interactive and dynamic content to prototypes, ActionScript skills are needed.

With the programming-oriented nature of Flash, software development paradigms should be used when developing UI prototypes. Reusable UI components should be created to speed up the development. In the S60 environment, optimization techniques are needed to make the prototype run smoothly in the mobile device.

Flash Lite is not suitable for all UI prototypes. For example, due to the processing power limitations, making 3D animations is hard (or impossible) with current Flash versions and mobile devices. However, the capabilities of Flash Lite can be extended by writing external code which can utilize the capabilities of the native platform better.

## 6. Terms and abbreviations

Term or abbreviation	Meaning
API	Application programming interface
CDK	Content development kit
FLA	File format for Flash application source code
FUI	Functional and user interface
IDE	Integrated development environment
ITU-T	International Telecommunications Union, Telecommunication Standardization Sector
MB	Megabyte
OpenGL ES	Open Graphics Library for Embedded Systems, a subset of OpenGL 3D graphics API designer especially for embedded systems
PC	Personal computer
PNG	Portable network graphics
S60	A smartphone platform from Nokia
SDK	Software development kit
SVG	Scalable vector graphics
SWF	File format for compiled Flash applications
Tweening	A technique to create animations
UI	User interface
XML	Extensible markup language

**Table 1: Terms and abbreviations**

---

## 7. References

- [1] Adobe Flash  
<http://www.adobe.com/products/flash/>
- [2] Python for S60  
<http://opensource.nokia.com/projects/pythonfors60/>
- [3] Adobe Flash Lite  
<http://www.adobe.com/products/flashlite/>
- [4] Adobe press release about Flash Lite  
<http://www.adobe.com/aboutadobe/pressroom/pressreleases/200702/021207FlashVideo.html>
- [5] Flash Lite 2 Content Development Kit  
[http://www.adobe.com/devnet/devices/development\\_kits.html#flashlite](http://www.adobe.com/devnet/devices/development_kits.html#flashlite)
- [6] Adobe Creative Suite  
<http://www.adobe.com/products/creativesuite/design/>
- [7] S60 Platform SDKs  
<http://www.forum.nokia.com/info/sw.nokia.com/id/4a7149a5-95a5-4726-913a-3c6f21eb65a5/S60-SDK-0616-3.0-mr.html>
- [8] Python framework for Flash Lite development  
<http://www.felipeandrade.org/flyer/>
- [9] OpenGL ES  
<http://www.khronos.org/opengles/>
- [10] User Experience by SYSOPENDIGIA  
<http://www.sysopendigia.com/C2256FEF0043E9C1/0/405001108>
- [11] Scrum Alliance  
<http://www.scrumalliance.org/>

## Appendix A. An XML loader class

```

class ContactsLoader extends XML
{
    private var _callback: Function;

    function LoadContacts(f: String, cb: Function)
    {
        // Callback function to call after contacts have been loaded
        _callback = cb;
        ignoreWhite = true;
        // Ignore whitespace when parsing the content
        load(f); // Load the file
    }

    function onLoad(b: Boolean)
    {
        var items: Array = [];
        // XML Loaded?
        if (b) {
            items = DoParse(firstChild); // Yes, parse.
        }
        // Notify about completion (success or failure)
        _callback(b, items);
    }

    /**
    Parse contacts list

    <contacts>
        <contact>
            ...
        </contact>
        <contact>
            ...
        </contact>
    </contacts>
    */

    function DoParse(node: XMLNode)
    {
        var items = new Array;

        // The root node is "contacts"
        if ("contacts" == node.nodeName) {
            for (node = node.firstChild; node != null;
                node = node.nextSibling)
            {
                // Found "contact" node?
                if ("contact" == node.nodeName)
                {
                    // Parse contact node
                    var contact = ParseContactNode(node);
                    items.push(contact); // Append it to contacts list
                }
            }
        }
    }
}

```

```

        }
    }
    return items;
}

/**
Parse a contact node

<contact>
  <name>Firstname Lastname</name>
  <number>0123456789</number>
</contact>
*/

function ParseContactNode(node: XMLNode): Object
{
    var name: String = "";
    var number: String = "";

    // Go through child nodes and collect the
    // data we're interested in
    for (var child: XMLNode = node.firstChild; child != null;
        child = child.nextSibling)
    {
        switch (child.nodeName)
        {
            case "name":
            {
                name = child.firstChild.nodeValue;
                break;
            }
            case "number":
            {
                number = child.firstChild.nodeValue;
                break;
            }
        }
    }
    // Construct new contact entry from
    // previously collected details
    return new ContactEntry(name, number);
}

/**
 * Contact entry
 */
class ContactEntry
{
    private var _name: String;
    private var _number: String;

    function ContactEntry(name: String, number: String)
    {
        _name = name;
        _number = number;
    }
}

```

```
function get name(): String
{
    return _name;
}

// other getters & setters omitted
}
```

---

## Evaluate this resource

Please spare a moment to help us improve documentation quality and recognize the resources you find most valuable, by [rating this resource](#).

---

## Poster page

### Utilization of prototypes:

- UI prototypes are used in usability tests to verify the usability of a UI design.
- With a highly visual UI prototype, design ideas are easier to communicate with the customer.
- A UI prototype can be used to raise more funding for a product concept.
- UI prototypes can be used to verify the feasibility of a certain technical issue in an application.

### Archetype of a Flash prototype developer:

- Good object-oriented programming skills.
- Basic skills in using graphics tools.
- Insight into UI design principles.

### Development for reuse:

- It is recommended to create a Flash UI component library.
- Making a component library with movie clips is easier.
- Usually movie clips require modification between projects.
- Reusable Flash UI components are harder to design.
- Real Flash UI components are easy to use, and cannot be modified.
- A framework class library leverages the effort to create basic features of a prototype (changing between views, using timers, managing key presses, and so on).

### Multiperson projects:

- Divide the work into smaller entities.
- Keep the focus area solid.
- Freeze the data model of the prototype.
- Invest time in design.
- Integrate often.
- Consider using shared libraries.



Implementation issues:

- Arrays are simple and easy-to-use data types.
- Standard arrays can be used as associative arrays.
- Use classes to promote reuse: interfaces, inheritance, and so on.
- Movie clips must be explicitly linked to classes.
- Remember to consider the small memory footprint of a mobile device.
- Delete resources you are not using to free the memory for garbage collection.
- Split the prototype into several `.swf` files.
- Use vector graphics when possible.
- Test the prototypes often. Test in an S60 device.
- Create native code to extend the capabilities of Flash.