Verizon Channel Developer Guide

Verizon Channel Developer Guide

Access to and use of this document and all information and data contained herein is subject to the Website Use Terms and Conditions which can be found at:

http://www.verizonwireless.com/b2c/globalText?textName=WEBSITE_USE&jspName=footer/webuse.jsp

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA.

Contents

Chapter 1: Verizon Wireless Dashboard 1
Dashboard service overview 1
Customer interface 2
Dashboard channels 2
Basic channels 2
Premium channels 3
Channel subscription limit 3

Chapter 2: Development Overview 5
Flash Lite and Flash Cast 5
Developing channels 7
Development team 7
Development process overview 8
Development process 9
Development tools 11
Design considerations 11
Development framework 12
File structure 13
Flash Lite FLA file 14
Adobe Mobile AppBuilder 15
Using Adobe Mobile AppBuilder 15
Flash Cast device sets in Adobe Mobile AppBuilder 15
Exporting and Sharing a Project 16
Understanding the contents of an exported project 16
Understanding the contents of a shared project 17
Using FLA files as resource files 17
Troubleshooting Adobe Mobile AppBuilder 17
Single-SWF testing 18

Chapter 3: Setting Up 21
Hardware requirements 21
Software requirements 21
Setup instructions 22
Installing Adobe Flash CS3 Professional 22
Installing the Adobe Mobile AppBuilder 22
Installing Flash Cast fonts for the Flash development environment 23
Setting up the Adobe Mobile AppBuilder extension for Adobe Flash CS3 Professional 24
Testing the installation in Adobe Mobile AppBuilder 24

Chapter 4: Getting Started 27
When to create Flash Cast channels 27
Flash Cast user interface 28
About the Verizon Wireless Dashboard 28
Subscribing to channels 29
Flash Cast system 29
Flash Cast client 30
Flash Cast client software 30
Supported features of the Flash Cast client 31
Flash Cast 1.2 server system 33
Managing data in Flash Cast 34
External data 34
Retrieving and aggregating external data 34
Creating channel data sources 34
Requesting data to the mobile handset 37
Exploring the Dashboard and navigating a channel 38
Define the channel publish settings 38
General channel testing process 40
Single-profile testing 40
Testing locally with Adobe Mobile AppBuilder 41
Testing on a Mobile Handset 42
Managing a project's variations 42
Create a new variation 42
Edit a variation's resource files and target devices 42
About variation preferences name-value pairs 43
Add variation preferences 45
Organize variation preferences 45
Delete variation preferences 45
Delete a variation 45
Sort variations 46
Exporting a project for deployment 46

Chapter 5: Typical Development Procedures 49
Device profile definition 49
Device profile file contents 49
Configuring the channel 50
Channel displays 51
Stacks and views 51
Create views 52
Define the channel profile 53
Error view 53
Defining channel stacks 54
Defining stack views 54
Defining the channel data 54
Creating the content stacks 54
Switching stacks, views, and elements 55
Adding ActionScript to the views 55
Initializing channel views 56
Controlling the speed of animation 57
Publishing using multiple devices 58
Testing with Multiple Devices 58
Multiple variations 58
Scaling Images For Best Fit 59
Loading Device Specific Data 60

Chapter 6: Targeted Development Tasks 61
Linkaway to other content 61
Types of linkaways 61
Linkaway data 62
ActionScript API 62
FSCommand (“Launch”) 62
ActionScript Call 62
ActionScript API examples 63
Launch pre-installed BREW application 63
Launch a non-resident BREW application 63
Launch mobile shop in alternate modes 63
Open a web page 64
Get user input in Dashboard 64
Command usage and parameters 65
EditText command examples 67
One input text field 67
Two input text fields 67
Three input text fields, one with password masking 67
One multi-line input text field with input method set to "word" by default 67
Channel data options 67
Loading feed data 68

About feed records 69
Load a list of feed records 70
Load an image or SWF file 70
Load feed item identifiers 71
Reviewing pseudo-arrays 71
Storing reusable methods centrally 71
Scrolling text 72
Reviewing text field properties 72
Reviewing looping 72
Loading binary data 73
Adapting a channel for user preferences 73
Understanding Preferences 74
Defining default preferences 75
Querying feed data for default preferences 75
Assigning default preferences using Mobile AppBuilder 76
Loading preferences 76
Saving User Preferences 77
Applying saving button functionality 77
Processing user preferences in a batch 77
Data updates 78
Check for data updates 78
Determine when an update last occurred 79

Chapter 7: Key Events and Commands 81
Navigation between views 81
Creating button functionality 82
Supported ActionScript key and button events 83
Creating visible/invisible elements 84
Handling key press events 84
Framework key handlers 85
Using the framework back functionality 86
Managing soft key state 86
FS-command: SetSoftKeyState 87
Soft key behavior 88
Example use case 90
Send clear key event in Dashboard 91
FS-command: SendCLRKeyEventToSWF 92
Examples 93
Show animation 93

Chapter 8: Structure & Security Considerations 95
About the feed item collection format 95
Reviewing XML Syntax 95
Understanding the Flash Cast XML nodes 96
Feedstore space limitations 98

Channel budget....98
Channel limits....99
Reviewing best practices for mobile optimization....100
Reviewing additional resources....101
Content structure....101
The XML descriptor file (db.xml)....101
Sample <feed> element....102
Sample <feedapp> element....102
Sample <datasource> element....103
Sandbox considerations....104

Chapter 9: Flash Cast Embedded Fonts....105
Device fonts....105
Embedded fonts....105
Flash Cast embedded fonts....106
Changing fonts....107

Developer Submittal Guidelines....109
Channel concept submission....109
Channel quality expectations....109
Channel content elements....110
Developer submittal....111

Dashboard Channel Submission....113
Instructions....113
[Channel Name]....113
[Date]....113
Channel Description....113
Contact Information....114
Revision History....115
Device Support....115
Channel Assets....116
Data Usage....117
Platform Dependencies....118
Advertising....118
Security Controls....119
Descriptive Walkthrough....121
Data Entry....121
Sounds....122
Telecommunication....122
Error Messages....123

Index....125

Verizon Wireless Dashboard 1

This chapter provides an introduction to the Verizon Wireless Dashboard, including:

- "Dashboard service overview"
- "Dashboard channels"

Dashboard service overview

The Dashboard service is a flash-based, graphically rich front-end that creates a system for consuming and discovering multimedia content and accessing wireless data applications. The Dashboard service provides a world class navigation experience for Verizon Wireless multimedia data services and applications. The Dashboard service consists of three major components:

- A content application server
- A Flash-based handset client
- A collection of rich multimedia Flash "channels"

These channels are updated by providing content deltas to the device from the content server over the EV-DO network. Each channel is able to render Flash content, launch an application, or invoke other media handlers on the device to process the selected content. As such, the Dashboard application is capable of launching or downloading a native BREW application on the handset in response to a user request from within a content channel. The range of native BREW applications available on a mobile handset includes WAP 2.0 browser, Verizon Tones application, Mobile Shop, video catalog/player, music catalog/player, search application etc. This allows Verizon Wireless to create a simple and unified experience across a range of consumer data services.

Customer interface

Now Playing Screen

The **Now Playing** screen serves as the Dashboard service's home screen. The screen displays a scrollable list of channels to which the user is subscribed

Dashboard channels

The Dashboard service provides two types of channels:

- **Basic** - Basic channels are included in the cost of the service. There is no additional fee for basic channels although megabyte charges may apply. When the user first launches the Dashboard service, the user is automatically subscribed to a group of "default" basic channels. For more information, see "Basic channels".
- **Premium** - Premium channels, content and services require additional fees. Within each of the channels (as applicable), users are able to personalize the content delivered to their device. For more information, see "Premium channels".

Basic channels

Within the set of basic channels, there is a group classified as "default." Default channels are those to which a new user is automatically subscribed when they sign-up for the service. A subscriber can always replace a default basic channel with another "non-default" basic channel.

Premium channels

Premium channels may require an extra recurring payment (for example, on a monthly basis) that the user will have to agree to. A purchase agreement is presented to the subscriber each time they attempt to subscribe to a premium channel.

Channel subscription limit

Subscribers can subscribe to a maximum of ten (10) basic channels, and a total of 15 channels. If a user attempts to subscribe to a basic channel that would put them over the 10 channel limit, the **Replace Channels** dialog is displayed, which lets the user replace an existing basic channel subscription with the new one. Likewise, if a user attempts to subscribe to a channel (either basic or premium) that would put them over the 15 channel limit, they are presented a dialog that lets them replace an existing channel with the new one. More information on the **Replace Channel** dialog is contained in the Dashboard system user interface section. A user can always replace a basic channel with a premium channel—for example, they can subscribe to 15 premium channels and no basic channels, if they like. But they can only subscribe to 10 basic channels at any one time. The set of basic default channels is determined by the default channel line-up template on the Flash Cast server.

Development Overview 2

This chapter provides an overview of the Verizon Flash Cast application development process, including:

- "Flash Lite and Flash Cast"
- "Developing channels"
- "Development framework"
- "Adobe Mobile AppBuilder"

The combination of the custom environment elements, Adobe provided utilities, and the standard framework significantly reduces the effort involved in creating, deploying, and managing channels.

A Flash Cast channel consists of the following combination of components:

- The data that feeds the channel.
- The SWF that displays the data.
- The JPG, SWF or GIF file used for the channel icon, which represents the channel in the main Flash Cast client user interface. (Of these file formats, only SWF files support transparency.)
- Settings and scratchpad data and any assets needed by the channel (content-specific backgrounds, dynamic images, etc.

All of these components are loaded via the Flash Cast client and stored on the user's device in the Feedstore. When a Flash Cast channel loads data of any type, the data is always loaded from the Feedstore rather than directly over the network. This approach allows channels to operate seamlessly even when the user's device has lost its connection.

Flash Lite and Flash Cast

Flash Lite and Flash Cast are related Adobe technologies used for mobile phone applications. Certain types of applications, like weather feeds, can be successfully implemented using either technology but each has a specific purpose.

The basic distinctions of Flash Lite and Flash Cast includes:

- **Flash Lite** is used for mobile user interfaces.
  - Standalone applications like games.
  - Browser-based applications.
  - Handset wallpaper.
  - Handset user interface and menus.
- **Flash Cast** is a client-server technology that aggregates data on the server and sends custom feeds to each mobile handset on a scheduled basis.

The Flash Cast client is software that resides on the handset and contains:

- Flash Lite to render the user interface.
- Data received from the server for online or offline viewing.
- Special programming APIs to expose the data to Flash Lite.

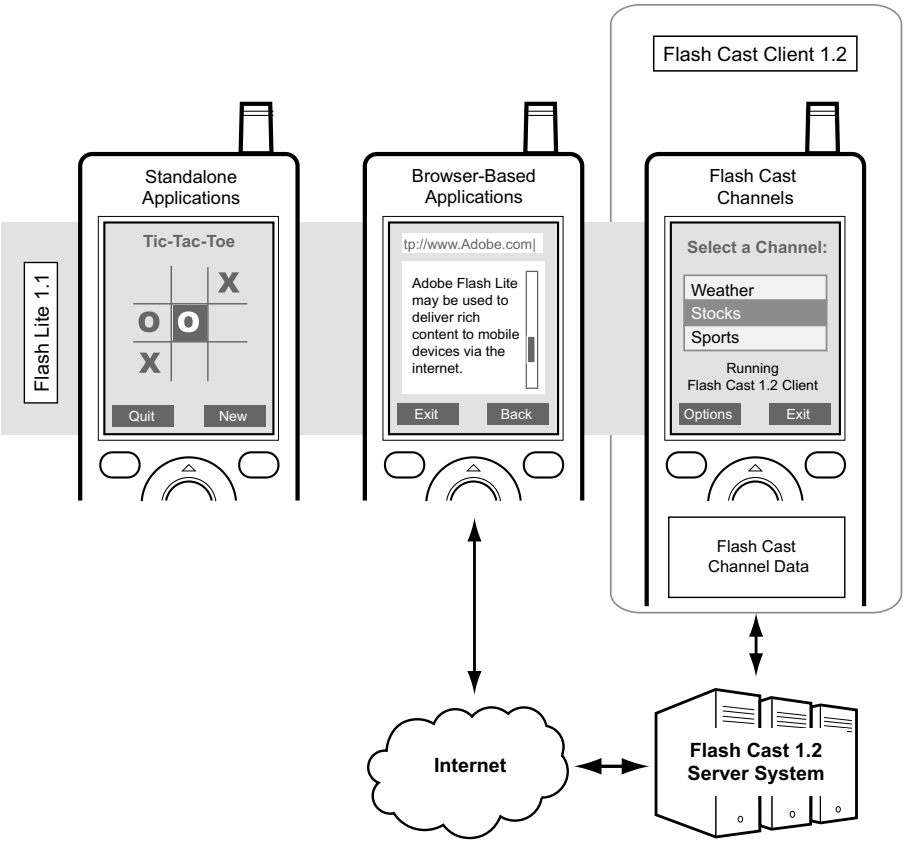
The Flash Cast server system is a cluster of servers that manages:

- The aggregation of data from multiple data sources.
- User subscriptions and data preferences.
- The distribution of custom data (as specified by each user's preferences) to each user's handset.

Flash Lite is a light version of the Flash Player that is specifically designed for mobile devices.

- Flash Lite is a versatile user-interface tool that can be used for self-contained programs or to send/receive information from servers.

Flash Lite is only one element in the Flash Cast client-server tech



Developing channels

Development team

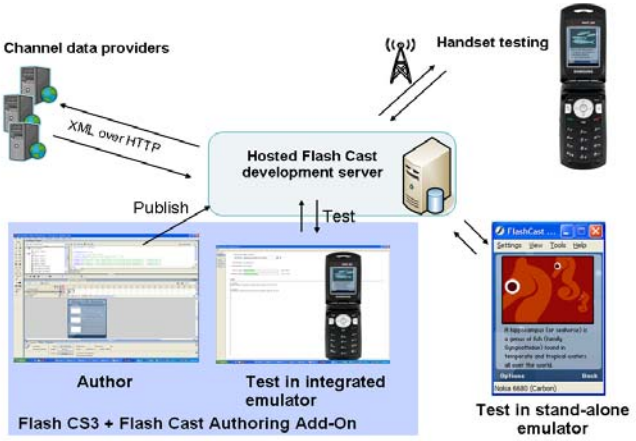
A typical channel development team consists of the following roles:

- Project Manager
- Designer
- Channel SWF developer
- If needed, an XSL expert to work on custom feed XSLT
- Quality Assurance (QA)

Development process overview

At a summary level, the development steps include:

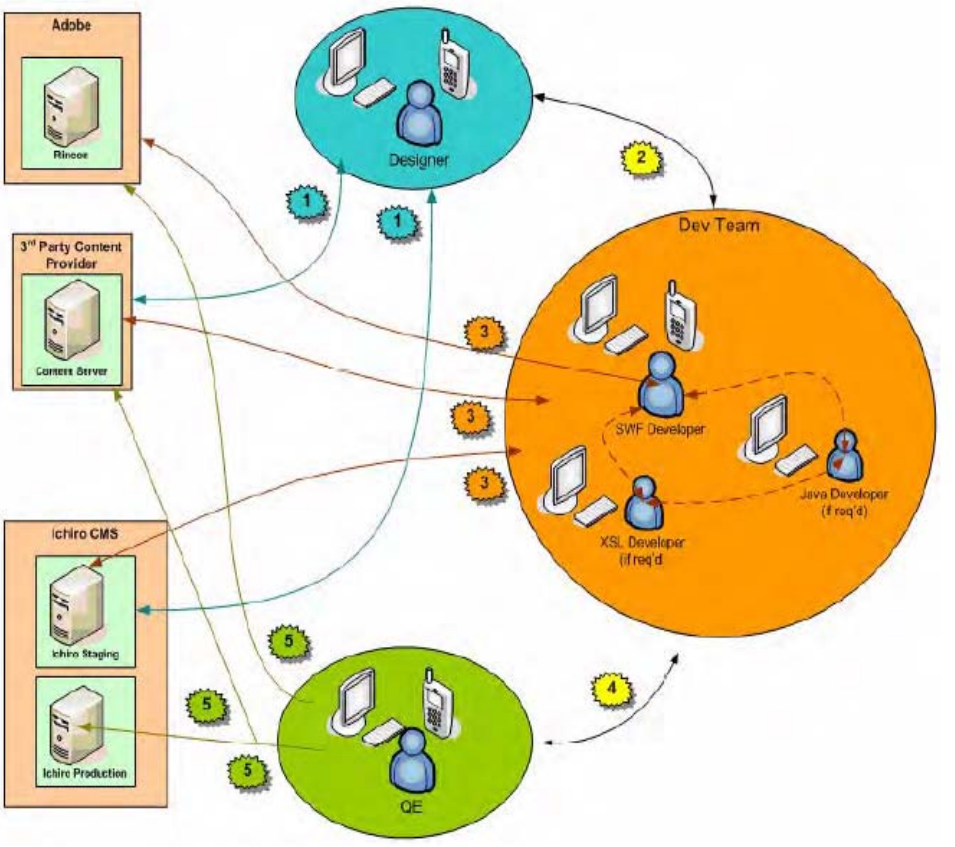
1. Submit for and receive project approval from Verizon Wireless.
2. Design and develop application.
3. Test during development, using integrated and standalone emulators.
4. Deliver completed application to Verizon Wireless for assessment and acceptance.



Channel development and testing process

Development process

The following diagram illustrates the channel development process. The steps in the diagram are explained below.



Channel development process

After getting approval on the channel concept from Verizon Wireless, channel development includes the following procedures:

1. The designer performs the following tasks:
    a. Works with the content provider and with Verizon Wireless to design a Flash Cast channel that makes effective use of the content.

   b. Creates a channel style guide based on the Verizon Wireless Style Guide for Channel Development (provided by Verizon Wireless upon acceptance of the channel proposal).
2. The Designer shares the designs with the development team, makes adjustments as required and hands off designs.
3. The development team accepts the designs from the designer and performs the following tasks:
   a. Using the information provided by Verizon Wireless upon acceptance of the channel proposal, creates the feed, feedapp and datasource entities.
   b. Works with content provider to determine the XSL requirements for the channel, if any.
   c. Creates a behavioral spec for the channel SWF.
   d. Works with Quality Engineers to create a test plan for the channel.
   e. Requests any needed Ichiro configuration changes, if required.
   f. Develops all code for the channel using the Adobe Mobile AppBuilder Flash Cast server (provided by Adobe). During development, its very important to test periodically on a device to confirm the channel code is performing on the phone, not just on the desktop.
   g. If required, creates XSLT files to convert XML from one format to another.
   h. Integrates data from 3rd party content provider and Ichiro staging server.
4. Upon completion of development, the dev team begins working more closely with the QA team to find and fix bugs.
5. The QA Team begins testing of the channel using the Adobe Mobile AppBuilder Server, 3rd party content provider and Ichiro staging server. During this process the QA team performs the following tasks:
   a. Tests designs on all devices required for deployment. This is very important, as designs often look quite different on a device then they did on the desktop. Also, designs will often look different from device to device, as the screen on one phone might accentuate a different set of colors than that of a different phone.
   b. Using test plans developed previously, finds sections of code that are not functioning correctly or are not supporting all of the required use cases.
   c. Logs errors in a bug tracking tool.
   d. Works with Flash or content developers to prioritize logged bugs.
   e. Confirms fixes submitted by developers.
   f. Performs regressing testing on previously fixed and confirmed bugs.

Development tools

To develop Flash Cast channels, developers need:

- **Adobe Flash 8 Professional or Flash CS3 Professional**
- **Flash Cast Authoring extension for Flash CS3 —**This provides a channel testing environment, including a Flash Cast client emulator that connects to a Flash Cast test server (see next item), as well as a channel packaging feature to facilitate deployment to an operator's staging server.\
- **Account on Flash Cast testing server** (**"Adobe Mobile AppBuilder server"**)**—**In order to test channels within Flash CS3 using the client emulator, developers need an account on the Flash Cast test server that's hosted and managed by Adobe.

The Adobe Mobile AppBuilder server is a standard Flash Cast server installation that exposes additional web service APIs to enable communication with the Mobile AppBuilder authoring plug-in. Mobile AppBuilder offers two different workflows to publish the channel content to the server. The developer can either upload his/her content for quick testing to a testing server, or he/she can have Mobile AppBuilder create a "channel package" to deploy the channel to a staging or production server (for example, within the operator's staging network).

Design considerations

Before you start to create a Flash Cast application, consider the following design issues:

- **Device and client specifications** - The Flash Cast client is supported by many different mobile phones and devices, each of which can have different screen sizes, color depths and resolutions, and user interfaces. For example, some devices support pen or stylus, while others might have only a key interface. Also, different types of devices can support different versions of the Flash Cast client (for instance, high-end consumer mobile phones might have features that lower-end mobile phones don't support). The content or logic of your channel may depend on the supported device type or types.

  The Flash Cast server can associate one or more applications (SWF files) with a single Flash Cast channel. The server automatically delivers the appropriate SWF file for each device.
- **Navigation model** - Flash Cast applications run in the context of the Flash Cast client. The client reserves the left and right navigation keys on the mobile phone's keypad to let users navigate between content categories in the Channel Manager, as well as the mobile phone's left and right soft keys. Most navigation in a channel is performed with the up, down, and select navigation keys.

- **Client storage and runtime memory limits** - The Flash Cast client persistently stores each Flash Cast channel, and its associated feed data, on the user's mobile phone or device. The client allocates a fixed amount of *disk space* to each Flash Cast channel. The allocation amount can vary per device. The combined sizes of the application SWF file, channel icon (JPEG file, GIF or SWF file), feed data, user preference data, and any scratchpad or settings data must be less than or equal to this amount. The client also imposes a *runtime memory limit* on each channel, which, like the disk size limit, varies per device. Chapter 8, Structure & Security Considerations covers this in more detail.
- **Authoring content for Flash Lite** - To render Flash Cast applications, the Flash Cast client uses Flash Lite 1.1, a version of Flash Player designed for mobile phones and devices. In general, the same authoring guidelines that apply to a standard Flash Lite application also apply to Flash Cast channels.
- **Channel data design** - A Flash Cast channel consists of a SWF file (the channel application) and the channel data feed.

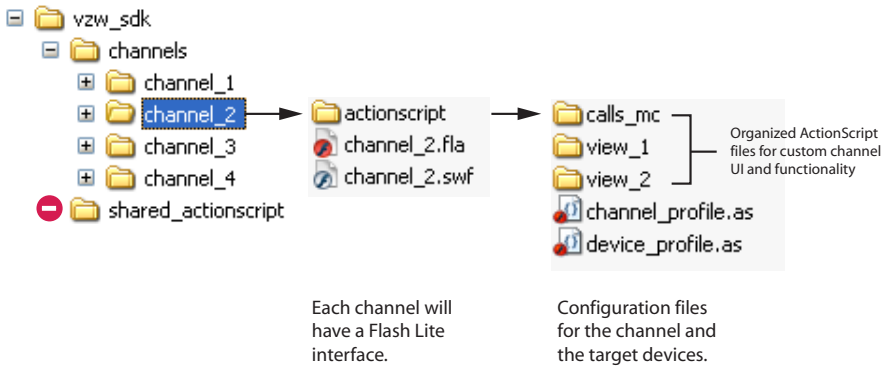
Development framework

The Flash Cast Channel Development Framework provides developers with common tools and a structure for creating channels. It also handles navigation between screens and elements within screens.

File structure

The programming framework is logically organized to separate the core framework files from the channel-specific files, as shown in the following figure.

Flash Cast Channel Development Framework Files



DO NOT modify core framework files unless you are extending the functionality. Flash 4 constrains the extensibility of the framework so proceed at your own risk.

Flash Cast Channel Development Framework file structure

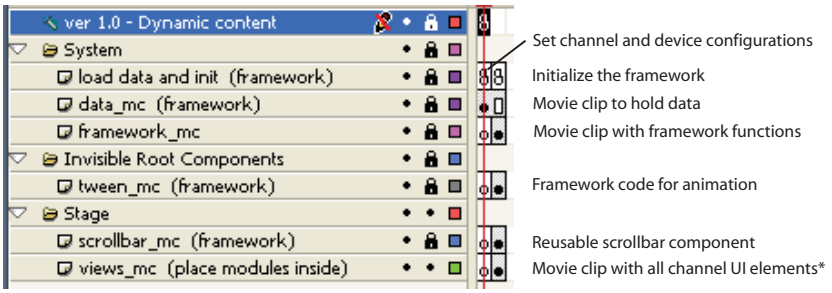
The framework file structure includes:

- A `shared_actionscript` directory with the core framework files.
- A `channels` directory with one subdirectory for each channel you create.
- Each channel folder contains:
  - Flash Lite files for the channel interface.
  - Device and channel configuration ActionScript files.
  - Additional subdirectories of the `actionscript` directory with custom ActionScript files for channel-specific functionality.

Flash Lite FLA file

Each Flash Lite FLA file must have some core framework component as shown in the following figure.

Channel Framework Flash Elements



* For the most part,"views_mc" is the only file that will be manipulated by the developer

The Flash components of the Flash Cast Channel Development Framework

The framework FLA file includes:

- A guide layer with the channel name.
- A frame that calls the device and channel configuration ActionScript files.
- A frame that initializes the framework files in the `shared_actionscript` folder.
- A movie clip, `data_mc`, that will hold the loaded feed data.
- A movie clip, `framework_mc`, that contains framework functions.
- A movie clip, `tween_mc`, that handles framework animation.
- A movie clip, `scrollbar_mc`, that is a reusable scrollbar component.
- A movie clip, `views_mc`, that contains all the user interface elements for the channel.

No framework elements, except `views_mc`, should be modified unless you are extending the framework functionality.

Adobe Mobile AppBuilder

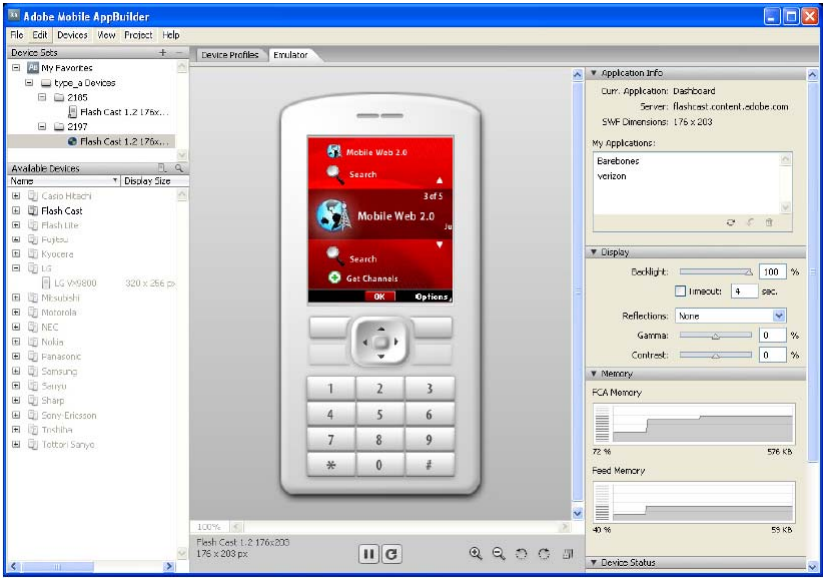
Using Adobe Mobile AppBuilder

Adobe Mobile AppBuilder lets you test your content as it will appear on the desired handset. You can switch to different handsets during a single session to test multiple devices at once. This makes developing for carrier-specific handsets very easy to manage.

The test view contains lists of available devices for your content type on the left side of the screen with device specific information and test parameters on the right.

Using Adobe Mobile AppBuilder you can:

- Publish to the test server
- Publish a SWF file locally
- View device information for multiple handsets



Adobe Mobile AppBuilder

Flash Cast device sets in Adobe Mobile AppBuilder

Adobe Mobile AppBuilder is created so the group of mobile devices that is being tested can be changed quickly and easily. In Mobile AppBuilder the current device sets being used are displayed in the **Device Sets** panel on the left side of the screen.

The information defining device sets is contained in the `solution.xml` file located in the `c:\Users\your user name\AppData\Local\Adobe\Adobe Mobile AppBuilder` directory on your machine.

If a developer wishes to change the device set that is being tested with Mobile AppBuilder, they must simply overwrite the current `solution.xml` file with one containing the desired devices.

When Mobile AppBuilder is next opened, the new device set will be displayed in the **Device Sets** panel. The default `solution.xml` file for the Brew operating system is available for download on the Adobe prerelease website under the name Verizon client section.

- The solutions.xml file is the central location of Device Sets that should be used by developers to determine what Dashboard devices need to be supported by a channel
- All new Dashboard devices will be added to this list when the device information is ready for public release so developers can immediately begin development/optimization for their channel(s) for that device

Exporting and Sharing a Project

In this section you will be introduced to Adobe Mobile AppBuilders ability to share or export a project and the difference between the two.

Understanding the contents of an exported project

Exporting a project with Adobe Mobile AppBuilder creates a compressed file containing the following:

- `db.xml` - Defines the properties of the channel, including:
  - A globally unique identifier (GUID) defined as the `feed id` property.

    NOTE The GUID is created for the channel the first time it is published and is the key value for connecting the channel properties.

  - The channel name.
  - The channel target device.
  - The location and name of the channel icon file.
  - The location and name of the channel SWF file.
  - The location and name of the channel default preferences.
  - The query path and other properties of the channel data source.
- `default_app.swf` - The default feed application SWF file.
- `default_icon.jpg` - The default channel icon if one was defined.

Understanding the contents of a shared project

Creating a shared project with Mobile AppBuilder creates a compressed file containing the following:

- `default_app.mabp` - The application created in Adobe Mobile AppBuilder that is used to view that specific channel.
- `Resource Files` folder - This folder contains the files used for that channel application. These files include:
  - `default_app.fla` - The default Flash file used for development.
  - `default_app.swf` - The default feed application SWF file.
  - `default_icon.jpg` - The default channel icon if one was defined. Channel icons can be JPG, GIF or SWF files although only SWF files support transparency.

NOTE This ZIP file can be sent to other developers who will have everything they need to run the channel in Adobe Mobile AppBuilder upon receiving the file.

Using FLA files as resource files

It is a best practice, while using Adobe Mobile AppBuilder, to include the current channel's FLA file as a **Resource File** in the project. Once the FLA file is included as a Resource file, the file can be opened in Flash CS3 by double-clicking it. The FLA will also be included in the ZIP file if the project is shared and sent to another developer.

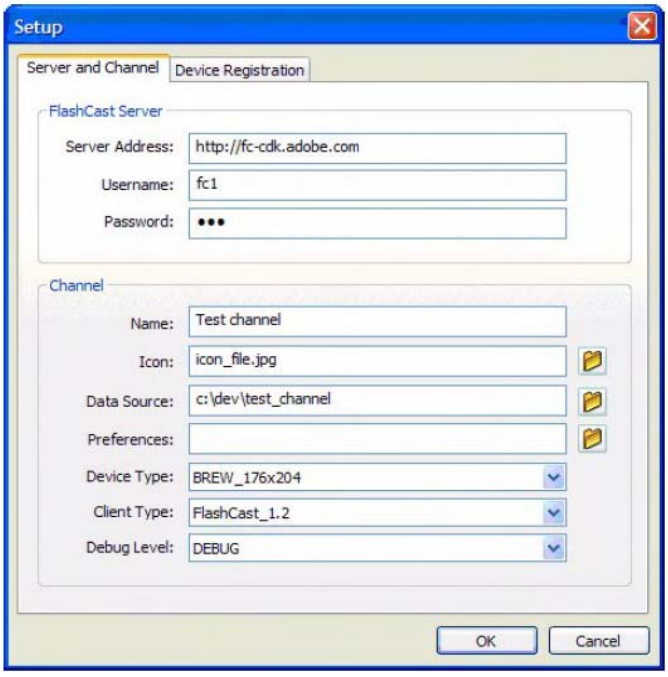
Troubleshooting Adobe Mobile AppBuilder

If you encounter a problem with the channel or the Flash Cast test server you can reset the Dashboard UI by doing one of the following:

- Type `#purple (#787753)` using the device keypad in Mobile AppBuilder to refresh the feedstore.
- Open `c:\Users\your user name\AppData\Local\Adobe\Adobe Mobile AppBuilder` and delete the `feedstore` directory.

Single-SWF testing

With Mobile AppBuilder, a Flash Cast channel developer can test their channel in the client emulator by using the standard **Test Movie** menu command in Flash CS3. The first time a developer tests their channel in Mobile AppBuilder, a dialog box appears requesting the developer's username and password, and the URL of the Mobile AppBuilder test-serve. This information is provided by the Mobile AppBuilder program administrator.

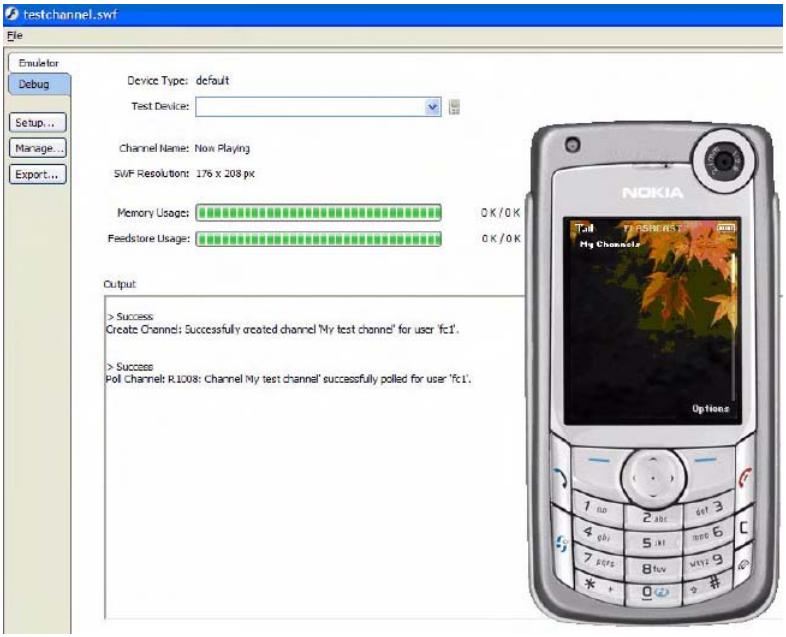


The developer's account settings and channel information is encrypted and stored within the FLA file. This ensures the developer does not have to re-enter all the settings again when re-opening a file. If a developer wishes to share his FLA source file with another developer, the new developer will need to re-enter the account information. This ensures that one developer's account information can not be used by someone else in possession of the FLA file.

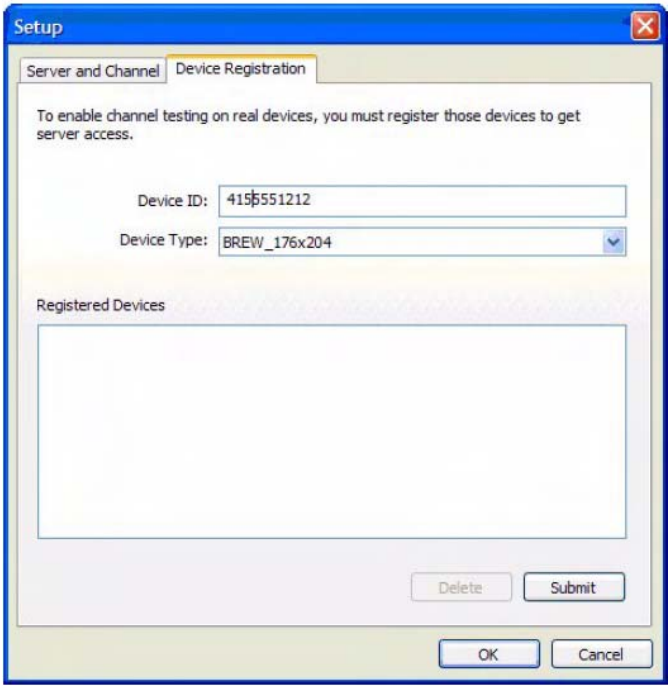
After the authentication procedure is successful, the published channel SWF file and all specified additional attachments (such as preferences, icons, or properties) are automatically uploaded to the Flash Cast server using a web service interface. The just-uploaded channel is automatically started and the user (represented by the developer) is subscribed to that channel.

A Flash Cast client emulator appears after uploading the content to the server. The emulator displays the skin of the selected mobile device and emulates the mobile phone. The user can now test the functionality and the design of his channel.

If the developer has a Flash Cast-enabled device, he can also test the channel on the device.This requires that developer's device be added into the developer's user group on the server. This can be done through the Mobile AppBuilder user interface.



A developer can add device IDs along with the corresponding clientType and deviceType to the server. A mobile phone that has been added is able to see all the content that has been created with the account through which the phone has been added. The mobile phone needs to have the Flash Cast client installed and a deviceID that matches the entry on the server (a deviceID should be unique on the server, otherwise it could be possible for a phone user to see other people's content).



For more information on the Flash Cast extension to the Flash Authoring Environment, please see *Developing Flash Cast Applications.*

Setting Up 3

This chapter contains information on setting up and running the software and environment for Verizon Wireless Dashboard application development, including:

- "Hardware requirements"
- "Software requirements"
- "Setup instructions"

NOTE Flash Cast channel development is supported on Windows-based computers. The Adobe Mobile AppBuilder is available only to registered developers. Developers can register through the Verizon ZON site at: http://www.vzwdevelopers.com/aims/

Hardware requirements

Your computer must have the minimum hardware requirements:

- 800 MHz Intel Pentium III processor
- 256 MB of RAM
- Windows XP
- 1024 x 768, 16-bit display
- 1.8 GB available disk space
- Your mobile handset must be equivalent to:
  - Samsung u-540
  - Samsung LG vx8700

Software requirements

The following software must be installed on your computer:

- Adobe Flash 8 Professional or Adobe Flash CS3 Professional
- Flash Mobile AppBuilder

The following software must be installed on your mobile handset:

- Verizon Wireless Dashboard

NOTE Developers will have to wait until Dashboard handsets are available on the market.

Setup instructions

The phases of this installation are:

1. "Installing Adobe Flash CS3 Professional"
2. "Installing the Adobe Mobile AppBuilder"
3. "Installing Flash Cast fonts for the Flash development environment"
4. "Setting up the Adobe Mobile AppBuilder extension for Adobe Flash CS3 Professional"
5. "Testing the installation in Adobe Mobile AppBuilder"

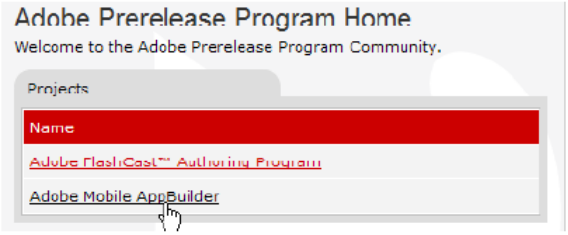
Installing Adobe Flash CS3 Professional

1. Browse to the Adobe download home page: http://www.adobe.com/downloads.
2. At the bottom of the **Featured product trials** section, select **Flash CS3 Pro** from the drop-down list and click **GO**.
3. Sign in with your Adobe ID and password or create an Adobe account if you do not have one.
4. Select the appropriate version from the menu drop down and click Download.
5. After the file is downloaded follow the setup instructions to install Adobe Flash CS3 Professional on your machine.

Installing the Adobe Mobile AppBuilder

1. Browse to the Adobe prerelease web site: https://prerelease.adobe.com/login.html.
2. Login in with your provided username and password.

3. Click the **Adobe Mobile AppBuilder** link located on the **Adobe Prerelease Program Home** page.



4. Click the **continue** button at the bottom of the **welcome** page.
5. Click the **Download Builds** link located in the **Resources** menu on the left side of the screen.

6. Click the **Download Adobe Mobile AppBuilder Build** and save the file to your machine.
7. Extract the contents of the .zip file and run the setup.exe file.
8. Follow the onscreen instructions to completion.

Installing Flash Cast fonts for the Flash development environment

1. On the Adobe prerelease site, click the **Adobe Mobile AppBuilder** link located on the **Adobe Prerelease Program Home** page.
2. Click the **Fonts** link located in the **Resources** menu.
3. Click on the **Fonts.zip** link and save the file to your hardrive.
4. Extract the **Fonts** folder from the compressed file.
5. Open the **Fonts** folder and right-click on the **win32** folder. Click **Search** to run a search on the **win32** folder.
6. Enter **.ttf* in the first search field to find all of the true type fonts.
7. After the search has finished, select all of the results and copy them.

8. Open the Control Panel and locate the **Fonts** folder.
9. Open the **Fonts** folder and paste the copied true type fonts into it.

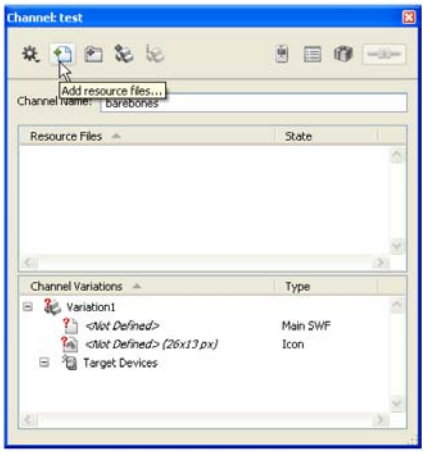
Setting up the Adobe Mobile AppBuilder extension for Adobe Flash CS3 Professional

1. Open Adobe Flash CS3 Professional.
2. Select **Help > Manage Extensions**.
3. Click the **Install New Extension** button.
4. Navigate to **My Computer\c:\Program Files\Adobe\Adobe Mobile AppBuilder** and select the **AppBuilder_Ext.mxp** file.
5. Click **Install**.
6. Make sure the checkbox next to **Mobile AppBuilder Extension** is checked and close the **Adobe Extension Manager** dialog.

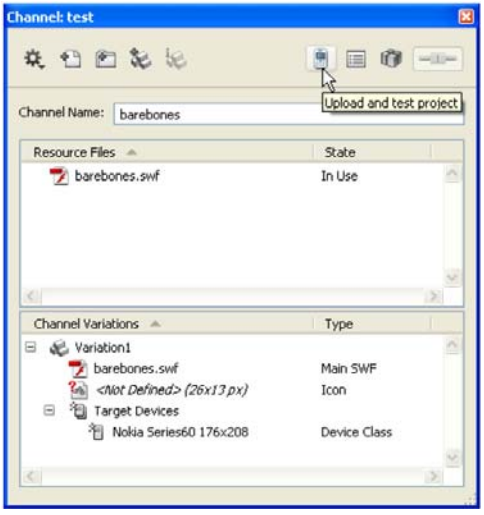
Testing the installation in Adobe Mobile AppBuilder

1. Browse to **c:\fccd140\channels\wt1**.
2. Open the **barebones.fla** file in Adobe Flash CS3 Professional.
3. Select **File > Publish Settings**.
4. Select **FlashCast 1.2 Client** from the **Version** drop-down menu and click **Publish**.
5. Click **OK**.
6. Open Adobe Mobile AppBuilder.
7. Click the **Channel Application** button under **Create New**.
8. Fill out the **New Project: Channel Application** dialog as follows:
   - **Project Name:** Test
   - **Save To:** c:\fccd140\channels\wt1
   - **Channel Name:** Barebones
   - **Server Address:** http://fc-cdk.adobe.com/feedserver
9. Enter your provided **User ID** and **Password**.
10. Click **OK**.

11. Click the **Add resource files** button.



12. Navigate to **c:\fccd140\channels\wt1** and select the **barebones.swf** file.
13. Click **Open**.
14. Expand **Variation1** and drag the **barebones.swf** to the **Main SWF** slot.
15. Double-click on **Target Devices**.
16. Mark the check box next to **Samsung LG vx8700** and click **OK**.
17. Click the **Upload and test project** button.



18. Click the center soft key on the emulator to enter the **Barebones** channel.

Getting Started 4

This chapter provides an introduction to Flash Lite and Flash Cast, including:

- "When to create Flash Cast channels"
- "Flash Cast user interface"
- "About the Verizon Wireless Dashboard"
- "Flash Cast client"
- "Flash Cast 1.2 server system"
- "Managing data in Flash Cast"
- "General channel testing process"
- "Managing a project's variations"
- "Exporting a project for deployment"

When to create Flash Cast channels

Many companies have successfully launched Flash Cast channels, however, not all mobile applications are suitable candidates for this solution.

Flash Cast is most useful for:

- Asynchronous, lightweight custom content updates.
- Always-on applications that can be viewed both online and offline.
- Interactive applications that only require small amounts of data like polls or short surveys.

Flash Cast is *not* recommended for:

- Heavily interactive applications like IM or real-time online games.
- Delivery of streamed media like videos.
- Large applications with large data or imagery needs.
- Heavily customized applications.

NOTE Flash Cast 2.0 supports streaming media.

Flash Cast user interface

Flash Cast implements a graphical channel metaphor, similar to television, that provides service providers with a centralized way of presenting mobile content to their subscribers.The Flash Cast service provided through Verizon Wireless is called Dashboard.

About the Verizon Wireless Dashboard

The Dashboard user interface includes:

Now Playing channel:

- A system channel that lists all the user-subscribed Flash Cast channels
- A Dashboard home screen

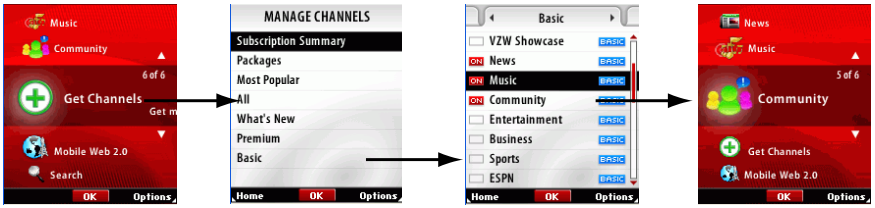
Channel Guide:

- A system channel that lists all the channels available to the subscriber
- Allows users to browse or subscribe to additional channels
- Users can access the Channel Guide through the Now Playing channel

Content Channels

- Flash Cast content movies
- Users can subscribe to multiple channels

System user interface: Supporting application elements like system menus and dialogs



Now Playing **Channel Guide** **Subscription** **Now Playing Channel Listing**

Flash Cast applications include (1) Now Playing (2) Channel Guide and (3) other system menus

Subscribing to channels

Dashboard users can subscribe to two types of channels:

- **Basic channels:** Free, default channels
- **Premium channels:** Additional cost channels (the premium channel system only supports a monthly subscription model)

Dashboard imposes a limit on the total number of channels to which users can subscribe. If users reach this limit, they are prompted to remove a channel before adding a new channel.

Flash Cast system

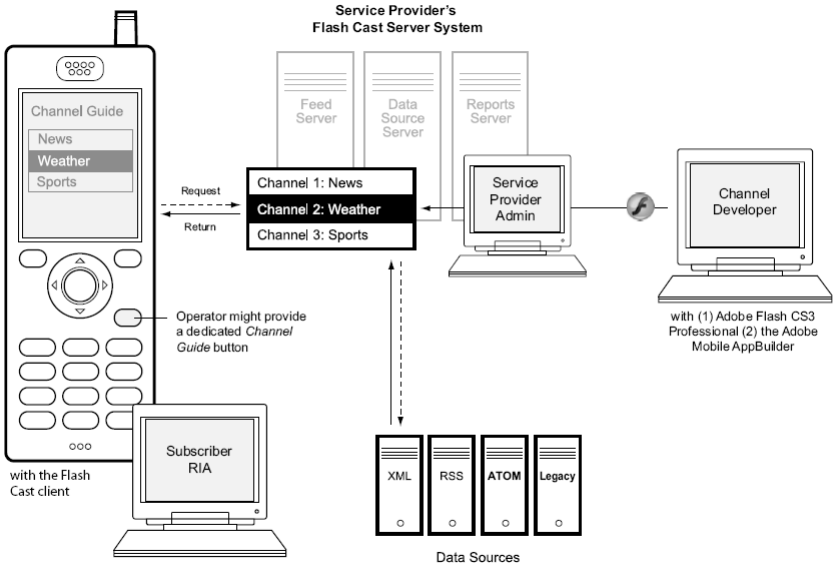
Flash Cast is a **white label offline portal solution**, which:

- Provides subscribers with custom online/offline content.
- Allows service providers who partner with Adobe to customize the implementation and brand of the technology, like Verizon Wireless Dashboard.

The players and technologies in the Flash Cast environment include:

- **Content developers** create Flash Lite user interfaces and provide data through external data sources.
- The **Flash Cast 1.2 server system** requests and aggregates data.
- The **service provider** markets and manages channels.

- **Mobile users** navigate the channels on their handsets and subscribe to the channels via the handsets or a desktop rich internet application (RIA).



An overview of the Flash Cast system includes (1) the mobile device with the Flash Cast client, (2) the Flash Cast 1.2 server system, (3) the channel developers and (4) the data sources.

Flash Cast client

The Flash Cast client and server system work together to serve and manage content for the mobile handset users.

Flash Cast client software

The Flash Cast 1.2 client consists of the following:

Feedstore

- Caches data for all channels
- Stores Flash SWF files for all channels
- Stores the currently running channel, Channel Guide data and system user interface elements

Flash Lite 1.1 player

- Renders SWFs including the currently running channel, the Channel Guide, system user interface elements and channels
- Loads the SWF file in and out of memory

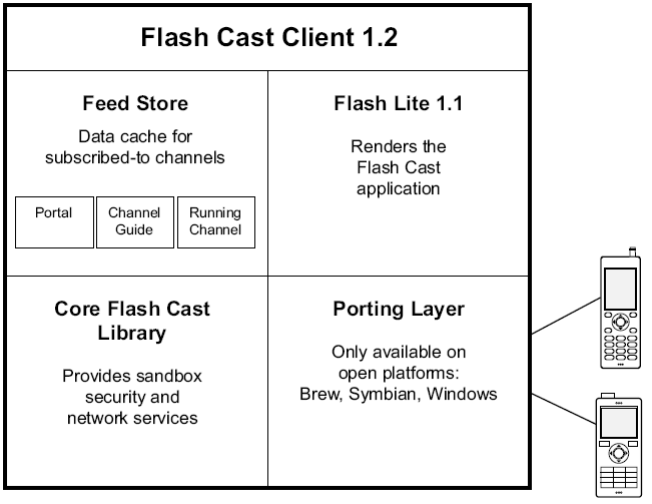
Porting Layer

- Enables the addition of the client to mobile handsets

Core Flash Cast Library

- Handles Flash Cast data services
- Protects the privacy of subscribers
- Prevents unauthorized parties from interfering with Flash Cast service
- Secures data for each channel
- Secures the client from other applications on the mobile handset

NOTE Once channels files are moved to the client, they cannot communicate directly with the server. Channel data is sent and received through the client.



Supported features of the Flash Cast client

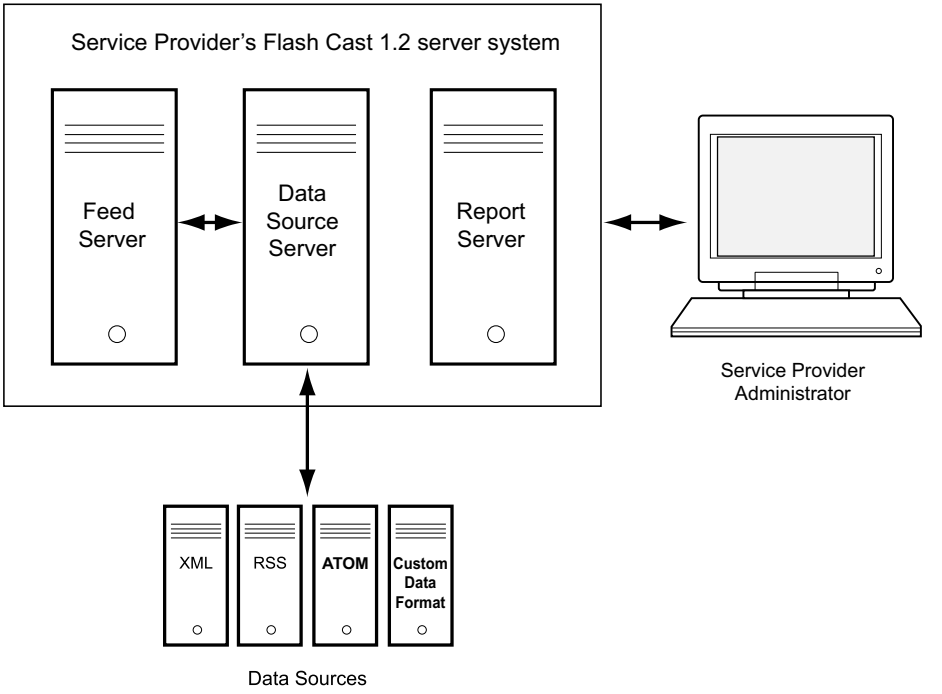
- Separates channel data from Flash Lite player rendering, supports:
  - Online/offline content viewing facilitated by cached data
  - Separation of channel browsing from data updates

    - Offline transactions (preferences, subscription), with updates made during the next network connection
- Interoperates with embedded applications on the mobile device including:
    - Phone calls
    - Browsers
    - Messaging
    - Media players
- Supports SMS signaling, enables the Flash Cast 1.2 server system to:
    - Wake up clients
    - Initiate immediate content updates in the case of breaking news
    - Initiate immediate retraction of inappropriate content
- Intelligent memory and resource management:
    - Minimizes the client's use of memory resources when other applications and services are in use.
    - Minimizes battery usage when in running in the background while periodically receiving updates from the feed server.
    - Allows hibernation which consumes only 5% of its normal operating memory. On some platforms it consumes 0% in this mode.
    - Can be configured to limit the number of channels to which a user can subscribe. When the limit is reached, the client asks users to remove channels before they can add new channels.

Flash Cast 1.2 server system

The Flash Cast 1.2 server system consists of three scalable servers:

- Feed Server - Manages preferences of the data source server and all mobile devices and sends updated incremental (or delta) content to each corresponding device.
- Data Source Server (DSS) - Queries external data sources and aggregates content.
- Reporting Server - Logs the activities from the feed and data sources servers and enables the generation of customized reports.



The Flash Cast 1.2 server system consists of the feed server, the data source server and the reporting server.

Managing data in Flash Cast

There are two distinct flows of data in a Flash Cast application:

- From the external data source to the Flash Cast 1.2 server system
- From the Flash Cast 1.2 server system to the mobile device

Each relationship occurs independently of the other.

External data

Retrieving and aggregating external data

Acquisition and aggregation of data is performed by the Flash Cast 1.2 server system:

1. The feed server requests incremental data from the DSS at predetermined intervals.
2. The DSS in turn requests and compiles data from the external data sources.
3. The data is stored in a common Flash Cast-specific XML format.

The stored data is known as the **feed state** (also referred to as the **feed model** or the **feed item collection**). The feed state:

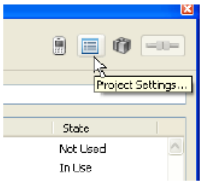
- Is made up of one or more **feed items**, which consist of a standard `type` and `id` value and one or more custom properties.
- Can store text data or images and SWF files as binary content.
- Contains all the unfiltered data available for a Flash Cast channel at a given moment.

Creating channel data sources

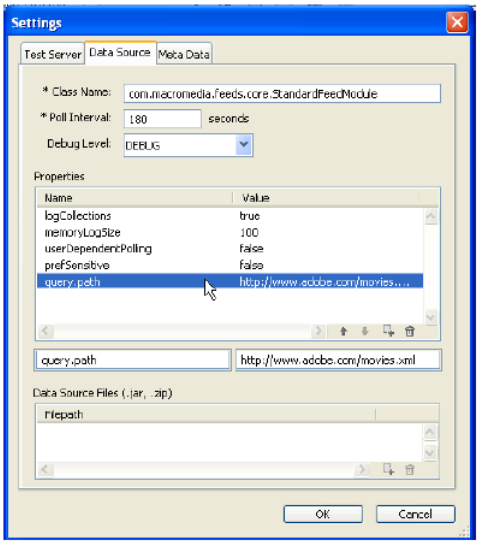
Any channel that requires external data must define the location of the XML data source(s) and the properties of the channel in the Settings dialog under the Data Source tab in Mobile AppBuilder.

Defining the query properties

Defining the channels data source is done by setting the query.path value to the XML file's HTTP address in the **Data Source** dialog. The **Data Source** dialog is accessed through the **Project Settings** button or by selecting **Project > Settings > Data Source**.



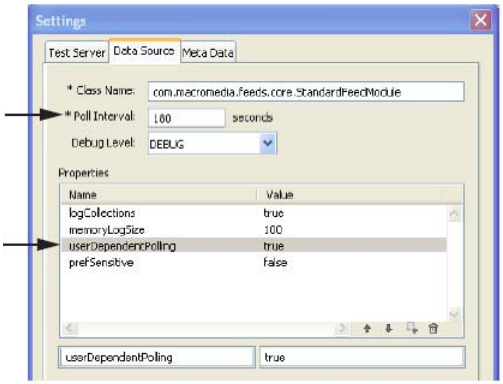
Project Settings button



Defining a data source path

NOTE The query.path value must be a fully qualified HTTP address.

The time interval at which a channel downloads updates from the server is defaulted to 180 seconds in the **Data Source** tab. This can be changed to any number of seconds up to 11 digits. It is also possible to set the channel so that the Polling Interval will depend on the user. This can be done by setting the `userDependentPolling` property to true in the **Properties** section of the **Data Source** tab.

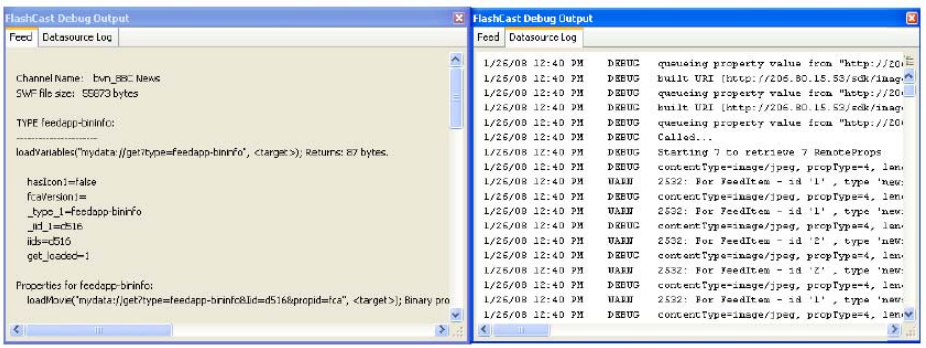


Setting a polling interval

Viewing the channel data with Mobile AppBuilder

When the Mobile AppBuilder runs the channel, it first loads the XML data to the server and then downloads it to the emulator, like it would to the Flash Cast client on the mobile handset.

The figure below shows the **FlashCast Debug Output** dialog content. The **Datasource Log** shows the interaction with the Flash Cast 1.2 server system and the **Feed** tab shows the variables available in the Flash Cast client feed store.



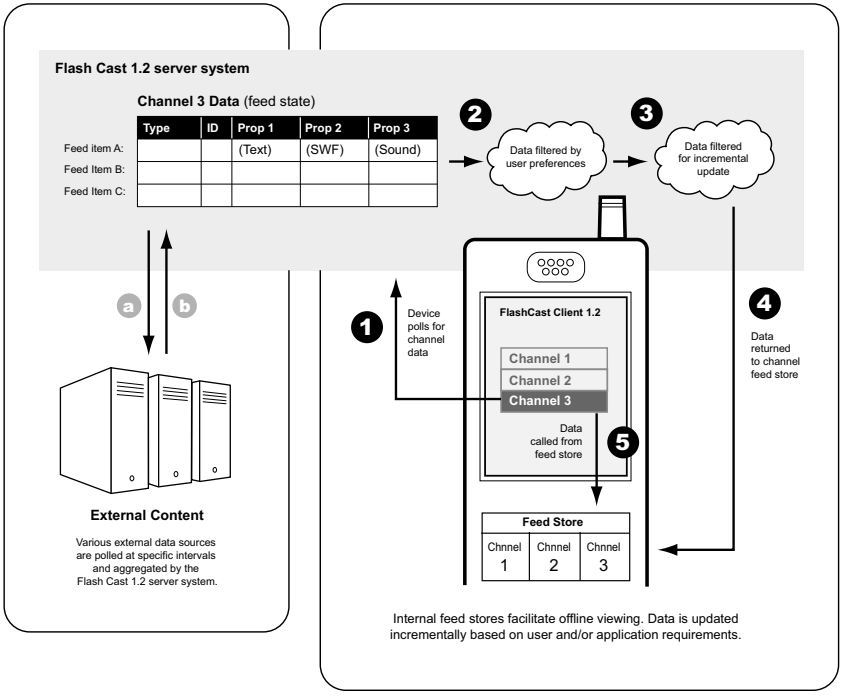
Adobe Mobile AppBuilder's Debug Output dialog shows the Datasource log

Requesting data to the mobile handset

Delivery, storage and display of data on the mobile device occurs in this order:

1. The Flash Cast client running on the mobile handsets requests data from the feed server over HTTP.
2. The packaging of data on the feed server into channel-specific **client feeds** and client **feed preferences.**
3. Client feeds are further filtered to ensure that only new content is passed to the mobile device over HTTP to minimize bandwidth consumption.
4. On the mobile device the data is placed into the feed store, which caches it for potential offline access.
5. The Flash Cast channel accesses the cached data in the feed store for display.

NOTE Flash Cast channels can only load data from the local feedstore. Unlike standard Flash or Flash Lite applications, they cannot load or post data directly over the network.



Data flow from external data sources to the Flash Cast 1.2 server system and out to the mobile handset.

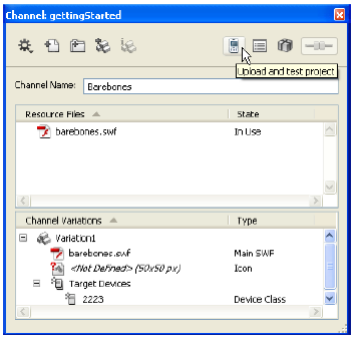
Exploring the Dashboard and navigating a channel

The following procedures require access to the Dashboard training course files.

Define the channel publish settings

To define the channel publishing settings:

1. In Flash CS3, open the file: `C:\fccd140\channels\wt2-1.`
2. Select **File > Publish Settings**.
3. Select **FlashCast 1.2** from the **Version** drop-down list, click **Publish**, and then click **OK**.
4. Open Adobe Mobile AppBuilder.
5. Under **Create New**, click the **Channel Application** button.
6. Fill out the **New Project: Channel Application** dialog as follows:
   - **Project Name:** gettingStarted
   - **Save To:** `c:\fccd140\projects`
   - **Channel Name:** Barebones
   - **Server Address:** `http://fc-cdk.adobe.com/feedserver`
7. Enter your provided **User ID** and **Password**, and then click **OK**.
8. Click the **Add resource files** button.
9. Navigate to `c:\fccd140\channels\wt2-1` and select the `barebones.swf` file.
10. Click **Open**.
11. Expand **Variation1** and drag the `barebones.swf` to the **Main SWF** slot.
12. Double-click on **Target Devices**.
13. Select the **Nokia Series60 176x208** box, and then click **OK**.
14. Click the **Upload and test project** button.



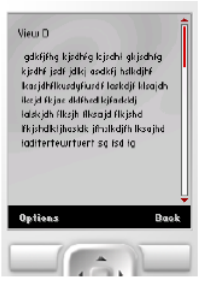
15. Click the **center soft key** on the emulator to enter the Barebones channel.

NOTE You should see View A of the Barebones channel.

Navigate a channel

1. Choose a button from View A and click the center soft key.

NOTE It does not matter which button you select as all of them transfer you to the story view.



2. Close the View D by clicking the left soft key.
3. Scroll to View B.

4. Click on the first button and see that other buttons appear.

5. Close Mobile AppBuilder.

General channel testing process

The testing process has two stages: single-profile testing and multi-profile testing. Single profile testing refers to testing a single SWF file channel. Multi-profile testing tests the combined multi-profile channel created as a Flash Cast ZIP file using the Export dialog box. In either case, you must have a live connection to a Flash Cast server and an account on the server with Administrator permissions.

Single-profile testing

Single-profile testing is the initial testing done while you are developing a channel for a specific device, or devices that share similar settings.

To test a channel:

1. In Flash, choose **File > Publish Settings**.
2. Select **FlashCast 1.2** in the **Version** field, and then click **OK**.
3. Set the size to match the device resolution of the mobile device:
   a. In the **Properties** pane, click **Size**, or choose **Modify > Document**.
   b. Change the dimensions to (for example) 176 pixels width and 208 (or 204) pixels height.
4. Click **OK**.
5. Select **Control > Test Movie**.

   If this is your first time running the emulator, a setup dialog box appears. In the setup dialog box, enter the following information, and then click **OK**:

- The URL of the test server
- The username and password of your admin account on the test server
- A name for your channel
- Optionally, the paths to an icon file, the directory that contains the properties file that defines the data source, and a default preferences file. To browse to these files, click the Folder button to the right of the text box. If you do not specify an icon file, the default Flash Cast icon is used.

If the application cannot locate any of the specified Icon, Properties, or Preferences files, you are prompted to check the paths you entered to point to valid directories.

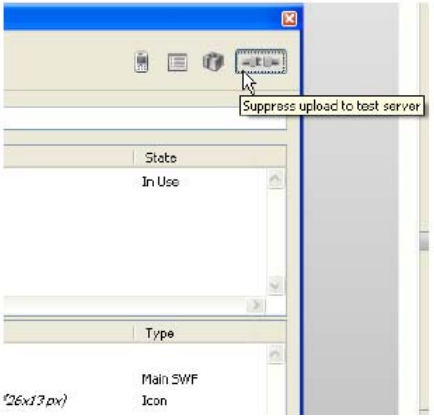
If the authentication process is successful, the server uploads your channel and any additional data or icon files, and the channel appears in the emulator.

6. Use the mouse to click on the emulated devices keys to test functionality and content views.

Testing locally with Adobe Mobile AppBuilder

Once a channel has been uploaded to the test server, changes may be made to the channel and tested locally with Mobile AppBuilder.

To do this you must click the **Suppress upload to test server** button. When the button is pressed, the **Channel Application** is "unplugged" and allows for channels to be published without the initialization of the Flash Cast test server.



Suppress upload to test server button

Testing on a Mobile Handset

Verizon Wireless Dashboard devices are pre-installed with the Flash Cast Client.

To test channels on the handset, you must:

1. Register your handset in the **Server > Register Devices** dialog of Adobe Mobile AppBuilder using the handset International Mobile Subscriber Identity (IMSI) number.

   NOTE The IMSI is also commonly referred to as the IMEI or International Mobile Equipment Identity. The IMSI of a phone can be found by dialing the code *#06#.

2. Publish the channel using Adobe Mobile AppBuilder to the target device.
3. From the handset access Dashboard and subscribe to your test channels.

NOTE Your handset is not automatically subscribed to your channels, even if it is registered. You must subscribe to the channel using the Test Channels category of the Channel Guide.

Managing a project's variations

When you create a project, Mobile AppBuilder creates an empty variation which you must edit.

You may optionally createmultiple variations for your project. You can also edit, sort, and delete variations. For each variation, you must specify:

- the SWF
- the file for the icon
- the target device(s)
- optional preferences

NOTE Mobile AppBuilder immediately saves any changes you make in the Channel Variations panel.

Create a new variation

You can create a new variation in one of the following ways:

- Click **Add New Variation** on the project dialog box.
- Duplicate an existing variation. To do so, right-click the variation on the project dialog box, and select **Duplicate**. The new variation is identical to the original, except it has no target devices and a new default name -- **Copy of <original name>**.

Edit a variation's resource files and target devices

Before you edit the variation, add resources to your project as described in "Add resources to a project" in the Application Development Guide.

Then edit a new empty variation or an existing variation as follows:

1. Double-click on the variation in **Channel Variations** in the project dialog box to bring up the **Variations Info** dialog box.
2. Edit the name of the variation.
3. Under **Main SWF**, click **Choose**.
4. Select a SWF on the **Choose Project Resource File** dialog box and click **OK**. Alternatively, click **Browse System**, navigate to a SWF, and click **Open**.
5. Under **Icon**, click **Choose**.
6. Select an icon file on the **Choose Project Resource File** dialog box , and click **OK**. Alternatively, click **Browse System**, navigate to an icon file, and click **Open**.
7. Click **Target Devices**.
8. Select one or more device classes for the variation. As you select each device class, the **Variations Info** dialog box displays the class devices, the Client Type, and the Device Type.
9. Click **OK** to save your changes. In the project dialog box, the updated variation expands to show the resource and target information.

About variation preferences name-value pairs

You can specify the default application preferences for each variation. If you do not specify any default application preferences, then theMobile Server delivers the entire feed item collection to the client database when the subscriber first installs the application. For more information about default application preferences, see "About default application preferences" in the Application Development Guide.

You specify the default application preferences for a variation by using name-value pairs. The name you enter for each name-value pair is one of the following:

- **select<select index>type**
- **select<select index>items**
- **select<select index>props**

The meaning of this syntax corresponds to the XML elements and attributes described in *Format of XML Preferences in FlashCast1.2 ActionScript* and *Data Format Reference* as follows:

- **select** - This corresponds to the select element.
- **<select index>** - This corresponds to the index attribute of the select element.
- **type** - This corresponds to the type attribute of the select element.
- **items** - This corresponds to the item element.
- **props** - This corresponds to the prop element.

The value you enter for a name-value pair corresponds to attribute values of elements in the feed item collection schema as described in *FlashCast 1.2 ActionScript* and *Data Format Reference*. The name-value pair correspondence is:

Name	Value
`select<select index> type`	`The value of the name attribute of a type element, or a space-delimited list of the values of name attributes of type elements.`
`select<select index>items`	`The value of the id attribute of an item element, or a space-delimited list of the values of id elements of item elements.`
`select<select index>props`	`The value of the name attribute of a prop element, or a space-delimited list of the values of name attributes of prop elements.`

In a stock ticker application example, you might set these application preferences. First, if you want to get all the information for all companies:

```
select1type ticker
```

However, if you want the default to get all the information for just a few specific companies:

```
select2type ticker
select2items ADBE IBM SNE
```

As a final example, if you want the default to get a subset of the information available for just a few specific companies:

```
select3type ticker
select3items ADBE IBM SNE
select3props todaysHigh todaysLow percentChange
```

Add variation preferences

1. Double-click on the variation in **Channel Variations** in the project dialog box to bring up the **Variations Info** dialog box.
2. Click **Preferences**.
3. Click the **New** icon
4. Enter the preference name and value in the text boxes.
5. To add more preferences, click the **New** icon again, and enter the preference name and value in the text boxes.
6. Click **OK** to save your changes.

Organize variation preferences

To order the preferences for readability, do the following.

1. Double-click on the variation in **Channel Variations** in the project dialog box to bring up the **Variations Info** dialog box.
2. Click **Preferences**.
3. Select the name-value pair for the preference you want to move up or down.
4. Click on the **Move Upwards** icon or **Move Downwards** icon to move the preference up or down.
5. Click **OK** to save your changes.

Delete variation preferences

1. Double-click on the variation in **Channel Variations** in the project dialog box to bring up the **Variations Info** dialog box.
2. Click **Preferences**.
3. Select the name-value pair(s) for the preference(s) you want to delete.
4. Click on the **Delete** icon .
5. Click **OK** to save your changes.

Delete a variation

1. Right-click a variation in **Channel Variations** in the project dialog box.
2. Select **Delete**.
3. Click **Yes** when asked to confirm the deletion.

Sort variations

Variations are sorted into ascending or descending alphabetical order in the project dialog box. An up arrow to the right of **Channel Variations** indicates the variations are sorted in ascending alphabetical order, and a down arrow indicates descending alphabetical order. Clicking **Channel Variations** toggles between ascending and descending alphabetical order.

Exporting a project for deployment

When you are ready to deploy your application on a Mobile Server, use Mobile AppBuilder to export the necessary files of the project intoaZIP file togive to the server administrator. Before youexport a project,make sure the project has access to the SWF and icon files you want to export, and that your variations and data source information are set up the way you want. When you export a project,Mobile AppBuilder puts copies of the following files into a ZIP file:

- All SWF and icon resource files that you included in at least one variation. Mobile AppBuilder renames each file with a globally unique identifier, a GUID, to ensure the file names are unique on the mobile server. For example, OceanLife_complete.swf might be renamed to 0_FFEA1C44-2F14-4603-A186-A9E215BBE728.swf.
- Preference files.MobileAppBuilder creates a PREF file for each variation forwhich you specified preferences , and names each PREF file with a GUID.
- All data source files that you included in the project settings. Mobile AppBuilder puts a prefix on each file name.
- A db.xml file. Mobile AppBuilder creates a db.xml file to provide the server the information about the project, including, for example, the project's data sources, variations, meta data, and application name.

NOTE The files included in the ZIP file for exporting a project differ from the files in a ZIP file for sharing a project . The ZIP file for exporting a project does not contain the project MABP file, source files such as FLA files which you may have added to the project, or any resource files which are not part of a variation.

To export a project:

1. Open a project.
2. Select the **Select Action** icon on the upper left of the project dialog menu.

3. Select **Export Project For Deployment.**
4. In the **Export** dialog box, navigate to the directory to which you want to save the ZIP file.
5. Accept the default name or enter a name of your choice. The default name is <project name>.zip, and is recommended so that the ZIP file for exporting a project always has a different name than the ZIP file for sharing a project.
6. Click **Save**.

The ZIP file is created in the directory you specified. You must deliver it to the server administrator outside of Mobile AppBuilder.

Typical Development Procedures

5

This chapter describes typical concepts and procedures used in developing Dashboard applications, including:

- "Device profile definition"
- "Configuring the channel"
- "Channel displays"
- "Adding ActionScript to the views"
- "Controlling the speed of animation"
- "Publishing using multiple devices"

For specific instructions, refer to the appropriate Adobe documentation.

Device profile definition

Each channel is designed for a specific device. Since the Channel Development Framework handles the navigation, you must provide the framework with the device dimensions so that it can dynamically build the display. The device profile is defined in the device_profile.as file, stored in the channel actionscript directory. The `device_profile.as` file, stored in the channel `actionscript` directory, should contain the variables listed below (values given are samples to illustrate syntax) and will change if the effective screen size of the device changes:

Device profile file contents

The `device_profile.as` file should contain the following variables (example from QCIF):

- Screen width and height
  - `screen_width = 176;`
  - `screen_height = 203;`
- The height of the channel header
  - `top_margin = 0;`

- The height of the soft key menu bar
  - `bottom_margin = 16;`
- Scrollbar properties
  - `sb_y = top_margin + 3; //scrollbar vertical position`
  - `sb_x = screen_width - 9; //scrollbar horizontal position`
  - `sb_ht = screen_height - top_margin - bottom_margin - 6; //scrollbar height`
  - `sb_thumb_ht = 40; //scrollbar thumb height`
- Scrollbar vertical position

  `sb_y = top_margin + 3`
- Scrollbar horizontal position
  - `sb_x = screen_width - 9;`
- Scrollbar height
  - `sb_ht = screen_height - top_margin - bottom_margin - 6;`
- Scrollbar thumb height
  - `sb_thumb_ht = 40;`
  - These procedures are covered in overview. For more detailed information, please refer to the Dashboard training materials and run the lab exercises using the course materials.

Configuring the channel

These procedures are covered in overview. For more detailed information, please refer to the Dashboard training materials and run the lab exercises using the course materials.

To configure the channel:

1. Review the current device profile configuration file. Make any necessary changes.
2. Create necessary content: movie clips, text fields, etc.
3. Add Var names to text fields.
4. Nest neadlines in movie clips, set other content associations.
5. Publish to the standalone emulator:
   a. Open the standalone emulator.
   b. Select **Settings > Connection Settings** and click **OK** to refresh the Verizon Wireless Dashboard **Now Playing** screen.
   c. Select **Get Channels**.
   d. On the **Manage Channels** screen, select **Test Channels** and click **OK**.
   e. On the BBC News channel you created, click **OK**.

      NOTE You should see a prompt that asks "Subscribe to this channel?"

f. Click **OK**.

g. After your changes are saved by Dashboard, use the PageDown key (F1) to navigate back to the Verizon Wireless Dashboard **Now Playing** screen.

h. Navigate to your channel to view it.

Channel displays

Each display in a channel is managed by the Channel Development Framework. When you add a screen to the channel, you must:

- Organize the displays into stacks and views.
- Add the views to the `views_mc movie` clip.
- Define the stacks and views in the channel profile.
- Attach ActionScript code to the movie clip to initialize it in the framework.

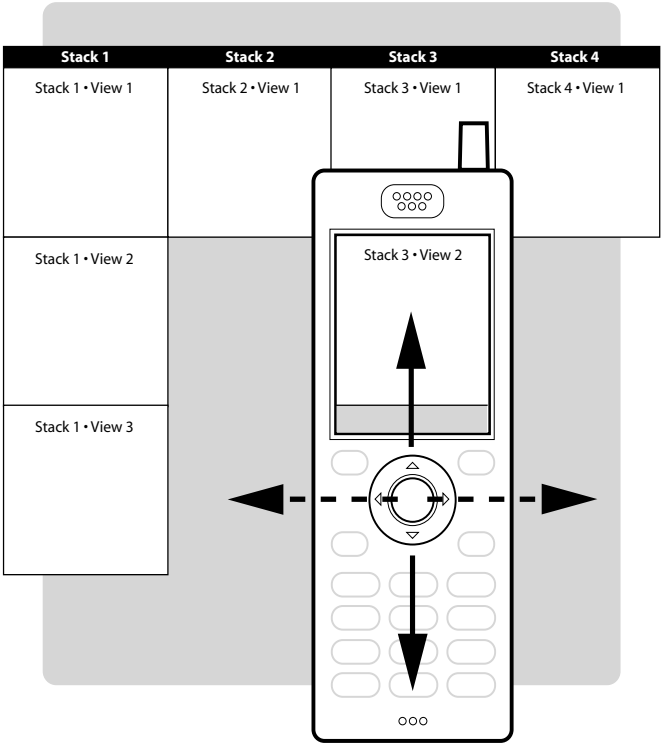
Stacks and views

Channel displays are defined in the Channel Development Framework as stacks and views:

- **Stacks** are columns of related displays.
- **Views** are the individual displays in a stack.

When the channel runs, the framework physically creates a grid-like structure of stacks and views, as shown in the following figure. As users navigate the channel, the framework animates the grid of structures and views to display the appropriate content.

Stacks and views are not a grid in the sense of an Excel spreadsheet or database table because views along the same row are not related to one another. Views are organized based on their relationship to other views in the same stack.



Create views

In Flash terms, a view is a complete movie clip that represents a screen or page of the Flash Cast channel. All views are placed within the `views_mc` movie clip that exists in the framework FLA file.

The framework programmatically creates the stacks and views grid when the channel initializes. When you place each of the view movie clips into the views_mc movie clip, you should try to visually mimic what the framework will do programmatically.

Mimic the framework stacks and views grid in the views_mc layout

Define the channel profile

Channel profile settings are defined the `channel_profile.as` file in the channel framework's `actionscript` directory. There are three primary configuration elements in the channel profile:

- Number of stacks
- Number of views in each stack
- Data associated with channel and each view

Error view

The Channel Development Framework provides an error movie clip, `e_0_mc`, that displays a message if data is not properly loaded into the application. This movie clip is the first view in the error stack.

Defining channel stacks

In the `channel_profile.as` file, you must number and name the stacks in the channel, as shown in the following code:

```
stack_0 = "headline";
stack_1 = "story";
stack_2 = "error";
total_stacks = 3;
```

Also define the `total_stacks` variable, which the framework requires.

The framework starts numbering elements with zero.

Defining stack views

To define views in a stack:

1. Create one variable for each view.
2. Name each variable with the stack name, followed by an underscore (_) and an incremented number, starting with zero.
3. Set each view variable equal to the string name of the associated movie clip.
4. Create a variable to hold the total number of views in that stack. Name the variable with the stack name prepended with the string total_ and appended with an “s”.

The following code defines the views for the headline stack:

```
headline_0 = "h_0_mc";
headline_1 = "h_1_mc";
headline_2 = "h_2_mc";
total_headlines = 3;
```

Defining the channel data

The first configuration values in the channel_profile.as file is related to channel data. Define a variable named `total_types` and set it equal to zero, to denote that there is no data associated with the channel.

```
total_types = 0;
```

Creating the content stacks

The three steps to create content stacks are:

1. Create the headline and story stack.
2. Update the channel profile configuration file.
3. Publish to the standalone emulator and view the results.

Switching stacks, views, and elements

The framework will automatically handle navigation between views in the same stack or elements in the same view. However, you can also target a view in a different stack or even a particular view with an specific element selected.

1. Uses the `tellTarget()` function to target the `framework_mc` movie clip. Within the function you should:
2. Specify whether tweening is on (true) or off (false) with the `tween` property.
3. Specify a particular stack to move to using the `stack_name` property.
4. Specify a particular view to move to using the `index` property.
5. Use the call ("`setView`") function to initialize the declared view.
6. Specify a particular element to highlight using the `element_index` property.
7. Use the ("`setElement`") function to initialize the declared element.

Currently, the framework tweens within a stack but not between stacks because the diagonal animation or pass-by of views can lead to unexpected results if you move between views that are not on the same row.

The following code sample tells the framework to animate to the first view in the story stack and highlight the first element.

```
tellTarget ("/framework_mc") {
  tween = false;
  stack_name = "story";
  index = 0;
  call("setView");
  element_index = 0;
  call("setElement");
}
```

Adding ActionScript to the views

Each view must also contain some ActionScript code to initialize it with the framework. Each view movie clip must contain:

- The visual display elements for the view in the first frame.
- A `stop()` command on the first frame of an ActionScript layer.
- A frame named `init` on the second frame of the ActionScript layer.

To initialize a view in the framework:

1. Create a directory in `channels/[channelName]/actionscript` for your view. Name the view directory with the view name appended `with _view`, for instance `headlines_view`.

2. Create an `init.as` file in the view directory.
3. Use the `#include` command on the init frame of the view to include the init.as file.

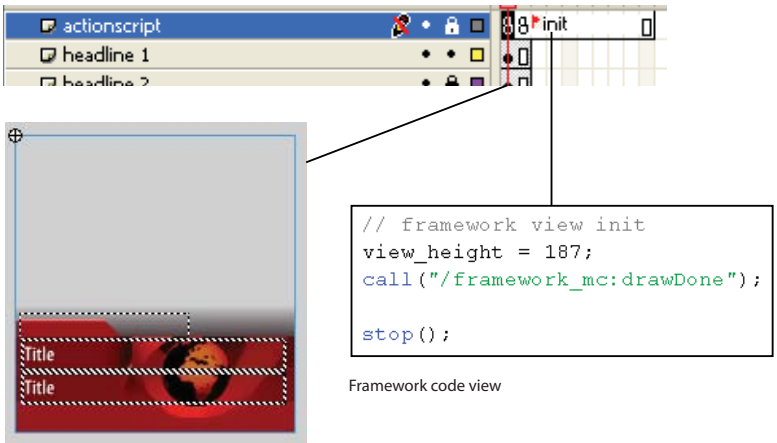
   NOTE By convention, code that is more than one line should be placed in a separate AS file.

4. Add the following code (with the appropriate values) to the `init.as` file:

```
view_height = 187;
call("/framework_mc:drawDone");
stop();
```

   - The first line defines the height of the view so the framework can handle the view display.
   - The `call()` function runs the *drawDone* code in the framework, which tells the framework to draw the view.
   - The `stop()` command ensures that the view movie clip stops.

Initializing a View in the Framework



Framework code view

Initializing channel views

Perform these steps as necessary to populate headline views with text and display headline views.

1. Include the stub data.
2. Initialize the headline view in the framework.
3. Enter the timeline of the movie clip:
   a. Select the actionscript layer and adjust its timeline if appropriate.
   b. Insert an init keyframe in the second frame of the actionscript layer.

c. Adjust other timeline elements as appropriate.

4. Specify the category being polled from the `hydrate.as` file
5. Define headline values based on the variables in the `hydrate.as` file.
6. Set the framework `display_height` variable.
7. Make other changes as appropriate
8. Add a stop command.
9. Initialize the story view:
   a. Enter the story view movie clip timeline.
   b. Add a new init layer above the actionscript layer.
   c. Add a keyframe in frame two of the init layer.
   d. Label the second keyframe init, and open the actions panel to insert an include statement that will access the story view file.
   e. Add code to the init.as file that tells the framework to display the code.
   f. Save and publish the file.
10. View the channel in the standalone emulator to check the results.

Controlling the speed of animation

The framework sets default values for some features of a channel. You can customize these values via ActionScript. One example of this is the rate at which an application switches views, or tweening.

NOTE Explore the framework code in the shared_actionscript directory to find more default values that you can control.

The framework defines the default tween ratio in the `c:\fccd140\shared_actionscript\templates\v3_template\framework\initTween.as file` on line 45.

```
tween_ratio = (/:tween_ratio eq "") ? 1.8 : /
:tween_ratio;
```

If you want to change the tween ratio in your channel input, define a value for the `tween_ratio` variable in the same init frame that defines your device and channel profiles.

Setting the `tween_ratio` equal to a higher number makes the channel switch between.

Publishing using multiple devices

This section explains how to publish a channel in Adobe Mobile AppBuilder using multiple devices. How to proportionately scale an image and how to upload device specific images dynamically are also discussed.

Testing with Multiple Devices

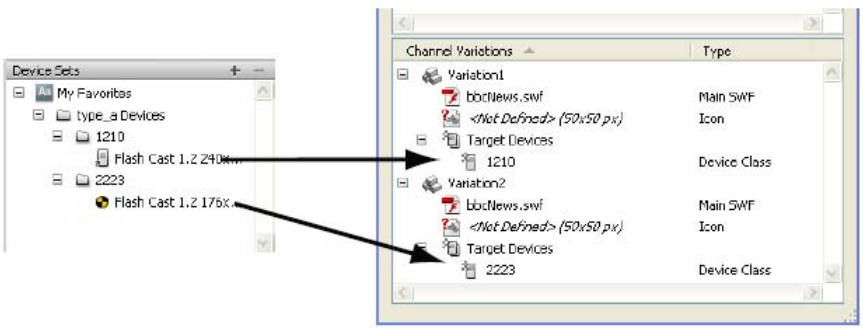
Using Adobe Mobile AppBuilder you can publish a single SWF or multiple SWF's with multilple phone profiles. This makes testing more efficient when developing a channel for more than one device.

This is all done within the same channel application.

Multiple variations

Channel Variations in Mobile AppBuilder allow the user to test a project on as many devices as are defined in the solution.xml being used. Each variation's SWF may be the same or different, but must be a resolution equal to or lower than that of the selected client.

The client selected within the **Device Sets** panel determines which variation will be published. A channel variation may have multiple target devices if there is only one SWF being tested.



The Device Sets panel determines which channel variation will be published.

NOTE Default preferences are tied to a variation because they may be different for each target device.

Scaling Images For Best Fit

Most cellular phones operate on one of three display types, QCIF, QVGA, or WQVGA.

- QCIF stands for Quarter Common Intermediate Format and used in phones with a resolution smaller than or equal to 176 x 203 pixels
- QVGA stands for Quarter Video Graphics Array and is a popular term for a computer display with 320 x 240 resolution. Quater VGA is derived from the fact that it offers 1/4 of the maximum resolution of VGA display technology
- WQVGA stands for Wide Quarter Video Graphics Array. WQVGA displays offer a resolution of 240 x 400 pixels

Because Flash Cast channels are created for different screen sizes, an image may need to be scaled to fit a higher resolution handset.

For example, if a bitmap image being used on a QCIF device is applied to a higher resolution device it will appear blurry. A possible way to solve this problem is to apply ActionScript that will upscale the image to fit the resolution of the larger device display.

This scaling can be done by applying the correct code to the `device_profile.as` file.

- Define the height and device you wish to upscale to.

```
_device_W = 240;
_device_H = 298;
```

- Create a variable that is equal to the aspect ratio of the device that it is being upscaled to.

```
aspect_ratio = _device_H / _device_W;
```

- Create a variable that multiplies the QCIF width by the ratio created above.

```
aspect_corrected_height = Math.ceil(176 *
aspect_ratio);
```

- Create a ratio that is equal to the smaller device width divided by the device width variable of the larger resolution device. Multiply that number by 100 to achieve a number larger than 1 since you want the image on the screen to grow.

```
scale_ratio = (176 / _device_W) * 100;
```

- Change the screen height to equal the value of the `aspect_corrected_height` variable so that the current channel's aspect ratio is now the same as the higher resolution device.

```
screen_height = aspect_corrected_height
```

- Scale the image using ActionScript to adjust the images size depending on the `scale_ratio` value created in the device profile. The following code must be placed in the same frame as the include function that is accessing the device profile.

```
image_mc._xscale = _root.scale_ratio;
image_mc._yscale = _root.scale_ratio;
```

Loading Device Specific Data

Images that are loaded dynamically from an online data source can be set to be device specific.

Each device is assigned its own platform ID. These platform IDs if unknown, may be found using a trace of the `/:fca_platformID` variable.

When defining the data in the online data source, the value of the `fca_platformID` variable needs to be added to the end of each type of data.

```
<data>
   <type name=”news1_1234”>
```

The fca_platformID variable needs to be appended to the data types being called dynamically to the channel as well. The fca_platformID variable will always call the ID of the device being emulated.

```
type_0 = "c1_0000";
type_1 = "_featured_links_" add /:fca_platformID;
type_2 = "_get_more_" add /:fca_platformID;
type_3 = "_search_" add /:fca_platformID;
total_types = 4;
```

Targeted Development Tasks 6

This chapter provides concepts and procedures on targeted development tasks, including:

- "Linkaway to other content"
- "Get user input in Dashboard"
- "Channel data options"
- "Reviewing pseudo-arrays"
- "Storing reusable methods centrally"
- "Scrolling text"
- "Reviewing looping"
- "Loading binary data"
- "Adapting a channel for user preferences"

Linkaway to other content

A Flash Cast channel can launch other applications and services on the device provided by Verizon Wireless. In addition to the regular editorial content, the channel also provides "links" to related content on the Verizon Wireless content deck.

Types of linkaways

There are four main types of linkaways:

- **Mobile web**—Open a page in the device's browser.
- **BREW application**—Open a resident BREW application, or, if the application isn't installed, go to the purchase page for that application on the Mobile Shop.
- **Search**—Open the Get it Now Search application, optionally with a specific search string.

Linkaway data

Except for links to Mobile Web sites (see "Open a web page"), which only require knowledge of site's URL, the other three types of linkaways require information about the application or piece of content to launch. Each BREW application has a class ID, which identifies that application on the device. If you don't know the class ID, you can't launch the application from a Dashboard channel.

ActionScript API

The linkaway ActionScript API is used by Flash Cast Channels to launch other BREW applications.

FSCommand ("Launch")

FSCommand is a standard Flash ActionScript API that allows the Flash player to extend its API set on particular platforms. The launch command works on many platforms and always attempts to launch an external process. The launch command on the BREW platform works as follows:

ActionScript Call

```
FSCommand ("Launch", "<appname>  <mimetype>  <itemid> <args>" );
```

Parameters:

- **Launch:** Tells the FSCommand API to invoke the *launch* extended command.
- **<appname>**: (optional). Application name or application class id.
  Valid appname values are application class ids and the following mobile shop specific strings:
  - mobileshop:Catalog
  - mobileshop:Search
  - mobileshop:ItemID
  - mobileshop:UpgradeCheck
- **<mimetype>**: (optional). The registered mime-type of the target application.
- **<itemid>**: (optional). The applications part ID in the operator's Application Download Server, (ADS). This allows BREW applications that aren't yet installed to first be downloaded and then launched, if required. The Dashboard client uses this parameter to display the appropriate purchase page in the "Get It Now" application.
- **<args>**: (optional). This parameter indicates the payload for the launched application.

- **<startmode>**: (optional). By default at the ActionScript level caller doesn't need to specify. If nothing specified, target application is being launched normally (via ISHELL_StartApplet/ISHELL_StartAppletArgs API). There are applications that need to be started alternatively and this parameter supports that mechanism. Currently, other than "default" mode, only "sms" mode is supported. If none specified, "startmode" would be "normal". Possible values for "startmode" are default and sms.

ActionScript API examples

The following examples use the ActionScript API to demonstrate various types of linkaways.

Launch pre-installed BREW application

The following code sample shows how to launch a pre-installed BREW application via mime type or class id.

```
FSCommand("Launch",  "appname 0x98099689 mimetype audio\ringtone args
  mytone.wav");
```

Launch a non-resident BREW application

The following code sample shows how to launch a non-resident BREW application.

```
FSCommand("Launch",  "appname 0x98099689 mimetype audio\ringtone itemid
  3920129 args mytone.wav");
```

If the operator wishes to integrate with a newly available BREW application the operator can update the Channel with the appropriate launch parameters and the new application can be used as a linkaway.

Launch mobile shop in alternate modes

If you wish to launch mobile shop in various modes (like downloading new item, upgrade checking, searching etc.), following way mobile shop can be launched.

1. To launch mobile shop catalog (note 'mobileshop:Catalog' is reserve word):

   ```
   FSCommand("Launch",  "appname mobileshop:Catalog");
   ```

2. To launch mobile shop and search for a given string (note 'mobileshop:Search' is reserve word):

   ```
   FSCommand("Launch", "appname mobileshop:Search args <search string>");
   ```

3. To launch mobile shop and go to the application download page (note 'mobileshop:ItemID' is a reserve word)

   ```
   FSCommand("Launch",  "appname mobileshop:ItemID itemid <item id>");
   ```

4. To launch mobile shop and check upgrade for a given catalog ID or item id (note ‘mobileshop:UpgradeCheck’ is reserve word)

```
FSCommand(“Launch”,  “appname mobileshop:UpgrageCheck appname <class id>
  mimetype <mimetype string> itemid <item ID>”);
```

NOTE For “appname” if you specify a hex or decimal number, or don’t specify anything, and then for “itemid” if you specify the item id or the catalog number, mobile shop always gets launched in download mode as described in step 3.

Open a web page

To open a web page in the device's native web browser, use the following syntax:

```
fscommand("launch", "args URL")
```

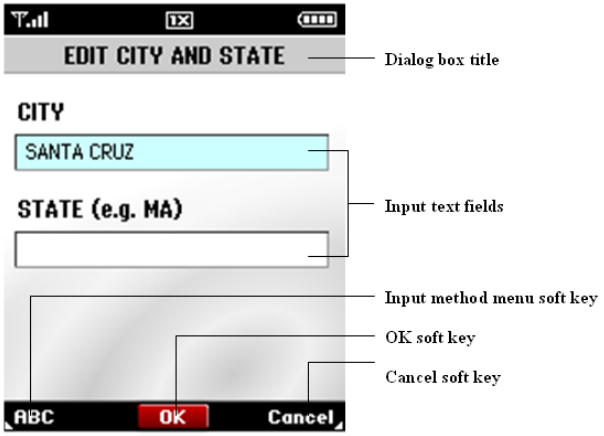
Where URL is the fully-qualified URL to the web page to open.

Get user input in Dashboard

To get text input from users, use the EditText fscommand. The command displays a modal dialog box in which users can enter information in up to three input text fields. The user navigates between text fields using the up and down arrow navigation keys. To commit changes to text input the user presses OK or Cancel to quit the dialog without saving changes.

The EditText command accepts several parameters that a Dashboard channel can specify including the dialog box's title, the number of text fields to display, a label to display above each input field, and the initial text input method (for example, all capital letters or numbers-only), for example.

This mobile device dialog contains two input text fields: city and state.



The code to display this dialog is:

```
fscommand("EditText", "title=EDIT CITY AND
  STATE&numfields=2&static1=CITY&textvar1=cityVar&static2=STATE(e.g.
  MA)&textvar2=stateVar");
```

Complete reference documentation is provided under command usage and parameters, below. The EditText command and its parameters are passed as a string of name-value pairs connected with ampersand (&) characters.

In the above example, the title is set to "EDIT CITY AND STATE". The numfields parameter specifies that the dialog box needs to display two input text fields. The first is labeled CITY and is associated with the ActionScript variable named cityVar. Similarly, the second text field is labeled STATE and is associated with the ActionScript variable named stateVar. If the user presses OK, the dialog box assigns the contents of input text fields to their corresponding ActionScript variables in the channel SWF, cityVar and stateVar in this case. If the user presses Cancel all edits are discarded and no changes are returned to the SWF.

Some considerations:

- While the dialog is open, the Dashboard channel is effectively stopped—no ActionScript is processed and the timeline is paused.
- The Dashboard client does not generate any ActionScript key press events in response to the user pressing either OK or Cancel; a result it's not possible to directly detect which key the user pressed.

Command usage and parameters

The EditText FSCommand API is:

```
fscommand("EditText", "parameter_string")
```

Where parameter_string contains a string of name-value pairs concatenated by ampersand (&) characters. Below are valid parameter names and values.

- **title:** Specifies the title of the text input dialog.
- **numfields:** The number of text fields to display in the dialog. The input dialog supports a maximum of three input fields.f
- **textvarN:** The name of the ActionScript variable to associate with the Nth input text field, where N is an integer between 1 and 3 that specifies the input text field, where 1 is the top-most input text field and 3 is bottom-most text field.

The input text field's initial value is taken from the ActionScript variable specified by the textVarN. For example, suppose you have two dynamic text fields on the stage named cityField and stateField which contain, respectively, the values "San Francsico" and "CA". When executed, the following (abbreviated) code would display two input text fields whose initial values are, respectively, "San Francisco" and "CA".

```
fscommand("EditText",
  "…static1=CITY&textvar1=cityField&static2=STATE&textvar2=stateFiel
  d…");
```

- **staticN:** The label to display for the Nth input text field, where N is between 1 and 3.
- **maxnumcharsN:** The maximum number of characters that the user can enter in the Nth input text field, where N is between 1 and 3.
- **inputmethodN:** The initial input method for the Nth input text field in the dialog. Valid values are "caps", “case”, “num”, “word".
  - **Word:** This is the word-complete mode. In this mode, the device will suggest complete words based on the previous characters typed by the user.
  - **Case:** In this mode, the first character of the string in each text input control will be capitalized. All other characters will be in lower case.
  - **Caps:** In this mode, the user can only enter upper-case letters.
  - **Num:** In this mode, the user can only enter numeric values (0-9) using the keypad.

    NOTE The inputmethodN parameter specifies only the initial input method for the specified input text field. The user may change this input method at any time and for any character in the input field by selecting a new input method with the left soft key.

    The '*' key is disabled in all modes.
    The '#' key enters a space (" ") in the input string in all modes.
    The '0' key is supported in all modes.
    The '1' key will toggle through a list of symbol character while in one of the non-numeric input modes. The symbols in the list are . , - ' @ : ! ? / 1.
- **usepasswordN:** If this parameter is set to true, the specified input text field will mask the user's input with asterix (*) symbol. If false (the default), text input will appear unmasked.

  NOTE This parameter only affects the masking of the text in the text input dialog, not in the channel SWF. If you want the text to be similarly masked in the text field in the SWF, you must enable the 'Password' option on the corresponding text field in the SWF. See Flash help for more information about setting password options on input text fields.
- **multiLine*N***: If set to **true**, the specified input text field will allow multiple lines of text input.

  NOTE If a text input dialog box contains a multi-line input text fields, then that field can be the only one in the dialog. For example, an input dialog box can't contain both a single-line input text field and a multi-line text input field.

EditText command examples

The examples below demonstrate usage of the EditText command.

One input text field

```
fscommand("EditText", "title=Enter name&numfields=1static1=Your
  Name&textvar1=nameVar");
```

Two input text fields

The second field accepts only numeric input up to three characters long.

```
fscommand("EditText", "title=Enter name and Age&numfields=2static1=Your
  Name&textvar1=nameVar&static2=Your Age&textvar2=ageVar&maxNumChars2=3");
```

Three input text fields, one with password masking

```
fscommand("EditText", "title=Enter Info&numfields=3static1=Your
  Name&textvar1=nameVar&static2=Your
  Age&textvar2=ageVar&static3=Password&textVar3=pwdVar&usepassword3=true"):
```

One multi-line input text field with input method set to "word" by default

```
fscommand("EditText", "title=Multiline Input Field&numfields=1static1=Your
  Life Story&textvar1=storyVar"&inputmethod1=word);
```

Channel data options

As a developer you either need to create, or team with feed data developers to create, appropriate feeds for their channels. The feed data is provided via static or dynamically generated XML format from sources external to the Flash Cast 1.2 server system. Data can be provided in non-XML format to the Flash Cast 1.2 server system, but its handling requires programming Java plug-ins, which is outside the scope of this course.

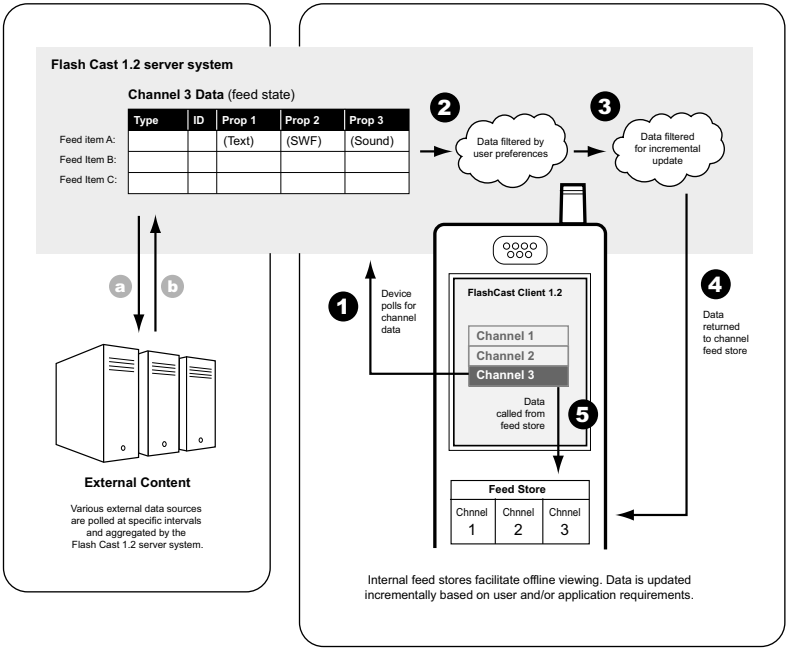
XML data can be provided in several variations:

- Flash Cast's standard feed item collection
- Non-Flash Cast, custom or legacy XML
- Rich Site Summary or Really Simple Syndication (RSS) or Atom

Flash Cast will natively handle its own standard schema. The system also provides an Extensible Stylesheet Language Transformation (XSLT) script to convert RSS and Atom feeds into the standard format. Developers will have to write their own custom XSLT scripts for the non-standard or custom formats.

The figure below represents the feed data lifecycle. Data flows from the external feed sources to the mobile handset in two independent processes:

- External data sources to the Flash Cast 1.2 server system
- Flash Cast 1.2 server system to the mobile handset



Two independent data process flows in the Flash Cast environment

Loading feed data

Flash Cast applications use the `get application` command to selectively query the feedstore for their feed data. The get command provides three parameters: type, iid, and propid. You call this command as part of a `loadVariables()` or `loadMovie()` function call.

Depending on the parameters you pass to the command, this command can return one of the following types of data:

- A feed record, or a list of feed records as a pseudo array. A feed record represents, as ActionScript variables, the properties and associated values for a single feed item.

- Image (SWF file) data that represents the property of a specific feed record. Call the get command with the `loadMovie()` function call to load SWF file data from the feedstore. You can also get a list of IDs for the feed items currently cached in the feedstore and determine when the last feed update occurred.

Flash Cast applications that do not use the Channel Development Framework use the `loadVariables()` function to load data. Using the Channel Development Framework, the elements you need to load data include:

- A movie clip into which data is loaded. By convention, the movie clip is named `data_mc`.
- Include the `load_data.as` file from the framework.

  ```
  #include "../../shared_actionscript/templates/load_data.as"
  ```

- Updating the `channel_profile.as` file with the names of the data types from the feed store:

  ```
  type_0 = "news1";
  type_1 = "news2";
  type_2 = "news3";
  total_types = 3;
  ```

  NOTE The names of the data for any given type are stored in a pseudo-array. The total number of data types is stored in a variable that is named total_types.

- A declaration for data for every view in the channel that is associated with data.

  ```
  headline_0_data_0 = type_0;
  total_headline_0_data = 1;
  ```

  NOTE The code above declares that the first headline view (headline_0) is associated with only one set of data (data_0). That data is accessed via a reference to the type_0 variable defined earlier.

About feed records

A feed record represents the properties and associated values for a specific feed item. It is analogous to a row of data in a database table. To load feed records as ActionScript variables, you call the get command as part of a loadVariables() function call. You can load a single feed record, or a list of feed records as a pseudo array.

Each feed record also contains meta information about each feed record that specifies the ID and type of the feed item whose properties are being represented in the feed record. For single feed records, this information is represented as ActionScript variables named _iid and _type. For lists of feed records, each variable has a suffix that consists of an underscore character (_) and a numeric value (for example, _iid_1 and _type_1).

Load a list of feed records

To load a list of feed records of a given type, specify a value for the get command's type parameter. For example, the following code loads the feed records for all the ticker feed items into the movie clip instance named stockData:

```
loadVariables("mydata://get?type=ticker", "/stockData");
```

Suppose that the feedstore contains three ticker feed items, and each item contains two properties, currentPrice and percentChange. The preceding code example would cause the following ActionScript variables (and sample values) to load onto the timeline of the stockData movie clip:

```
// Feed record 1
/stockData:currentPrice1="20.22"
/stockData:percentChange1="+0.25"
/stockData:_iid_1="ADBE";
/stockData:_type_1="ticker";
// Feed record 2
/stockData:currentPrice2="52.22"
/stockData:percentChange2="-0.64"
/stockData:_iid_2="XYZ";
/stockData:_type_2="ticker";
// Feed record 3
/stockData:currentPrice3="13.22"
/stockData:percentChange3="+1.75"
/stockData:_iid_3="ZZZ"
/stockData:_type_2="ticker";
// Number of feed records loaded
/stockData:get_loaded = 3;
```

The Flash Cast client automatically adds the _iid and _type variables to each feed record.

Load an image or SWF file

To load a SWF file from the feedstore, specify values for all three parameters to the get command: type, iid, and propid. In this case, the get command returns the binary value that the feed item property contains that the propid parameter specifies. For example, the following code loads the image or SWF file that the property named logo contains, which the tick feed item (feed item identifier ADBE) contains. The SWF file that the get command returns is loaded into the movie clip target named companyLogo.

```
loadMovie("mydata://get?type=tick&iid=ADBE&propid=logo", "/companyLogo");
```

You can only load image data when you use the get command in this way. That is, you can't load the text value for a specific feed-record property.

Load feed item identifiers

To get a list of the all the feed item IDs currently cached in the feedstore, use the getids command. This command takes a single parameter type, which specifies the type of feed item whose IDs you want returned. The getids command returns two variables: iids, a space-delimited list of item IDs; and getids_loaded, the number of items returned in the list of IDs. For example, the following code loads the item IDs for the stock ticker feed (abbreviated as "tick" in the following example) item currently cached in the feedstore into the stockIDs movie clip:

```
loadVariables("mydata://getids?type=tick", "/stockIDs");
```

Suppose that, in the preceding example, three feed items of type tick are present and that the ID for each tick feed item is the stock's ticker symbol (such as ADBE, APPL, and QQQ). The preceding code would create the following ActionScript variables on the timeline of the stockIDs movie clip:

```
/stockIDs:iids=ADBE APPL QQQ
/stockIDs:getids_loaded=3;
```

To make the list of space-delimited IDs easier to use and manage with ActionScript, convert it to a pseudo array. After you have a list of feed item IDs in a pseudo array you can, for example, loop over the number of items in the array to load individual feed records from the feedstore.

Reviewing pseudo-arrays

Pseudo-arrays are nothing more than variables named with sequential numbers. Item properties are automatically numbered by Flash Cast when data is returned for ActionScript use:

```
movieTitle0
movieTitle1
movieTitle2
movieTitle3
```

Storing reusable methods centrally

All of the code that we have used in this course that has been referenced via the `call()` method has been stored and referenced locally to a movie clip. If you have reusable code, you can place that code into the `framework calls_mc` movie clip.

Scrolling text

In Flash Lite 1.1, text fields cannot be given instance names and therefore cannot be directly referenced via ActionScript.

You can only reference a limited amount of information about a text field through its `Var` property.

Reviewing text field properties

Important review notes about the `Var` property:

- Populate a text field in ActionScript by setting the variable, with the same name as the `Var` property, to a string value.

  ```
  content = "BBC News";
  ```

- The `maxscroll` property of the variable tells you how many times you need to scroll beyond what's visible in order to see the last line of the text.

  ```
  numberOfTimesToScroll = content.maxscroll;
  ```

  NOTE The maxscroll value is not available until the frame following the population of the text field Var property.

- The `scroll` property holds the value of how many times you have scrolled.

  ```
  timesHaveScrolled = content.scroll;
  ```

- You can use the `scroll` and `maxscroll` properties together to scroll the text field.

  ```
  if (content.scroll < content.maxscroll) {
  // increment the scrolling amount
  content.scroll++;
  }
  ```

Reviewing looping

Use the `for` loop to iterate code a specified number of times. First declare the initial value of the iterant, then declare its upper looping limit. Finally, increment the iterant to continue the loop:

```
numberOfMovies = 10;
for (i = 1; i < numberOfMovies; i++)
{
trace("this is iteration number: " add i);
}
```

Loading binary data

Image or SWF data is maintained in the feed store as binary data. Consider the following XML data that contains a property with a reference to an image URL.

```
<data>
   <type name=“dailyMovies”>
      <item id=“movie_1”>
        <prop name="movieTitle">Behind the Mainframe</prop>
        <prop name="startTime">12:15pm</prop>
        <prop name="endTime">2:00pm</prop>
        <prop name="image"
         url="http://www.adobe.com/movie1.jpg"
         mimeType="image/jpeg"
         maxHeight="105"
         maxWidth="144" />
      </item>
   </type>
</data>
```

Some notes about the XML data above:

- Add a `mimeType` to specify the type of content.
- Specify a `maxHeight`, `maxWidth` or both to have Flash Cast proportionately resize image data.

  NOTE If you omit the mimeType, Flash Cast will attempt to guess.Load binary data by:

- Using the `loadMovie()` function.
- Using the `get` command.
- Passing the the `type`, `iid` and `propid` name-value pairs in the `get` command.
- Placing the binary data in a movie clip.

```
loadMovie(“mydata://
get?type=dailyMovies&iid=movie_1&propid=moviePic
”, “movies_mc”)
```

The code above loads the first movie’s moviePic image into the movies_mc movie clip.

NOTE loadMovie() will only load binary data. Load the text data using loadVariables() into a separate movie clip.

Adapting a channel for user preferences

This section explains how to create default user preferences and save user-defined preferences. It also explains how to check for data updates.

Understanding Preferences

One of the most powerful features of Flash Cast is the ability for users to set custom filters, or preferences, for the data that is returned to their mobile handsets from the feed server.

Key preference concepts:

- Preferences act to filter, on a per-user basis, a large data set maintained on the Flash Cast server into a subset of data that is of interest to the user.
- Preferences can filter by feed item type, item ID (within a type), or property name (within a type).
- A channel's default preferences determine the initial subset of data that's delivered to a user's device when they first subscribe to a channel.
- A channel can modify a user's channel preferences using the `setfilter` command.

Important notes about channel preferences:

- Not all channels have preferences.
  - **Preference-sensitive channels** have user-defined preferences.
  - **Preference-insensitive channels** do not have user-defined preferences.
  - Preference data is known as **feed preferences** or **preference records**.
- User preferences are stored on a secure Flash Cast server.
  - These preferences are persistent.
  - If users lose their mobile handsets, their preferences are not lost.
  - Every time a user updates their preferences, the changes are uploaded to the server during the next synchronization.
- The Flash Cast server aggregates the combined preferences for all channel subscribers.
  - The default preferences determine the initial subset of data that's delivered to a user's device when they first subscribe to a channel.
  - This makes external data calls more efficient because all data is polled at the same time, rather than for each individual user.
- Channels can be assigned default preferences.
  - Allows developers to choose the most common filters by default.
  - Ensures that the client feed store is not overwhelmed with data that exceeds the channel budget.

Defining default preferences

A channel's default preferences determines the initial subset of data that's delivered to a user's device when they first subscribe to a channel. For instance, the default preferences for a weather channel might specify the locations that should be displayed when the user first subscribes to that channel.

Adobe Mobile AppBuilder is designed with a dialog that handles what would normally go into a `.pref` file with queries to retrieve content from the data source.

Querying feed data for default preferences

A `.pref` file contains XML-formatted query statements to instruct the server to filter feed data. The following code shows the skeleton content of a `.pref` file:

```
<pref>
   <select type="news1" index="1">
</pref>
```

Some notes about the code above:

- The root node is `<pref>`.
- The `<pref>` node contains one or more `<select>` nodes.
  - Each `<select>` node contains a type attribute that specifies a feed item type to include in the channel's initial data feed.
  - Each `<select>` node represents a feed preference record, which is uniquely identified by its index attribute. You use this value in your SWF when you query the feed preference record.
  - The type argument matches the `<type>` node of the Flash Cast feed item collection schema that you would like to query.
  - The index argument of the `<select>` node defines the order in which the select statements will run.
- By default, all feed items of the specified type (and their properties) are included in the initial feed.
  - You can further filter the default data set by specifying the IDs of those feed items you wish to include.
  - You can specify which properties belonging to those feed items should be included.
- For instance, the following default preference specifies that the default data set should include the feed item of `type` *weather* whose `item ID` is *94619*.

```
<select type=”weather” index=”1”>
   <item id=”94619”/>
</select>
```

- This one specifies that only those properties named *temp* and *windspeed* should be delivered in that one feed item.

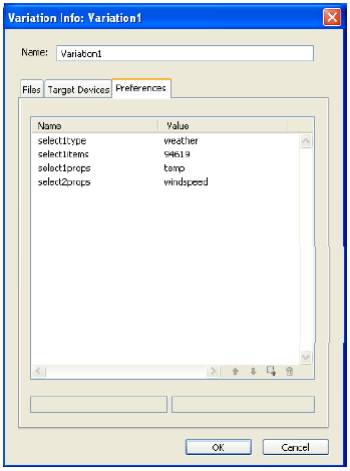
```
<select type="weather" index="1">
   <item id="94619"/>
     <prop name="temp"/>
     <prop name="windspeed"/>
</select>
```

Assigning default preferences using Mobile AppBuilder

Adobe Mobile AppBuilder allows you to set the default preferences within the emulator rather than in your channel or an external file. The **Preferences** tab in the **Variation Info** dialog takes the place of creating a `.pref` file.

The **Preferences** tab is accessed by double-clicking on the variation in the **Channel Variations** window or by selecting **Project > Variation > Preferences**. In the dialog there is an unlimited ammount of space for which the default data can be defined in the form of a list.

The information for the above `.pref` file would appear in the Mobile AppBuilder **Preferences** dialog as shown in the following figure:



Defining default preferenes in Mobile AppBuilder

Loading preferences

Use the `getfilter` command in the `loadVariables()` function to return all the preferences data to an offstage empty data movie clip.

```
loadVariables("mydata://getfilter?id=1", "/prefs_mc");
```

NOTE The id parameter is not the item id from the feed store, but rather maps to the index value of the select node of the preferences file.

In regular channels, the feed type is specified in the channel profile by the developer. In preference-sensitive channels, the user-specified feed type is defined by the mobile handset user and must be provided to the channel dynamically at run time. It will be returned from the `getfilter` command.

From a programmatic perspective, you will retrieve the feed data type from the preferences data movie clip and assign it as the `type_0` variable upon channel initialization.

```
/:type_0 = /prefs_mc:type;
```

Saving User Preferences

Use the `setfilter` command inside the `loadVariables()` function to apply the new user preference to the channel.

The following code applies the change in preference data to the channel:

```
loadVariables("mydata://setfilter?id=1&type=" add newPref, "");
```

Some notes about setting preferences:

- Pass the `type`, and `id` parameters to the `setfilter` command.
  - The `id` parameter is not the item id from the feed store, but rather a reference to the `select` node's `index` argument in the preference file.
  - The `type` parameter is the `type` from the feed store.
- Once the preferences are sent to the Flash Cast server, new preference and feed store data is returned to the client.

Applying saving button functionality

Once the user is on the preferences screen, they should be presented with a Flash Cast menu that displays **Save** and **Cancel** buttons. Use the `fscommand()` action with the showpreferences value to change the display.

```
fscommand("showpreferences", '');
```

If the user presses **Save**, the client generates a `<PageUp>` key press event. If the user presses **Cancel**, the client generates a `<PageDown>` key press event.

Processing user preferences in a batch

Process multiple preferences in one command by stringing name/value pairs together.

```
loadVariables("mydata://
setfilter?id2=2&type2=Chicago&id3=3&type3=Denver
&id4=4", "");
```

Some notes about the code above:

- The parameter names on the string have numeric indexes which represent the placement of the element in the preference record.
- To update a record you must pass in both the `id` and the `type` parameters.
- Pass only the `id` to delete a preference record. The `id4` parameter on the string will delete the fourth preference record.

Data updates

Check for data updates

The Flash Cast client sets the _feedDataAvailable variable whenever new data for your channel has arrived from the Flash Cast server. When your channel starts, the value of the _feedDataAvailable variable is initially set to undefined. When the Flash Cast client receives a data update from the Flash Cast server, the client sets the _feedDataAvailable variable to 1. Your application can periodically check the value of this variable to see whether new data has arrived in the feedstore. After determining new data is available, the application can set the value of this variable to 0 (or any value except 1) so that it can detect the next data update. The following code example illustrates this process:

```
if(/:_feedDataAvailable <> 0 && /:_feedDataAvailable ne ""){
 /:_feedDataAvailable=0;
 call(callback_action);
}
```

gotoAndPlay(_currentFrame - 1);

Similarly, when the client downloads new data from the server, the client sets the _isDataUpdating variable from undefined to 1. Your application can periodically check the value of this variable; after determining that new data is currently being downloaded, the application can set the value of _isDataUpdating to 0 (or any value except 1) so that it can detect the next data downloading event. The Flash Cast client sets this property to 0 when the connection to the server is closed. The application polls this property to update any user interface notification.

Determine when an update last occurred

To determine when the last feed update occurred, use the getupdatetime function. This function returns a variable named feedupdatetime, which contains a string containing the time or date of the last feed update. The function takes a single parameter, format, whose value determines the format of the string that it returns. For example, the following code returns the time of the last update, if the update occurred today, or both the time and date, separated by a comma, if the update occurred before today's date:

```
loadVariables("mydata://getupdatetime?format=time[date]", "updateClip");
```

Key Events and Commands 7

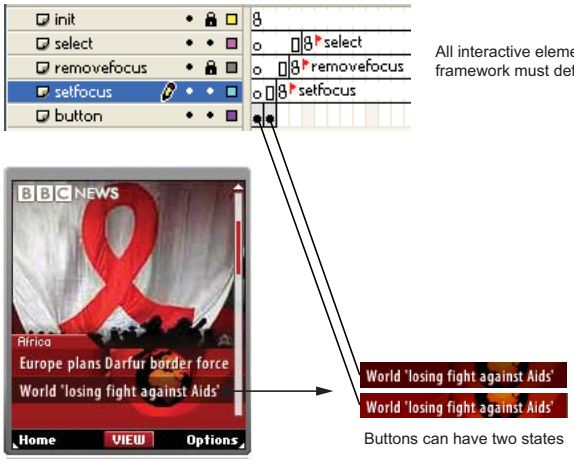
This chapter describes basic key functionality plus following tasks and commands, including:

- "Navigation between views"
- "Creating button functionality"
- "Supported ActionScript key and button events"
- "Creating visible/invisible elements"
- "Handling key press events"
- "Managing soft key state"
- "Send clear key event in Dashboard"
- "Show animation"

Navigation between views

The Flash Cast Channel Development Framework automatically handles navigation between views.

Creating View Elements or Buttons



Headlines view

Creating button functionality

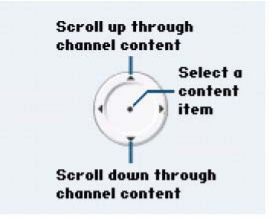
Within a view, the framework also handles navigation between elements, but it requires you to create a button movie clip with:

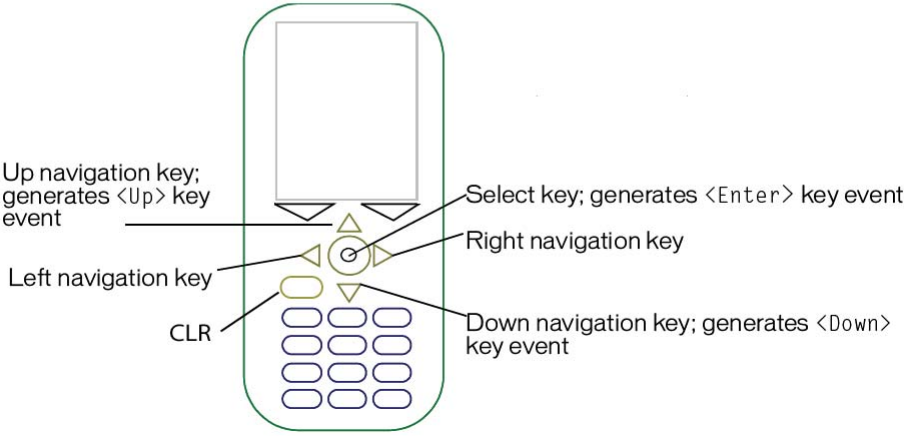
- A `stop` action on frame 1.
- Two visual states on frames 1 and 2.
- Three named frames in the movie clip:
  - `setfocus` - Move the playhead to frame 2
  - `removefocus` - Move the playhead to frame 1
  - `select` - Any code you want to run when the user selects the button

NOTE By convention, any code on a frame with only one line (or complex code), is left in the frame. Otherwise, code is placed in a subdirectory of the channel's actionscript directory and included in the frame using the #include command.

Supported ActionScript key and button events

The Flash Cast client generates ActionScript key and button press events in response to the user pressing keys on the mobile phone's keypad. Some of these keys—such as the mobile phone's left and right navigation keys, and left and right soft keys— are reserved by the Flash Cast client to let users navigate between channels, and access the Flash Cast system menus. Under certain circumstances, the Flash Cast client makes these reserved keys temporarily available to a Flash Cast application.





The following keys are always available to Flash Cast applications:

- The number keys (0 to 9).
- The star (*) and pound (#) keys.
- The up, down, and select keys on the mobile phone's five-way navigational keypad. (These keys generate <Up>, <Down>, and <Enter> key-press events, respectively.)

The Flash Cast client reserves the following keys, except where noted:

Mobile phone's left and right navigation keys. These keys may, depending on the platform, let users navigate between adjacent channels.

Creating visible/invisible elements

The Channel Development Framework for Flash Cast has been created so that a developer can easily incorporate non-visible components.

For example, you can make a button invisible by default using the following code:

```
button_0._visible = false;
```

`button_0` represents the instance name for that particular button and `._visible = false` sets the visible property of the button to false, or invisible.

In the example of the Barebones channel, the visible/invisible toggle functionality of the buttons is handled by the `enabledisable.as` file located in the `calls_mc` movie clip.

The `enabledisable.as` file contains the following code:

```
// show / hide the two middle buttons, making them selectable or not

if(eval(/
framework_mc:selected_view).button_1._visible == false)

{

eval(/
framework_mc:selected_view).button_1._visible = true;

eval(/
framework_mc:selected_view).button_2._visible = true;

}

else

{
eval(/
framework_mc:selected_view).button_1._visible = false;
eval(/
framework_mc:selected_view).button_2._visible = false;
}
```

Handling key press events

In Flash Lite applications, you handle user key press events by creating a button with an on(keyPress) statement as shown below:

```
on (keyPress “<Down>”)
{
// do something
}
```

Framework key handlers

The Flash Cast Channel Development Framework provides keyhandlers that know which view they are in when called. The Flash Cast key handlers are:

Framework key handler	Key that triggers handler
enter_keyHandler	select (center) key
up_keyHandler	up arrow
down_keyHandler	down arrow
right_keyHandler	right arrow
left_keyHandler	left arrow
pageUp_keyHandler	right soft key
pageDown_keyHandler	left soft key
backSpace_keyHandler	CLR key
n0_keyHandler	0
n1_keyHandler	1
n2_keyHandler	2
n3_keyHandler	3
n4_keyHandler	4
n5_keyHandler	5
n6_keyHandler	6
n7_keyHandler	7
n8_keyHandler	8
n9_keyHandler	9

Key handlers are set with a value that references to a named frame. The following code defines key handler code located in the `scroll_up` frame of the current movie clip.

```
up_KeyHandler = _target add ":scroll_up"
```

NOTE Remember that _target represents the current movie clip. In other programming languages, _target is the equivalent of the keyword "this".

Using the framework back functionality

The Flash Cast framework manages user history and exposes the ability to return to the last view the user was reading. To move the user back, set the appropriate key handler to `/framework_mc:back`.

```
enter_KeyHandler = "/framework_mc:back";
```

Managing soft key state

When navigating within a channel, the channel SWF handles the soft-key actions. Since the actions of these keys are determined by the SWF, it needs to set the text labels of the soft keys.

For example, in the Channel Guide, when viewing the Subscription Summary pane (below, left), pressing the center soft key (CSK) displays help text. The CSK needs to be labeled "INFO" or a similar value. In the same channel, when viewing the channel lists (below, right) the CSK opens a subscribe or unsubscribe dialog, in this case the CSK needs to be labeled "OK".

The soft-key text is controlled by the currently loaded FCC .swf and so the desired soft-key state must be sent by the channel .swf to the client using the SetSoftKeyState fs-command.

FS-command: SetSoftKeyState

This section describes the signature, functionality and usage of the soft key management function:

```
fscommand "SetSoftKeyState", "sftkeyconfig=<value>" )
```

Usage

```
fscommand( "SetSoftKeyState", "sftkeyconfig=<label>");
```

Soft key label mapping

The following table describes, for every configuration, which labels will be assigned to each soft key.

Label	Left Soft Key	Center Soft Key	Right Soft Key
Ok_Options		OK	Options
Done_Options		DONE	Options
Info_Options		INFO	Options
Home_Info_Options	HOME	INFO	Options
Home_Ok_Options	HOME	OK	Options
View_Options		VIEW	Options
Save_Cancel		SAVE	Cancel

Label	Left Soft Key	Center Soft Key	Right Soft Key
OK		OK	
Done		Done	
Home_View_Options	HOME	VIEW	OPTIONS
Edit_Options		EDIT	Options
Home_Options	HOME		Options
Home_Edit_Options	HOME	EDIT	Options
Home_Open_Options	HOME	OPEN	Options
Home_Close_Options	HOME	CLOSE	Options

Soft key behavior

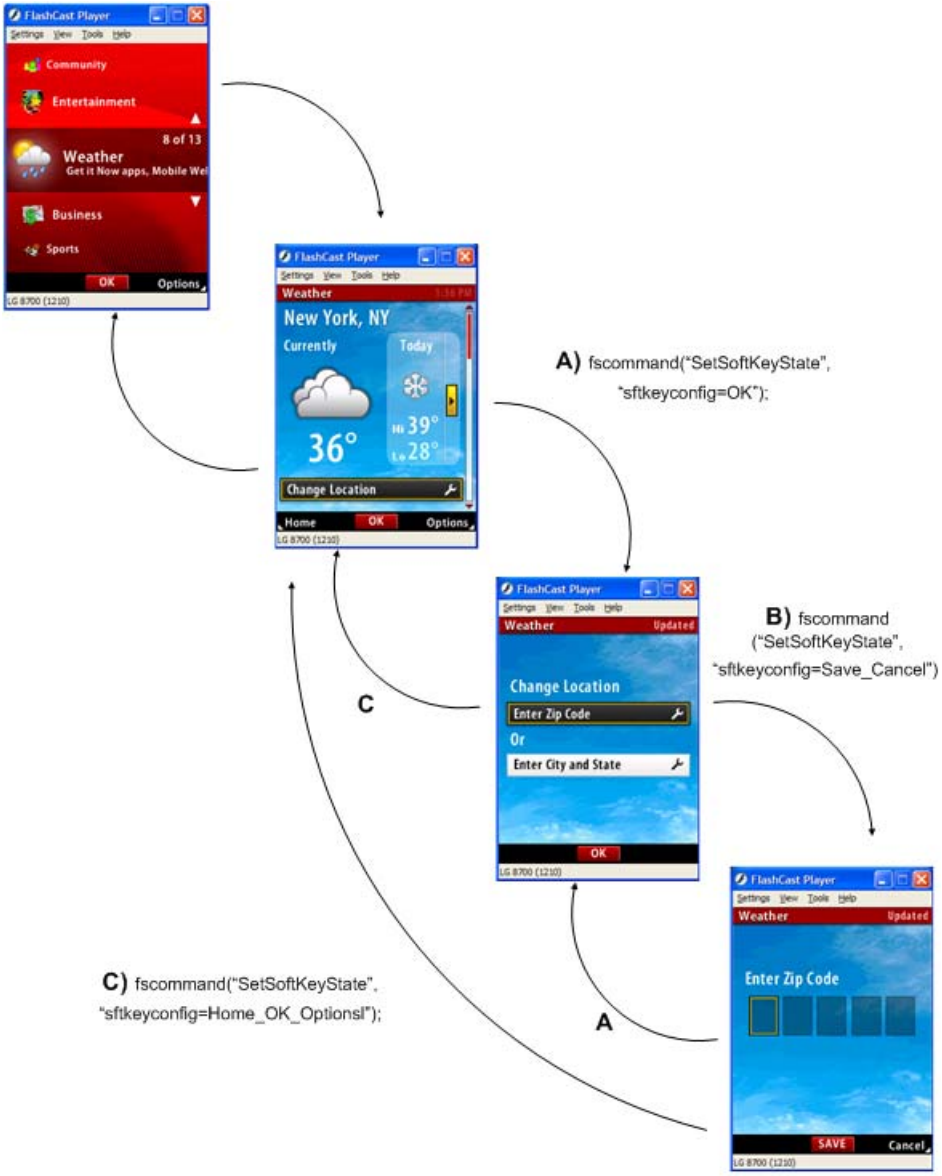
The following table describes, for every configuration, the event or behavior of each soft key.

Label	Left Soft Key	Center Soft Key	Right Soft Key	Clear Key (CLR)
Done	N/A	<“Enter”>	N/A	Exit Channel
OK	N/A	<“Enter”>	N/A	Exit Channel
Save_Cancel	N/A	<“Enter”>	<“PageDown”>	Exit Channel
Done_Options	N/A	<“Enter”>	Open Client’s Option’s Dialog	Exit Channel
Edit_Options	N/A	<“Enter”>	Open Client’s Option’s Dialog	Exit Channel
Ok_Options	N/A	<“Enter”>	Open Client’s Option’s Dialog	Exit Channel

Label	Left Soft Key	Center Soft Key	Right Soft Key	Clear Key (CLR)
Info_Options	N/A	<“Enter”>	Open Client’s Option’s Dialog	Exit Channel
View_Options	N/A	<“Enter”>	Open Client’s Option’s Dialog	Exit Channel
Home_Options	Exit Channel	N/A	Open Client’s Option’s Dialog	Exit Channel
Home_Close_Options	Exit Channel	<“Enter”>	Open Client’s Option’s Dialog	Exit Channel
Home_Edit_Options	Exit Channel	<“Enter”>	Open Client’s Option’s Dialog	Exit Channel
Home_Info_Options	Exit Channel	<“Enter”>	Open Client’s Option’s Dialog	Exit Channel
Home_Ok_Options	Exit Channel	<“Enter”>	Open Client’s Option’s Dialog	Exit Channel
Home_Open_Options	Exit Channel	<“Enter”>	Open Client’s Option’s Dialog	Exit Channel
Home_View_Options	Exit Channel	<“Enter”>	Open Client’s Option’s Dialog	Exit Channel

Example use case

The example below demonstrates how you can use the `SetSoftKeyState` fscommand to manage the soft key labels and actions.



Send clear key event in Dashboard

Pressing the CLR key on the device takes the user back one step in the views. On Verizon Wireless devices, regardless of content or application, the CLR key is meant to act as a "back" button. That is, if the user presses CLR the application should always return them to its previous view or screen. If the application is at its top level already, the CLR key should act as a "quit" or "exit" button.

NOTE Screen-shots in this document are to illustrate points made in the text, and they should not be taken as descriptive of final product UI or graphics.

In the example above the client handles the first transition: opening the Manage Channels .swf and so it can handle the CLR key which simply exits the channel. However the second transition: opening the Subscription Summary occurs within the channel .swf itself and the Flash Cast Client has no knowledge of the transition. If the CLR key event was now handled by the Client then would simply exit the channel, breaking the UI guideline.

For the CLR key to be handled correctly it needs to be managed by the channel .swf. The FS-command “SendCLRKeyEventToSWF” is used. In the example, when the second transition occurs (Subscription Summary is opened) the Channel Guide .swf used the FS-command SendCLRKeyEventToSWF to signal to the client that it will handle future CLR events. On receiving and CLR key events the Client will take no action other than passing the command on to the channel .swf. If the Channel Guide .swf subsequently returns to the Manage Channels (center) screen then it will signal to the client that the client should again handle CLR key events. A CLR key event will now exit the Channel Guide.

By default, the Dashboard client handles all CLR key presses, according to the following rules:

- If a user presses CLR while viewing a top-level channel, the Dashboard client returns to the Now Playing (i.e., "Home") screen.
- If a user presses CLR while viewing the Now Playing screen, the client exits altogether. This is typical BREW application behavior on Verizon Wireless devices.
- In neither case can a channel detect the CLR key press event, as it is handled natively by the Dashboard client.

If a channel wishes to use the CLR key for intra-channel navigation—for example, as a "back" button to return the user to a previous screen—it must first request the CLR key event from the client using the SendCLRKeyEventToSWF fscommand. This command takes a parameter that, when set to true, causes the Dashboard client to generate a <backspace> ActionScript key press event for each CLR key press. Calling the same command with the send parameter to false relinquishes control of the CLR key back to the Dashboard client and its default CLR key handling behavior (as described previously).

IMPORTANT A channel must always relinquish control of the CLR key back to the Dashboard client once the channel no longer needs it. Otherwise, the channel effectively breaks (if only temporarily) the Dashboard client's default CLR key handling behavior.

FS-command: SendCLRKeyEventToSWF

Arguments:

- **send:** Boolean: “true” or “false”, unrecognized values are taken as “false”. When “true” CLR key events are sent to the channel .swf; when false they are handled by the Flash Cast Client and no event is sent to the channel .swf

```
fscommand("SendCLRKeyEventToSWF", "send=true");
fscommand("SendCLRKeyEventToSWF", "send=false");
```

NOTE By default, CLR key events are handled by the Flash Cast client.The CLR key event is sent to the channel .swf as keycode FlashKey::KEY_BACKSPACE. On the Windows version of the Flash Cast client, the CLR key is simulated by pressing the F3 key on the keyboard.

Examples

The following examples demonstrate usage of the SendCLRKeyEventToSWF command. In this example, the channel temporarily requests the CLR key from the client so it can use the CLR key to let the user return to the main menu from a secondary screen.

```
on(keyPress
fscommand("SendCLRKeyEventToSWF", "send=true");
```

After calling this command as shown, the client no longer handles CLR key presses and also generates a <backspace> key press event that the channel can detect and respond to, as shown below:

```
on(keyPress "<Backspace">) {
  // Handle CLR key press.
  previous()// eg, return to previous screen/frame
```

The channel will continue to receive <backspace> key press events until it calls `SendCLRKeyEventToSWF` fscommand with its `send` parameter to `false`, as follows:

```
Fscommand("SendCLRKeyEventToSWF", "send=false");
```

IMPORTANT In some cases, a channel may use the CLR key for navigation within the channel itself. For instance, suppose a channel developer wanted to use the CLR key to let users "go back" a screen within the channel's navigation.

Show animation

When a channel is added or removed from the channel guide, user feedback is provided as an animation. Example:

Since subscription and unsubscription is handled by the Flash Cast Client it needs to send a signal to the soft key .SWF (fc01) to run the animation. If the Subscribe dialog is dismissed (with the CLR key), then the animation is not shown.

The variable “fcc_showAnim” is used for this. When the dialog is exited the soft-key .swf fc01 is reloaded and fcc_showAnim is set by the client to have the value “true” if the action was confirmed, or it will be undefined if the action was cancelled. When fc01 is run is should check the value of fcc_showAnim and if it is true it should display the animation, if it is undefined or has any other value then it should continue without displaying the animation. The fc01.swf should then reset the variable so the animation will not be played again until set by the Flash Cast client.

NOTE During processing of the dialog, the future value of fcc_ShowAnim is stored internally in the Flash Cast player in the Boolean variable FCCUserInterface::bShowAnim. The client must also set the variable “fcc_changesSavedTitle” with the string to be displayed in the animation (in the above example this is “Changes Saved”)

Structure & Security Considerations

8

This chapter provides application parameters, including:

- "About the feed item collection format"
- "Channel budget"
- "Content structure"
- "Sandbox considerations"

More detailed information is available in *Developing Flash Applications*.

About the feed item collection format

The Feed Item Collection Schema defines the "native" XML format for feed item collections. The standard feed module included with the Flash Cast server can directly consume XML in this format. The feed module can also be configured to use a custom XSLT file to transform XML in other formats into the native format. Refer to the Developing Flash Cast Applications Guide (Adobe) for sample XML formats.

The XML defines a collection of three feed items of type directory. Each feed item of this type is assigned a unique ID, and contains four properties: first, last, title, and image. The values for some of these properties (first, last, and title) are provided as text in the XML document itself, while the value of the image property is specified remotely by URL.

Feed items of the same type are analogous to rows of data in the same database table. Typically, but not necessarily, feed items of the same type also contain the same property names. This is similar to how rows in the same database table are constrained to have the same field.

For a complete description of the elements and attributes of the Feed Item Collection Schema, see "Content structure" on page 101.

Reviewing XML Syntax

As experienced developers, you are already familiar with Extensible Markup Language (XML) syntax. Remember that XML is a general-purpose, tag-based hierarchy of meaningful data.

The following XML code is generic XML data that describes movie times for a local fictional movie theater's Film Festival:

```
<festival>
   <festivalDetails>
      <festivalName>LOL Festival</festivalName>
      <festivalDesc>A bunch of classic technology
funny movies that will definitely make you laugh
out loud.</festivalDesc>
      <startDate>June 7th</startDate>
      <endDate>June 16th</endDate>
</festivalDetails>
<dailyMovies>
   <movie id="movie_1">
      <movieTitle>Behind the Mainframe</movieTitle>
      <startTime>12:15pm</startTime>
      <endTime>2:00pm</endTime>
   </movie>
   <movie id="movie_2">
      <movieTitle>Three Wireless Mice</movieTitle>
      <startTime>2:30pm</startTime>
      <endTime>4:35pm</endTime>
   </movie>
   </dailyMovies>
</festival>
```

The `<festival>` tag is referred to as the root node. Each additional tag pair is called a node, or element, and further describes the data. Notice that the `festivalDetails` node and the `dailyMovies` nodes are two child nodes at the same level.

Understanding the Flash Cast XML nodes

The Flash Cast XML format has specific format rules:

- The root node is always `<data>`.
  - In a database analogy, the `<data>` node is your database.
  - All the information in the `<data>` node is known as a feed source.
- The <type> node defines the type of data within this root element.
  - In a database analogy, the <type> node is a table.
  - There can be one or more <type> nodes within a <data> node.
  - Each <type> node contains a name attribute, that describes the data.
- The <item> node defines individual elements within each <type> node.
  - In a database analogy, the <item> node is a row.
  - The <item> node is also known as a feed record.

- There can be one or more <item> nodes within a <type> node.
- Each <item> node contains an id attribute that uniquely identifies this particular item in the type.
- The `id` value is usually a sequentially named variable that will be used as a pseudo-array in ActionScript.

- The `<prop>` node defines all the properties about any particular item.
  - In a database analogy, the `<prop>` node is a column.
  - There can be one or more `<prop>` nodes within a `<item>` node.
  - Each `<prop>` node contains a name attribute that describes the particular property value.
  - Property data can be non-asset data (text) or asset data (images or SWF). The latter is stored in binary format.

The following Film Festival XML data conforms to the Flash Cast feed item collection schema and describes the same information as the previous generic XML example.

```
<data>
   <type name="festival">
      <item id="festivalDetails">
        <prop name="festivalName">LOL Festival</prop>
        <prop name="festivalDesc">A bunch of classic technology funny
          movies that will definitely make you laugh out loud.</prop>
        <prop name="startDate">June 7th</prop>
        <prop name="endDate">June 16th</prop>
      </item>
   </type>
   <type name="dailyMovies">
      <item id="movie_1">
        <prop name="movieTitle">Behind the Mainframe</prop>
        <prop name="startTime">12:15pm</prop>
        <prop name="endTime">2:00pm</prop>
      </item>
      <item id="movie_2">
        <prop name="movieTitle">Three Wireless Mice</prop>
        <prop name="startTime">2:30pm</prop>
        <prop name="endTime">4:35pm</prop>
      </item>
      <item id="movie_3">
        <prop name="movieTitle">Skip Intro</prop>
        <prop name="startTime">5:05pm</prop>
        <prop name="endTime">6:55pm</prop>
      </item>
      <item id="movie_4">
        <prop name="movieTitle">Wacky Wacky Web</prop>
        <prop name="startTime">7:25pm</prop>
        <prop name="endTime">9:30pm</prop>
```

```
      </item>
   </type>
</data>
```

Feedstore space limitations

As mentioned in Chapter 2, each Dashboard content channel is limited to a preset amount space in the client's feedstore. This value is customizable per device, but all channels on a given device are subject to the same limit. All of the channel's components together must total equal the limit, or be less than this limit, in order to be a valid Flash Cast channel. This includes channel data, the channel SWF, channel icon, images, user preference data, and the channel scratchpad. Any channel which uses more than this amount of data will not display correctly. This limit ensures that one poorly designed channel does not use more than its share of Flash Cast's total memory allotment on the device. Channels limints are imposed to allow for the best combination of richness and small footprint, both in terms of bandwidth and runtime memory usage.

Channel budget

All mobile applications require that you pay particular attention to application file size as well as device memory consumption. Flash Cast developers must also be cognizant of even more stringent requirements imposed by the service provider.

For Verizon Wireless Dashboard development, the channel budget limits (for QCIF-sized handsets) are:

- All channel assets must remain below the client imposed device limit of file size.
- Any ActionScript code that takes longer than 15 seconds to run will be stopped by the Flash Cast client.

Channel budget limits are imposed for each channel on a handset. The Flash Cast client will present the user with an error message if they exceed the limits.

The per channel disk limits per device family are as follows:

- QCIF Portrait (203 x 176) - 133k
- QVGA Portrait (320 x 240) - 133k
- WQVGA Landscape (240 x 400) - 180k
- QVGA Landscape (240 x 320) - 180k

Channel limits

In general, it's safe to assume that more than half of the overall space requirements of a channel will be consumed by the channel SWF and other visual assets. Thus, the data feed for channels is in effect limited to 50k to 80k. Any channel requiring more than this amount of data is likely to encounter problems running within Flash Cast.

For most channels this will be more than ample, as a user can specify the subset of the data he or she is interested in through the use of channel preferences. For instance, a weather channel doesn't need to download all weather reports for all locations to a user's device. The user can choose which zip codes are of interest in the channel preferences and the channel only downloads data for those zip codes.

In cases where channel data threatens to exceed the allotted space, some tactics can be used to keep data within the allowed size. For instance, news and sports channels might decide to impose a five-paragraph limit to stories. This will prevent an individual story from using up the channel's allotted space in the feedstore. In this case, a paragraph limit is more useful and practical than a hard character limit because it prevents mid-sentence or mid-word truncation which would be a poor user experience. Content providers should also try to eliminate any redundant or unneeded data wherever possible. If no way can be found to keep the channel content smaller than the limit, the channel should be redesigned to require less content.

For instance, a channel with a data feed of 2KB, a SWF of 80KB, an icon of 1KB, scratchpad and/or settings data of 1KB and content-specific assets of 41KB should display correctly (assuming all components are formatted correctly) because the components, when totaled, use 125KB. This, of course, will work on any of the supported devices.

Preference-sensitive Feed data:	2KB
Channel swf:	80KB
Icon:	1KB
Scratchpad/settings:	1KB
Content/assets	41KB
Total	125KB

However, if the content specific images were 50KB in size instead of 41KB, the channel would not load because it exceeds the 133KB limit.

Preference-sensitive Feed data:	2KB
Channel swf:	80KB
Icon:	1KB
Scratchpad/settings:	5KB
Content/assets	50KB
Total	138KB

Reviewing best practices for mobile optimization

The following are some general mobile best practices that will improve your file size or memory usage:

- Vector and bitmap images can have a large impact on memory usage. To best determine which type of image to use, test both ways if appropriate.
- For embedded bitmap images, sometimes using lossless compression can use less memory because there is no need to uncompress the file at runtime.
- Decrease the complexity and curves of vector graphics.
- Use shapes rather than symbols in Flash assets.
- Use vector gradients and animations sparingly.
- Keep animation elements simple.
- Use bitmap text, rather than vector text, for better performance.

Consider the following best practices that are specific to Flash Cast development:

- Do not use embedded fonts.
- Impose limits on the number of characters or paragraphs in text data.
- Implement default preferences and user preferences to filter the amount of data returned to each user's handset.

Two techniques for testing channel performance are:

- Test your channel on a real device.
- Iterate over elements you would like to optimize in a separate file to test various techniques. Only move them into your channel once you are satisfied that you have the best performance and file size.

Reviewing additional resources

The following two articles are recommended reading for mobile application best practices and optimization:

- www.adobe.com/devnet/devices/articles/ota_delivery_print.html
- www.adobe.com/devnet/devices/articles/optimizing_anim_print.html

Content structure

The deployment package that Adobe Mobile AppBuilder creates is a ZIP file that contains all the information and assets required to import and deploy the channel on any Flash Cast server. The ZIP file contains the following items:

- An XML descriptor file (db.xml) that describes the three components of the channel, namely: the feed object, the feedapp object, and the datasource object. (These elements are described below.)
- The channel assets, including SWF files, image icons, default preferences, and supporting files for the channel's data source, such as XSLT files for any necessary data transformations. The XML descriptor file contains references to all the assets in the ZIP file.

The feedapp, by definition is, for a complete channel, bundled with a feed and datasource definition. In Mobile AppBuilder, this information is automatically sent from the Flash Development environment to a Mobile AppBuilder development server. The format of the feedapps, datasources, and feeds is defined in the database xml document.

For deployment to a staging server, the deployment tool is used. For deployment to a production server, the standard import command is used.

The XML descriptor file (db.xml)

The XML descriptor file (named db.xml) that's exported by Mobile AppBuilder describes the three main server objects related to a channel:

- **The feed object**, represented by the <feed/> XML element. The feed object is the "lynchpin" that correlates one or more feedapp objects with zero or more datasource objects. (A complete channel definition requires at least one feedapp object, but not a datasource object). A GUID, or Globally Unique Identifier, is the key that links these parts together.

- **The feedapp object**, represented by the <feedapp/> XML element. The feedapp object contains information about the channel such as name, description and pricing information, as well the channel assets, such as SWF, JPEG, and default preferences file for each target device type and client type. These assets are contained in folders within the ZIP file.
- **The datasource object**, represented by the <datasource/> XML element. The datasource object contains information about where the data source server (DSS) acquires the channel's data, including the Java feed module responsible that does the work of acquiring that data.
- **The GUID** (mentioned above) is a unique number assigned to the application and content and maintained within the deployment environment. It can be used by a developer or system administrator to select or identify content. But, it cannot be changed by user interaction.
- Below is a discussion of a sample db.xml file generated by Mobile AppBuilder and a description of each element in the XML file.

Sample <feed> element

The <feed> element itself does not contain much information, except for a short name, start-up options, and the GUID, specified by the element's id attribute.

```
<feed id="BD1DAEA3-533A-2B9A-1135-951AB1F5FD4A" name="fc_test2"
  startup="auto"/>
```

Sample <feedapp> element

The <feedapp> element contains general information about the channel including name, description, and pricing information, and also references the channel assets (for example, SWF files) included in the ZIP package. For each targeted client type or device type an <app/> sub-element identifies the location of the channel's SWF file and image icon within the ZIP package.

Also note that the <feedapp/> element's feedId attribute specifies the same GUID of the associated feed object. This links the feedapp object to the feed object.

```
<feedapp id="174DEA4E-6040-8477-744E-1C95B01682FF"
feedId="BD1DAEA3-533A-2B9A-1135-951AB1F5FD4A"
name="fc_test2"
version="0.1"
description=""
longDescription=""
startup="auto">
```

```
<!-- The client and device type may vary based upon the actual device in
  use. -->
<app clientType="default" deviceType="default"
fca="fcas/174DEA4E-6040-8477-744E-1C95B01682FF/default_app.swf"
icon="fcas/174DEA4E-6040-8477-744E-1C95B01682FF/default_icon.jpg"/>
<app clientType="FlashCast_1.2" deviceType="BREW_176x204"
fca="fcas/174DEA4E-6040-8477-744E-1C95B01682FF/default_app.swf"
icon="fcas/174DEA4E-6040-8477-744E-1C95B01682FF/default_icon.jpg"/>
</feedapp>
```

Sample <datasource> element

The <datasource/> object contains information about the channel's data source. The className attribute specifies the path (on the Flash Cast server) to the Java class that's responsible for acquiring and processing the channel's data from an external source. In the example XML below this attribute specifies the path to the Standard Feed Module, which is included with the Flash Cast server. This feed module, developed by Adobe, provides a flexible an customizable way to acquire channel content as XML data over HTTP.

The values of the <prop/> elements specify configuration properties for the standard feed module, including paths to URLs where the feed module will acquire channel data, poll intervals, and other information. Note again the feedID attribute that specifies the GUID of the same feed object referenced by the <feedapp/> element.

```
<datasource type="3"
id="87FE082F-7C89-11DC-6513-8DFC436CD57D"
feedId="BD1DAEA3-533A-2B9A-1135-951AB1F5FD4A" name="fc_test2"
  startup="auto"
feedLog="WARN"
className="com.macromedia.feeds.core.StandardFeedModule"
  pollInterval="43200">
<!-- Specify the number of lines of logging data to keep for this data
  source -->
<prop name="memoryLogSize" value="100"/>
<!-- Do not poll unless specified by the user -->
<prop name="userDependentPolling" value="true"/>
<!-- Preferences affect the query -->
<prop name="prefSensitive" value="false"/>
<!-- See 'Developing FlashCast Applications' for these fields -->
<prop name="query.xslParam[lifetime]" value="86400"/>
<prop name="query.xslParam[maxItems]" value="10"/>
<prop name="query.xsl" value="misc/xsl/feedTran.xsl"/>
<prop name="query.path"
value="http://www.flickr.com/services/feeds/
  photos_public.gne?format=rss_200"/>
</datasource>
```

Sandbox considerations

As discussed above, the Dashboard client sandbox prevents a badly (or maliciously) designed channel from compromising the security or reliability of the Dashboard service. However, there are some things channel developers should avoid to ensure that Dashboard users have the best possible experience.

- **Feedstore overruns**—Each Flash Cast channel is allotted 133KB of data storage space in the client's feedstore. All the channel assets (SWF file, channel icon) and data (including text and image assets) must fit within this allotment, or else the user will see an error dialog when they try to view the channel. The Flash Cast emulator in the Flash CS3 authoring tool contains a feedstore meter that displays the feedstore usage of your channel at any one time.
- **Runtime memory overruns**—In addition to the feedstore limits discussed above, each channel also has a runtime memory limit of 1.1MB. If a channel goes over this limit, the Dashboard client displays an error dialog to the user. The Flash Cast emulator in the Flash CS3 authoring tool contains a memory meter that displays the total memory usage of your channel at any one time.
- **Script timeout**—If a channel contains ActionScript that takes longer than 15 seconds to execute, the client will unload the channel and display an error dialog in its place. Therefore, it's important that developers avoid writing scripts that may take a long time to execute. Developers should always test their channel code on an actual handset, and not just in the desktop Flash Cast emulator, as the processing speed of the handset is much slower than desktop computers. Actionscripts are run on cached content only. There is no network access by SWFs.

Flash Cast Embedded Fonts 9

Working with fonts in the mobile environment requires trade-offs between consistent display and file size. Flash Cast channels follow standard font usage rules. The Flash Cast client provides you with additional benefits using specialized embedded fonts.

Device fonts

Some important review notes about using device fonts:

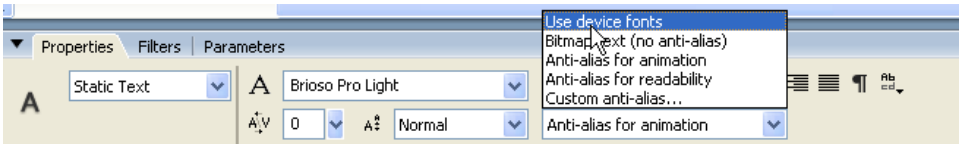
- Device fonts will not increase the application file size because they are not embedded, rather, they rely on the device to render an appropriate font.
- You cannot control the consistency of how device fonts will display on different devices. Implement a device font using one of these methods:
  - Select **_sans**, **_serif,** or **_typewriter** from the **Flash Fonts** drop-down list.
  - Select **Use device fonts** from the **Font rendering method** drop-down list.
- If the device does not have the font you've chosen, it will choose a replacement font.
- Device fonts are not anti-aliased and so usually appear clearer at smaller font sizes.

Embedded fonts

Unlike raster (vector) fonts, device fonts usually don't scale well and should not be scaled. Device fonts are less complex and require less runtime memory to render when compared to raster (vector) fonts. If you are planning to display large amounts of text and Dashboard's runtime memory limits are becoming an issue, consider using device fonts. They will reduce memory usage of your channel. Some important review notes about using embedded fonts:

- Embedded fonts are fonts that have one or more character outlines embedded in the Flash movie.
- Embedded fonts will look consistent across devices, but they increase the file size of the movie.

- Flash will automatically embed font outlines for static text fields that are not using one of the device fonts.
  - Select **uses device fonts** from the **Font rendering method** drop-down list to prevent font embedding.



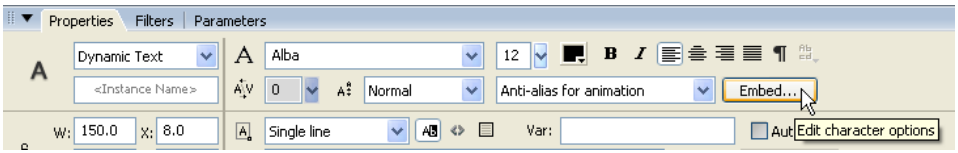
Force a static text field to use device fonts instead of default embedded fonts

Additional considerations:

- If **use device fonts** is selected and the font is available on the user's device, then it will be displayed. Otherwise, a similar font on the device will be substituted.

  NOTE Anti-alias for readability is not available for Flash Lite 1.1 applications.

- Dynamic and input text fields do not automatically embed fonts.
  - If **use device fonts** is selected and the font is available on the user's device, then it will be displayed. Otherwise, a similar font on the device will be substituted.
  - If you select **Bitmap text (no anti-aliasing)** or **Anti-alias for animation** from the **Font rendering method** drop-down list, you must use the **Embed** button and it's related dialog box to select the characters you would like embedded.



Use the Embed button to embed font outlines for dynamic text fields

NOTE If you would like to embed extended characters for localization or other reasons, remember that there is a file size cost that impacts your channel budget. For a better localization option, create multiple channels that specifically target localized devices.

Flash Cast embedded fonts

The Flash Cast client provides developers with the best of both worlds by implementing specialized embedded fonts.

Important notes about Flash Cast embedded fonts:

- They do not increase file size since the fonts are provided as part of the Flash Cast client on the device.

- They provide a consistent display across devices.
- There are 6 specialized embedded fonts:
  - Myriad Pro Light Condensed - non-raster font
  - Myriad Pro Light SemiCondensed - non-raster font
  - Myriad Pro SemiCondensed - non-raster font
  - Standard 0755 - raster font
  - Standard 0765 - raster font
  - copy 10_55 - raster font
- The non-raster fonts (Myriad Pro) looks best at larger point sizes (12 points or greater)
- The raster fonts are designed to look their best as small point sizes (8 points or smaller)

Developers need to be aware of the following:

- Use Flash Cast embedded fonts whenever possible. They will provide a consistent display and have no impact on your channel budget.
- You will need the Flash Cast embedded fonts for development.
- Make sure that the **Font rendering method** is set to **Anti-alias for animation**.

Changing fonts

To change fonts:

1. Open the appropriate FLA file.
2. Select **File > Publish Settings**. In the **Publish Settings** dialog, in the **Options** menu, check the **Generate size report** box.
3. Click **Publish**. The **Output** panel appears.
4. In the **Output** panel, navigate to the **Font Name** header and notice the Arial fonts.
5. Enter the name of the movie clip and lock all the layers except the title layer.
6. Select the dynamic text box on the stage and change its font from Arial to another available font.
7. Lock the title layer and unlock the title shadow layer.
8. Select the dynamic text field on the stage and change its font to the font you selected.
9. Enter the timeline for the movie clip and lock all the layers except the category layer.
10. Publish the movie and view it in the standalone emulator to verify the font changes.

You can also change fonts for the category layer, and other elements using the same method.

Developer Submittal Guidelines

A

Channel concept submission

Developers submit a channel concept to the Zon (www.vzwdevelopers.com). Channel concepts and submissions should adhere to the Verizon Wireless Style Guide for Dashboard Channel Development. The channel concept submission should contain the following:

- Photoshop (PSD) files that illustrate the user and channel design.
- A stand-alone reference SWF file that emulates channel design and functionality.
- A descriptive walk-through of the channel from a user perspective (storyboard), including interaction model, and intended animation behaviors.

The Verizon Wireless Content and Programming team then reviews the channel concept, provides necessary feedback to the developer, and rejects or approves the channel concept.

Channel quality expectations

Upon approval of the channel concept, Verizon Wireless contacts Adobe to request that the channel developer be provided with access to the Adobe Flash Cast Authoring Program. The program provides the channel provider with the tools and resources they need to fully develop and test their channel concept.

If the channel requires content from the Ichiro Content Management System (CMS), the developer contacts the Verizon Wireless Content Administrator to obtain the Ichiro staging URL(s) for development purposes. The developer integrates the Ichiro staging URLs (and any other non-Ichiro content) into their channel's data source.

Once the developer has completed their initial testing using the Ichiro staging URL content, the developer contacts the Verizon Wireless Content Administrator to have the Ichiro content URLs promoted to the Ichiro production environment. Upon receipt of the Ichiro production URLs, the developer updates their channel's data source definition to point to those new URLs.

Once channel development and Ichiro integration is complete, the developer must validate the channel according to the following criteria:

- Overall look and feel conforms to any style guide requirements
- Channel data is being fed properly to the channel
- Channel data conforms to the delivered XSD file so the current feed and all future feeds are properly handled by the channel
- If channel lets users specify channel data preferences, confirm that preference changes result in expected data updates being delivered and displayed
- Confirm default preferences are set correctly (if required)
- Confirm all link-aways link to expected BREW application or WAP page
- Confirm ACL for the channel is unset
- Channel assets for all supported devices are available and published

Verizon Wireless or its designates will also evaluate the submitted content to ensure that it meets requirements for customer appropriateness of content, performance, and fit with their business needs. Elements they will assess include, but, are not limited to:

explicit content	unlicensed content	masquerading (channel isn't what it says it is)
branding	description	categorization (in the Add Channels screen)
image or video quality	frame rate	preference settings
spelling and grammar	content updates	style compliance
responsiveness	navigation	sound volume and quality
readability	size and cropping	marketing assets (for example for the Companion website)
scrolling	data update requirements	

Channel content elements

A specialized deployment tool is used to facilitate deployment of Dashboard channels to the Flash Cast server. The tool requires as input the following items:

- A channel deployment package (ZIP file) provided by the developer. This ZIP file contains all channel assets (SWFs, image icons) and the channel descriptor file (db.xml).

- Channel meta-data from Verizon Wireless. This includes information such as channel name and description; channel categorization; whether the channel is premium or basic; and other information.
- A policy file that defines accepted and default database values for channel deployment.

Developer submittal

The completed development package that the developer submits to Verizon Wireless should include:

1. A digitally signed channel deployment package (ZIP file) exported from the Flash CS3 authoring tool. The ZIP file should contain the following items:
   a. The XML descriptor file for the channel (db.xml)
   b. All channel assets (SWF files, channel icon file—**or files if different sizes are needed**—default channel preference files, and XLST file)
   c. Browser loadable SWF for demonstration/validation.
   d. A set of test cases to run to verify channel functionality.

   NOTE Contact Adobe for more information reqarding the digital signing process.
2. A completed Dashboard Channel Submission document, provided to the developer by Verizon Wireless.
3. A final stand-alone reference SWF file that emulates channel design and functionality. (This is an updated version of original reference SWF file originally submitted for channel approval.)
4. A final descriptive walk-through of the channel from a user perspective, including interaction model, and intended animation behaviors.
5. Content for the Dashboard companion site including channel description and screenshot.

Dashboard Channel Submission

B

Instructions

The Channel Provider will complete all sections of this document, except items that are specifically marked for Verizon Wireless use only. Replace the [Channel Name] and [Date] items in the title page with the title of the channel and the date that the channel is submitted to Verizon Wireless.

[Channel Name]

[Date]

Channel Description

Channel Description	
Channel Title *This title will be displayed to subscribers of the Dashboard service (64 characters maximum).*	
Version *Each submission of a channel must have a new version.*	
Short Description *This description will be displayed to subscribers of the Dashboard service (255 characters maximum).*	

Long Description *This description will be displayed to subscribers of the Dashboard service.*	
Category *(VZW Use Only)*	
Price *(VZW Use Only)*	
Placement *(VZW Use Only)*	
Operation *Describe whether this is a new channel, modification to an existing channel, or addition of device support. (VZW Use Only)*	

Contact Information

Contact Information	
Company Name	
Business Contact	Name: Phone: E-Mail:
Technical Contact	Name: Phone: E-Mail:
Operational Contact	Name: Phone: E-Mail:
FlashCast SWF Developer *Provide the name of the organization that developed the channel SWF. This organization must be registered with Adobe's FlashCast developer program.*	
Content Server Provider *Provide the name of the organization that hosts or operates the content server.*	

Revision History

Describe all changes made to the channel submission document and channel assets.

Revision History		
Version	Date	Description of Changes

Device Support

Dashboard channels are required to support every device that the service supports. List all devices supported by the channel.

Device Support		
Device Name (e.g. Samsung U540)	Platform Identifier (e.g. 2197)	Channel Resolution (e.g. 176x203)

Channel Assets

All assets must be delivered with this submission document.

Channel Assets	
Development FLA Filename	
Development Properties Filenames	
Development ActionScript Filenames	
Deployment ZIP Filename	
Reference SWF Filename	
Channel Icon Filename	
Companion Description HTML Filename *This file may contain up to three paragraphs of HTML-formatted text describing the channel in detail. The HTML tags used in the text must be supported by version 7 of Adobe Flash Player*	
Companion Large Image Filename *A 176px by 208px JPG or SWF image to be displayed in the channel detail view.*	
Companion Small Image Filename *A 66px by 72px JPG or SWF image to be displayed in the available channel and featured channel lists.*	
Terms and Conditions HTML Filename *This file contains the legal agreement presented to a subscriber when they attempt to subscribe to a premium channel. HTML tags used in the text must be supported by version 7 of Adobe Flash Player*	
FAQ XML Filename *This XML document contains frequently asked questions about the channel.*	
Content Server URL	
Content Server Credentials	Username: Password:

Content Server IP Address	
Company Logo Filename	

Data Usage

Describe the usage of data by the channel SWF, and the aggregate data usage required for updates from the content server.

Client Data Usage	
Channel SWF Size (KB)	
Channel Preference Size (KB) *Enter the maximum size of the preferences in the feed store.*	
Channel Feed Data Size (KB) *Enter the maximum size of the channel feed data in the feed store.*	
Total Channel Feed Store Size (KB) *The total feed store size must be less than 133KB.*	
Average Feed Data Update Size (KB) *Average size of an incremental update to a client, including all the updates normally received during a week of usage. Describe the range of preferences used to calculate this average.*	
Feed Data Update Frequency *Enter the frequency of feed data updates requested by the channel.*	
Server Data Usage	
Data Source Update Frequency *Enter the frequency of requests from the FlashCast server to the content server.*	
Data Source Update Size (KB) *Enter the expected size of a single update to the FlashCast server from the content server, based on the expected subscriber base and a normal distribution of preferences.*	

Content Update Frequency *Enter the frequency of changes to content on the content server.*	

Platform Dependencies

Hard-coded linkaways to BREW applets are not permitted. Channels must use Ichiro data for all linkaways to BREW applets.

Ichiro Linkaway Data	
Ichiro Data URL	
Ichiro Credentials	Username: Password:

Other Platform Dependencies	
Description *Detail any other dependencies on the Platform ID or device characteristics.*	

Advertising

Verizon Wireless will place advertising in channels using feeds from Ichiro. Ads must use standard sizes defined in the channel style guide.

Advertising		
Location	Context	Size

Ichiro Advertising Data	
Ichiro Data URL	
Ichiro Credentials	Username: Password:

Security Controls

Describe the process or technical controls that have been implemented to ensure the integrity and availability of the channel.

Client Security Controls	
SWF Format / Corruption *Describe the controls in place to ensure that SWFs used by the channel are in the correct format, for the correct Flash Lite version, and have not been corrupted. Include controls that apply to SWFs delivered from the content server.*	
Data Store Usage *Describe the controls in place to ensure the channel does not exceed its data store limit.*	
Maximum Preferences and Channel Data *Insert the specific preference settings and channel update data that result in the maximum data store usage.*	Preferences: Channel Data: Total Bytes:
Runtime Memory Usage *Describe the conditions under which the channel uses the maximum runtime memory, and the controls in place to ensure the channel does not exceed its runtime memory limit.*	
Processor Usage *Describe processing intensive tasks in the channel, including animations, and the controls in place to ensure that the channel remains responsive to user input.*	
Preference Specification *Describe the controls in place to ensure that the channel is updated only with the data that is specified by the preferences.*	
Preference Validation *Describe how the channel handles corrupt or outdated preferences.*	

Server Security Controls	
Content Privacy & Integrity *Describe the mechanism which ensures the privacy and integrity of channel data transmitted between the content server and the FlashCast server.*	
Content Server Authentication *Describe the mechanism that the content server uses to authenticate the FlashCast server.*	
OWASP Security Guideline Compliance *Describe the steps that have been taken to ensure that the content server complies with the OWASP Security Guidelines*	
Input Validation *Describe the controls in place to validate requests from the FlashCast server to the content server. Describe the response to invalid URLs, improperly encoded characters, or requests for missing/prohibited objects.*	
Output Validation *Describe the controls in place to ensure that the content delivered from the content server to the FlashCast server is suitable for the channel. Describe the controls that prevent improper encoding of content, or delivery of unsupported objects.*	
Payload Size *Describe the controls in place to ensure that the content server does not deliver content which will overrun the channel's data store limit.*	
Device Dependencies *If the content server delivers content that depends on the Platform ID or other device characteristics, describe the mechanism that ensures that this content is compatible with the device.*	

Descriptive Walkthrough

Provide a walkthrough of the channel's user interface and navigation, including all of the screens or modes that the channel displays. Include a diagram of navigation through the channel.

[Insert navigation diagram]

Describe the functions assigned to hard and soft keys, e.g. LSK, CSK, RSK, CLR, SEND, END, 0-9, *, #, VOL+, VOL-.

Hard and Soft Key Usage			
Key	Soft Key Label	Context	Function

Data Entry

Describe all the data entry options presented to the user by the channel.

Data Entry		
Field Name	Range of Valid Entries	Response to Invalid Entry

Sounds

List all of the sounds played by the channel.

Sounds		
Sound Name	Context	Sound Format

Telecommunication

Describe all of the cases where the channel can initiate communications.

Telecommunication	
Call Initiation *Describe the cases, if any, in which the channel initiates voice calls. Describe the source of the number that is dialed.*	

Message Initiation *Describe the cases, if any, in which the channel initiates SMS messages. Describe the source of the recipient MDN and message body data.*	

Error Messages

Describe error messages that are returned to the end user, or delivered by the content server to the FlashCast server.

End User Error Messages		
Message	Context	Remediation

Content Server Error Messages		
Message	Context	Remediation

Index

A

ActionScript 62
    ActionScript call example 62
    adding to views 55
    button events 83
    examples 63–64
    FSCommand ("Launch") 62
    key events 83
    parameters 62
adding
    variation preferences 45
adding ActionScript to views 55
Adobe Flash CS3 Professional
    configuring extension 24
    installing 22
Adobe Mobile AppBuilder 15
    configuring extension 24
    device sets 15
    installing 22
    testing 41
    troubleshooting 17
aggregating data 34
animation 93
API, ActionScript 62
applications
    available keys 83
    Channel Guide 28
    designing 11–12
    developing 5–14
    Flash Cast 28
    Now Playing 28
    reserved keys 84–??
appname parameter (FSCommand) 62
args parameter (FSCommand) 62

B

basic channels 2
binary data 73
BREW application
    examples 63
    launching 63
    links 61
buttons
    Cancel buttons 77
    events (ActionScript) 83
    examples 84
    invisible 84
    Save buttons 77
    visible 84

C

Cancel buttons 77
changing fonts 107
Channel Development Framework
    navigating 81
channel guide 28
channel headers
    height limits 49
    properties 49
channels 2
    basic 2
    budget 98
    channel data 36
    channel guide 28
    components 5
    configuring 50
    content channels 28
    creating 27
    data options 67
    data sources 34

defining data 54
header limits 49
limits 99
linking to applications 61
linking to services 61
navigating 38–40
Now Playing channel 28
premium 3
profiles 53
quality expectations 109
single-profile testing 40
stacks 51–52
subscribing 29
subscription limits 3
testing 40
viewing data 36
views 51–52
client software (Flash Cast) 30
CLR key event 91–92
code samples
See examples
configuring
channels 50
development environment 21–25
content
channels 28
content structure 101
content submissions 109
creating
content stacks 54
Flash Cast channels 27
variations 42
views (channels) 52

D

<datasource> element 103
Dashboard 28
channels 2
CLR key event 91–92
service 1
troubleshooting 17
user input 64
user interface 28
data
aggregating 34
defining data (channels) 54
external data 34
managing data 34
requesting to mobile handset 37
retrieving 34
updates 78
data source server (Flash Cast) 33
default preferences 75, 76
defining 53
channel stacks 54
stack views 54
stacks (channels) 54
deleting
variation preferences 45
variations 45
descriptor file (db.xml) 101
developing applications
designing applications 11–12
development process 9–10
development team 7–8
environment 21–25
framework 12–14
single-SWF testing 18
submittal guidelines 109–111
tools 11
workflow 8
development environment 21–25
hardware requirements 21
setting up 22–25
software requirements 21
development framework 12–14
file structure 13
Flash Lite FLA file 14
development team 7
development tools 11
devices
device sets 15
Flash Cast 15
fonts 105
profiles 49–50
DSS
See data source server

E

EditText command
examples 67
embedded fonts 100, 105, 106
error view (channels) 53
examples
<datasource> element 103
<feed> element 102

<feedapp> element 102
ActionScript API 63
BREW application 63
EditText command 67
hide button 84
SendCLRKeyEventToSWF (FSCommand) 93
SetSoftKeyState (FSCommand) 90
show buttons 84
XML syntax 95
exporting projects 16, 46

F

<feed> element 102
<feedapp> element 102
feed data
loading 68
feed item identifiers, loading 71
feed records 69
lists 70
feed server (Flash Cast) 33
feedstore
overruns 104
space limitations 98
feedstore (Flash Cast client) 30
file structure (development framework) 13
FLA files 17
Flash Cast
applications 28
client 30
client software 30
data source server 33
defined 5–27
device sets 15
external data 34
feed server 33
feedstore 30
library 31
managing data 34
porting layer 31
reporting server 33
requesting data 37
retrieving data 34
server system 33
supported features 31–32
system overview 29–30
XML nodes 96
Flash Lite 31
defined 5–27
FLA file 14
fonts
changing 107
device fonts 105
embedded 100
embedded fonts 105, 106
installing 23
framework
key handlers 85
FSCommand API
EditText 65
inputmethodN parameter 66
multiLineN parameter 66
numfields parameter 65
parameters 65
staticN parameter 66
textvarN parameter 65
title parameter 65
userpasswordN parameter 66

H

hardware requirements (dev environment) 21

I

images 100
loading 70
scaling 59
initializing
channel views 56–57
views 55
inputmethodN parameter (FSCommand) 66
installing
Adobe Flash CS3 Professional 22
Adobe Mobile AppBuilder 22
Adobe Mobile AppBuilder extension 24
Flash Cast fonts 23
invisible buttons 84
itemid parameter (FSCommand) 62

K

key events (ActionScript) 83
keys
available keys 83
clear (CLR) key event 91

key handlers 85
managing soft keys 86
reserved keys 84–??

L

labels, soft keys 87
last update 79
Launch parameter (FSCommand) 62
library (Flash Cast) 31
linkaways 61
ActionScript 62
BREW application links 61
data 62
examples 63–64
mobile web links 61
Search links 61
types of linkaways 61
links 61
looping 72

M

managing data (Flash Cast) 34
managing variations 42
memory
memory usage 100
runtime overruns 104
methods 71
mimetype parameter (FSCommand) 62
Mobile AppBuilder
testing 24
mobile optimization 100
mobile shop example 63
mobile web links 61
multiLineN parameter (FSCommand) 66
multiple devices 58
multiple variations 58
testing 58

N

navigating
channels 38–40
elements 82
views 81
Now Playing channel 28
numfields parameter (FSCommand) 65

O

opening
Web pages 64

P

porting layer (Flash Cast) 31
preferences, variations 43
premium channels 3
profiles 53
definitions 49–50
projects
exporting 16, 46
managing variations 42
sharing 16
properties
screens 49
scrollbar 50
text fields 72
pseudo-arrays 71
publishing
multiple devices 58

Q

quality expectations (channels) 109
query properties 34

R

removefocus frame 82
reporting server 33
reserved keys 84–??
resource files 17, 42
retrieving data 34
reusable methods 71

S

sample code
See examples
Save buttons 77
screens
height limits 49
properties 49
width limits 49
script timeout 104
scrollbar

height 50
horizontal position 50
properties 50
thumb height 50
vertical position 50
scrolling text 72
Search links 61
security 104
feedstore overruns 104
runtime overruns 104
script timeout 104
select frame 82
SendCLRKeyEventToSWF (FSCommand)
example 93
SendCLRKeyEventToSWF (fscommand) 92–93
server system (Flash Cast) 33
servers
data source server (DSS) 33
feed server 33
reporting server 33
server system (Flash Cast) 33
services 61
setfocus frame 82
SetSoftKeyState (FSCommand) 87, 90
SetSoftKeyState (fscommand) 87–??
setting up development environment 21–25
sharing projects 16
single profiles 40
soft key menu bar
limits 49
properties 50
soft keys
labels 87
managing 86
soft key behavior 88
software
Flash Cast client 30
requirements (dev environment) 21
sorting variations 46
stacks (channels) 51–52
content stacks 54
defining 54
stack views 54
switching 55
startmode parameter (FSCommand) 63
staticN parameter (FSCommand) 66
submittal guidelines for channels 109–111
subscribing to channels 29
subscription limits 3
SWF file, loading 70
switching
stacks 55
views 55

T

testing 40, 100
Adobe Mobile AppBuilder 41
channels 40
locally 41
Mobile AppBuilder installation 24
multiple devices 58
single-SWF testing 18
text fields 72
textvarN parameter (FSCommand) 65
title parameter (FSCommand) 65
troubleshooting
Adobe Mobile AppBuilder 17
Dashboard 17

U

user input (Dashboard) 64
user interface (Dashboard) 28
user preferences 73, 74
Adobe Mobile AppBuilder 76
batching 77
default preferences 75, 76
loading preferences 76
saving preferences 77
userpasswordN parameter (FSCommand) 66

V

variations
adding preferences 45
creating 42
deleting 45
deleting preferences 45
editing 42
managing 42
organizing preferences 45
preferences 43
sorting 46
Verizon Wireless Dashboard 28
See Dashboard
viewing
channel data 36
views

adding ActionScripts 55
initializing 55
switching 55
views (channels) 51–52
creating 52
error view 53
initializing 56–57
visible buttons 84

W

Web pages 64

X

XML
descriptor file 101
Flash Cast nodes 96
syntax 95