Autodesk®
Scaleform®

Mobile Game Kit Overview

This document describes the architecture, source code, and content of the Scaleform 4.2 Mobile Game Kit, a cross-platform, touch/gesture-based game demo designed for mobile and tablet hardware using Scaleform's mobile player.

Author: Nate Mitchell

Version: 1.01
Last Edited: June 5, 2012

Autodesk®
GAMEWARE

Copyright Notice

Autodesk® Scaleform® 4.2

© 2012 Autodesk, Inc. All rights reserved. Except as otherwise permitted by Autodesk, Inc., this publication, or parts thereof, may not be reproduced in any form, by any method, for any purpose.

Certain materials included in this publication are reprinted with the permission of the copyright holder.

The following are registered trademarks or trademarks of Autodesk, Inc., and/or its subsidiaries and/or affiliates in the USA and other countries: 123D, 3ds Max, Algor, Alias, AliasStudio, ATC, AUGI, AutoCAD, AutoCAD Learning Assistance, AutoCAD LT, AutoCAD Simulator, AutoCAD SQL Extension, AutoCAD SQL Interface, Autodesk, Autodesk Homestyler, Autodesk Intent, Autodesk Inventor, Autodesk MapGuide, Autodesk Streamline, AutoLISP, AutoSketch, AutoSnap, AutoTrack, Backburner, Backdraft, Beast, Beast (design/logo) Built with ObjectARX (design/logo), Burn, Buzzsaw, CAiCE, CFdesign, Civil 3D, Cleaner, Cleaner Central, ClearScale, Colour Warper, Combustion, Communication Specification, Constructware, Content Explorer, Creative Bridge, Dancing Baby (image), DesignCenter, Design Doctor, Designer's Toolkit, DesignKids, DesignProf, DesignServer, DesignStudio, Design Web Format, Discreet, DWF, DWG, DWG (design/logo), DWG Extreme, DWG TrueConvert, DWG TrueView, DWFX, DXF, Ecotect, Evolver, Exposure, Extending the Design Team, Face Robot, FBX, Fempro, Fire, Flame, Flare, Flint, FMDesktop, Freewheel, GDX Driver, Green Building Studio, Heads-up Design, Heidi, Homestyler, HumanIK, i-drop, ImageModeler, iMOUT, Incinerator, Inferno, Instructables, Instructables (stylized robot design/logo),Inventor, Inventor LT, Kynapse, Kynogon, LandXplorer, Lustre, MatchMover, Maya, Mechanical Desktop, MIMI, Moldflow, Moldflow Plastics Advisers, Moldflow Plastics Insight, Moondust, MotionBuilder, Movimento, MPA, MPA (design/logo), MPI (design/logo), MPX, MPX (design/logo), Mudbox, Multi-Master Editing, Navisworks, ObjectARX, ObjectDBX, Opticore, Pipeplus, Pixlr, Pixlr-o-matic, PolarSnap, Powered with Autodesk Technology, Productstream, ProMaterials, RasterDWG, RealDWG, Real-time Roto, Recognize, Render Queue, Retimer, Reveal, Revit, RiverCAD, Robot, Scaleform, Scaleform GFx, Showcase, Show Me, ShowMotion, SketchBook, Smoke, Softimage, Sparks, SteeringWheels, Stitcher, Stone, StormNET, Tinkerbox, ToolClip, Topobase, Toxik, TrustedDWG, T-Splines, U-Vis, ViewCube, Visual, Visual LISP, Vtour, WaterNetworks, Wire, Wiretap, WiretapCentral, XSI.

All other brand names, product names or trademarks belong to their respective holders.

Disclaimer

THIS PUBLICATION AND THE INFORMATION CONTAINED HEREIN IS MADE AVAILABLE BY AUTODESK, INC. “AS IS.” AUTODESK, INC. DISCLAIMS ALL WARRANTIES, EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE REGARDING THESE MATERIALS.

How to Contact Autodesk Scaleform:

Document	Mobile Game Kit Overview
Address	Autodesk Scaleform Corporation 6305 Ivy Lane, Suite 310 Greenbelt, MD 20770, USA
Website	www.scaleform.com
Email	info@scaleform.com
Direct	(301) 446-3200
Fax	(301) 446-3199

Table of Contents

1 Introduction

The Scaleform Mobile Game Kit provides a best-practices sample implementation of a mobile, touch-based game built using Autodesk® Scaleform® as the engine and renderer. The game, Starforce Battlement, is written entirely in ActionScript 3 and leverages Scaleform's 'Shippable Mobile Player' for cross-platform deployment. A detailed description of the implementation is included within this overview. Although the game is specifically designed for mobile devices, it can be played on any platform that supports mouse or touch input, including PC, Mac, Linux, iOS, and Android.

The kit includes the full ActionScript 3 source code for the game and the main menu, including sample frameworks for multi-resolution support, sound playback, saving and loading to device, achievements, and an iOS Game Center integration. The kit also includes all of the Adobe® Flash® content and assets used in the game. All of the kit's resources can be reused by developers and/or leveraged as best-practices samples for architecture and implementation of their own game using Scaleform.

Although the game was designed to leverage Scaleform for playback on mobile devices, it can also be played back in a web browser using Adobe Flash Player making it a good benchmark for Scaleform's memory footprint and performance on various platforms.

A simplistic breakdown of the core components that makeup Starforce Battlement from a development/runtime perspective:

1. Flash Professional is used for interactive art, animation, UI layout, and level editing.
2. Gameplay logic is written in ActionScript 3 (FlashDevelop 4 is our recommended editor).
3. Exported .SWF files are played back by Scaleform on the device.
4. Scaleform's renderer takes care of drawing the playback result to the GPU of the device, effectively displaying it on-screen.

1.1 Starforce Battlement

Figure 1: Starforce - Level 1 Screenshot

Starforce Battlement is a tower defense game with role-playing game elements. The primary objective of each level is to prevent oncoming waves of enemies from reaching their destination. Every enemy that reaches the destination subtracts one point from the player's life. If enough enemies reach the destination that the player's life is reduced to zero, the game is over.

Players can build units/towers at designated locations on the map that will attack the oncoming enemies, preventing them from reaching the destination. Eliminating enemies rewards the player with credits which can be spent on building new towers or upgrading existing towers. When all enemy waves have been eliminated, the level is complete.

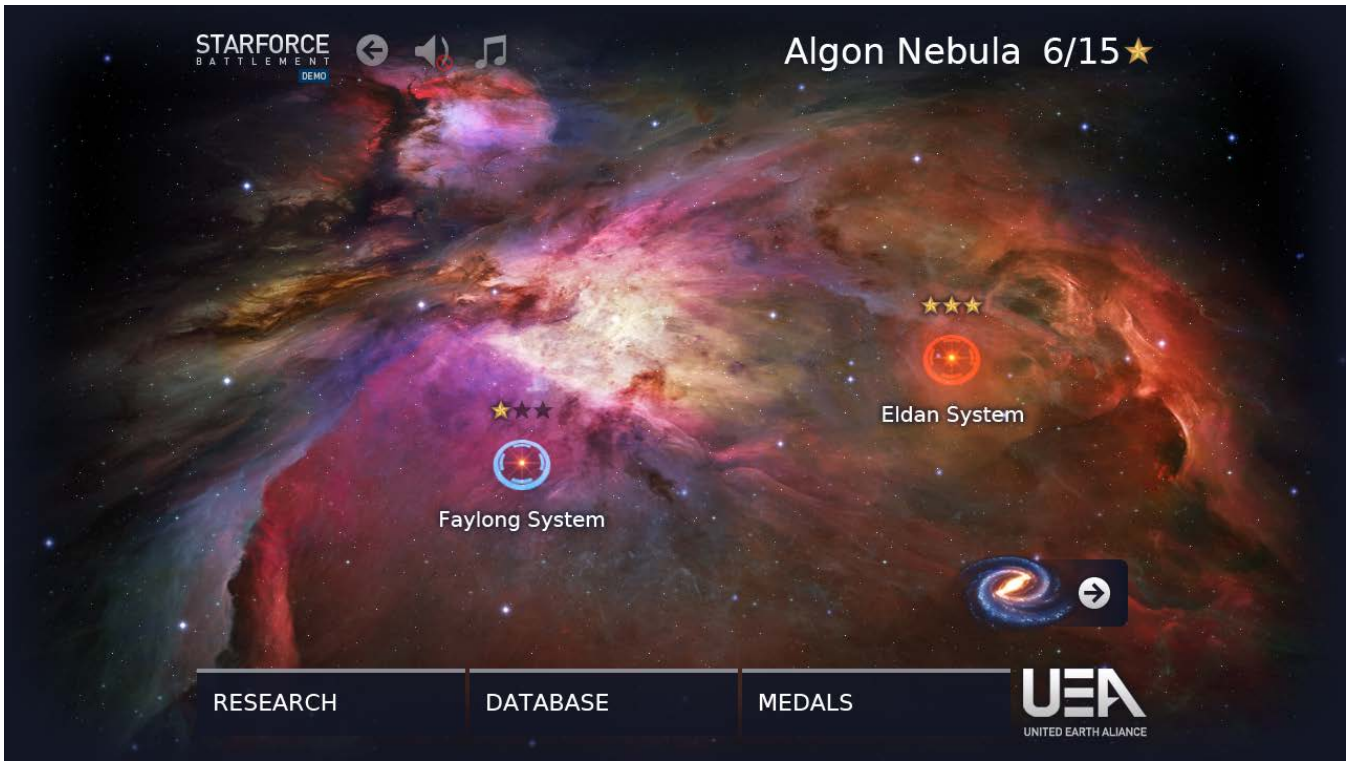

Figure 2: Starforce - Main Menu

As a full game demo, Starforce Battlement also includes a main menu for level select, tutorials, and viewing player statistics/achievements, all of which can be reused by developers for their own titles.

Starforce Battlement will be available as a free download on the iOS App Store in May 2012. Developers are encouraged to download the application to an iOS device to see how Scaleform works on iOS.

1.2 Features

Starforce Battlement showcases the following Scaleform mobile features:

1. Cross-Platform Mobile Support
2. GPU Vector and Bitmap Rendering
3. Full ActionScript 3 Support
4. Multi-Touch and Gesture Input Handling
5. Saving and Loading Data
6. Sound and Music Playback
7. iOS Game Center Integration
8. Orientation Lock / Orientation Handling

The kit also provides sample ActionScript 3 frameworks for the following:

1. Player Achievements
2. Sound and Music Playback
3. iOS Game Center Integration
4. Saving and Loading Data

2 Overview

2.1 File Locations and Build Notes

The files associated with this demo are located in the following locations:

- *Bin/Data/AS3/Kits/StarforceTD/* - Contains ActionScript 3 source code, Flash content, and other resources used during gameplay (icons, sounds, etc…).
- *Projects/Win32 /{Msvc80, Msvc90, or Msvc10}/GFx 4.1 SDK* / – Contains the Visual Studio project for building and debugging FxPlayer Visual Studio 2005/2008/2010 on Windows. Note that this application is not automatically configured to playback Starforce.
- *Projects/iPhone/Xcode4/GFx 4.1 iPhone SDK* – Contains the Xcode 4 project for "Shippable Mobile Player" on iOS. Note that this application is not automatically configured to playback Starforce.
- *LocalApps\StarforceTD\Android* – Upon your first successful build of Scaleform via make, this directory will contain a properly configured Eclipse project for "Shippable Mobile Player" already configured to playback StarforceTD on Android.

A pre-built executable of the demo for Windows, StarforceTD.exe, is included in *Bin/Data/AS3/Kits/StarforceTD.* It is also accessible via the Windows Start Menu or the Scaleform SDK Browser.

The project/solution files for FxPlayer or the FxMobilePlayer applications are used to compile, deploy, and run the mobile game kit on different platforms.

On Windows, ensure that the "Working directory" for Debugging is set to the *Bin/Data/AS3/Kits/StarforceTD* directory and that "Command line arguments" is set to *StarforceTD.swf*.

On iOS, after building and deploying the application to the device via Xcode, copy the SWF files and subdirectories (audio, icons) to the application using iTunes.

2.2 Demo Usage

2.2.1 Main Menu

Figure 3: Starforce - Main Menu

StarforceTD.as is the entry point for the game. This class is the Document Class for StarforceTD.fla and when initialized, initializes the MainMainState, which loads StarforceMenu.swf. StarforceMenu.swf contains the Main Menu art assets and the ActionScript 3 class definitions that drive the main menu.

The Main Menu allows the player to select a mission and provides additional options for changing the player's technology tree and viewing tutorials, the player's statistical history, and the player's achievements. The buttons at the top left of the Main Menu allow the player to toggle music, toggle sound effects, or exit the game.

Level selection consists of indicators placed on the galaxy/nebula map which the player can select. Each nebula contains a set of missions that are unlocked in a linear progression. The player can drag the map around in 2D space to explore available options before selecting a particular mission.

Figure 4: Starforce - Mission Data

When the player selects an available mission, the Mission Data View (MissionDataView.as) is displayed. The Mission Data View provides the player with details about the mission. From this view, the player may either launch the mission by selecting the “Enter Mission” button or may return to the Mission Select View by selecting the “X” button at the top right of the view.

2.2.2 Gameplay and HUD

Figure 5: Starforce - Level 1 Gameplay w/ UI

The levels and gameplay for Starforce Battlement are packaged in StarforceTD.swf. All data for the game's levels is loaded into memory when StarforceTD.swf initially begins, but no level is initialized until the LevelGameState is created. The game is broken up into five missions/levels, each with a unique setting.

The first wave of enemies will not be launched until the "Rush Wave" button, the pulsating crossed-swords button on right-hand side of the screenshot above, is clicked. After launching the first wave, enemies will arrive at set intervals for the duration of the level until all enemies are eliminated or the player is defeated. The current wave number and total number of waves are shown in bottom left hand corner of the HUD, along with a preview of the enemies that makeup the next wave.

If the player successfully eliminates all enemy waves, the player will be awarded a 1, 2 or 3-star rating for the level depending on the number of enemies that were allowed to reach the destination. These stars can be used to develop the player's technology tree.

Tower build sites, denoted by the black circles with diamonds on the map, are scattered throughout the level. When selected, a contextual radial menu appears above the site that allows the player to build a new tower. Only one tower may be built on a tower build site at a time. Towers can only be built on valid, available sites and the player must have enough credits to afford the tower. The player's current credit count is displayed on the right side of the HUD, just above the player's health.

Once a tower has been constructed, it can be upgraded a maximum of twice. Statistics for the current tower level along with the next level are presented at the center of the HUD when a tower is selected.

The player has two abilities, “Orbital Strike” and “Deploy Mercenaries,” that may be used at any time by clicking respective ability button at the bottom right of the HUD. After activating an ability, the player must select a location on the map where the ability should be launched or directed. Note that the Orbital Strike can be dragged after being launched by pressing and dragging the input, either the mouse or touch. Once an ability is used, the ability will ‘cool down’ and will become disabled for a set period of time. The duration of the cool down is indicated to the user by a dark tint over the ability button that drains as time passes.

The set of buttons at the top right of the HUD allow the player to view the tutorial, pause/resume the game, toggle sound effects, toggle music, and restart/exit the current mission.

3 Architecture

3.1 Flash Files

The majority of the art assets for the game are contained within the Flash .FLA files, including the levels, the entities, and UI, and the majority of the game's animations. Starforce Battlement primarily uses Bitmap art because the models for the units and towers were done in 3D for convenience.

A subset of the game's art, primarily icons used in the UI, is external to the FLA files. These external images are loaded at runtime via ActionScript.

Starforce Battlement consists of three distinct FLA files:

1. **StarforceTD.fla** – The primary .FLA for the game. This file contains the ActionScript code and art assets for all levels, UI, and entities for Starforce Battlement. It also serves as the manager for the game by loading and unloading the Main Menu / Loading Screen appropriately.
2. **StarforceMenu.fla** – This file contains the ActionScript code and art assets for the Main Menu and the Mission Select. Although it shares some user interface class definitions with StarforceTD.swf, it is entirely self-contained and can be tested stand-alone.
3. **LoadingView.swf** – This file contains the ActionScript code and art assets for the Loading Screen that is displayed during Game State transitions.

3.2 ActionScript Code

3.2.1 Implementation Summary

The logic for Starforce Battlement's gameplay and UI is written entirely in ActionScript 3. This makes gameplay code highly portable, since there is no device specific implementation inherent to Starforce other than the iOS Game Center integration.

3.2.1.1 Entry Point

The entry point for the game is the StarforceTD.as class, which is tied to StarforceTD.swf using the Flash Document Class property. More information on Document Classes can be found online at Adobe's Flash documentation.

3.2.1.2 Game States

The game is broken up into three 'GameStates', each representative of a different state that the game may be in. Only one GameState is active at a time and the active GameState is owned and managed by the StarforceTD class. The three game states are as follows:

1. MenuGameState – GameState for the Main Menu.
2. LoadingGameState – GameState for the Loading Screen.
3. LevelGameState – GameState for the levels and gameplay.

The constructor for StarforceTD initializes the subsystems (data, sound, multi-touch) and loads the main menu which is defined by StarforceMenu.swf and MainMenuView.as.

3.2.1.3 Main Menu Implementation

The main menu is primarily implemented in the com.scaleform.std.menu package (see below for more information on packages). The main menu is built on an event-driven architecture where events are passed between the various main menu views using ActionScript 3's native event system. As the user makes selections, events are dispatched and processed to open and close windows, read data, and/or transition to entirely new states (e.g. launching a mission).

When a mission is launched, a GameEvent is dispatched from the main menu. The StarforceTD.as class listens for this event and transitions to the level immediately by loading the LoadingGameState, unloading the MenuGameState, and loading the LevelGameState. The LevelGameState loads the selected level and begins the game as soon as the level as completed loading.

Note that a LoadingGameState is actually unnecessary from a technical point of view because the level content that will be drawn is already loaded in memory within StarforceTD.swf. However, the LoadingGameState transition is used here to hide the initialization of the Level and the HUD from the user and make for a smoother transition into gameplay.

3.2.1.4 Gameplay Implementation

Level.as is at the core of the game's implementation. This class serves as the base class for all of the game's levels. The Level.as class initializes and manages almost all of the elements that make-up the game including the level, the HUD, enemies, towers, and user abilities. Level is the base class for all of the levels. Each level has its own subclass which defines the enemies and may override other behaviors and variables to make the level more unique (e.g. default amount of gold, time between waves, enemy types, number of waves, etc…).

The Level is responsible for the creation, destruction, and management of all entities. Generally, the entities in the game are the towers and the enemies. All entities are ticked by the level, which causes them to update (i.e. move across the map, search for enemies, attack, etc…).

Flash Professional makes it easy to bind an ActionScript class to Symbol, which is comprised of graphics, animations, and ActionScript. These symbols can then be referenced and instantiated at runtime via ActionScript. This is the method Starforce Battlement uses for instantiation of almost all graphical content in the game and menu system. More information on binding Symbols to ActionScript and creating instances at runtime can be found in Adobe's Flash documentation.

For example, the TowerMech Symbol can be found in the Flash Library for StarforceTD.fla. In the properties for the TowerMech symbol, note that the symbol is bound to the com.scaleform.std.entities.TowerMech ActionScript class. Whenever this class is instantiated via ActionScript, an instance of this graphic Symbol will be created and can then be manipulated via code.

The Level.as class manages the game using an event-driven framework. The class listens for events that will be dispatched from entities, the user interface, or any one of the other system frameworks (sound, data, achievements) and updates the game and HUD accordingly. For example, when a Tower attacks a unit, rather than directly modifying the health of the unit it is attacking, the Tower dispatches an AttackEvent. The Level, listening for AttackEvents globally, receives the AttackEvent, processes the attacker and defender, calculates the amount of damage that the defender should receive, and applies that damage to the defender.

Most interoperability between game entities passes through the Level as a means of abstracting the various classes from one another and including data outside the scope of the entities in calculations (e.g. the player's technology tree, which may affect the amount of damage the defender takes in some cases). This design pattern also makes many game-wide changes easier to because it removes redundant code and keeps entity classes independent of one another.

3.2.1.5 Game Data

The majority of the data for the game, including statistics for all of the towers, missions, abilities, and enemies, is defined in the com.scaleform.std.data package. This data is accessed throughout the implementation using the GameData static class. This allows the GameData class to make modifications to the data before returning it to the requesting class in the case that an external variable is currently affecting it (for example, a player's technology tree may change the standard attack damage of a tower).

Ideally, the default data for the game would not be stored within ActionScript but rather some external data definition format like XML or JSON and then loaded at runtime. Unfortunately, XML was

unsupported in Scaleform 4.0 and therefore the quickest workaround was to store the data within class definitions. This may be updated in a future release to reflect best-practices.

3.2.1.6 Input Handling

Input for the game is scripted using ActionScript event listeners. This allows input logic to function cross-platform as the Scaleform Player will abstract the mouse/touch input across platforms and pass it to ActionScript appropriately.

The game listens for either mouse or touch input depending on whether the native Scaleform application currently supports touch input. Assuming FxPlayer has been setup correctly, the application will discern whether touch is supported at runtime based on whether a MultitouchInputState is installed on the MovieView and whether the proper operating system functions related to touch are available.

All interactive elements in Starforce have logic for both TouchEvents and MouseEvents, although only one set of listeners is actively used based on the application settings. The types for TouchEvents and MouseEvents are different, although many TouchEvent types have very similar MouseEvent types and vice versa. For example, a TouchEvent.TOUCH_TAP can be used like a MouseEvent.MOUSE_CLICK and a TouchEvent.PRESS can be used like a MouseEvent.MOUSE_DOWN.

To use a single function declaration as a listener for both TouchEvents and MouseEvents, the parameter of the function must be of type Event. If further information is required from the event, like the *target* or the coordinates of the input event, the event must be cast to the proper. One standard method for discerning the type at runtime is the *“is”* operator, shown below:

```
protected function onSetRallyClick( e:Event ):void {
    var point:Point;
    if (e is MouseEvent) {
        var me:MouseEvent = e as MouseEvent;
        point = new Point( me.stageX, me.stageY );
    }
    else {
        var te:TouchEvent = e as TouchEvent;
        point = new Point( te.stageX, te.stageY );
    }
}
```

3.2.2 ActionScript Packages

The ActionScript code for the game is divided into ActionScript ‘packages’ based on the functionality and/or purpose of the classes it contains.

All of the following packages are prefixed by “com.scaleform.std”:

1. **abilities** – Class definitions and gameplay logic for player abilities (Orbital Strike, Hire Mercenaries).
2. **controls** – Class definitions for UI elements and components reused throughout.
3. **core** – Class definitions for basic components of the state flow and data storage/retrieval.
4. **data** – Data definitions for the game including the player, entities, levels, and abilities.
5. **entities** – Data definitions for the game including the player, entities, levels, and abilities.
6. **events**– Event definitions for all game, UI, and state events.
7. **fx** – Class definitions for special effects in the game (explosions, death animations).
8. **hud**– Class definitions for the in-game HUD and menus.
9. **levels** – Class definitions for each level. Note that the Level class is one of the primary classes for the game.
10. **loading** – Class definitions for the Loading Screen.
11. **menu** – Class definitions for the Main Menu including all of the UI elements, mission select, and input handling.
12. **system** – Frameworks for interacting with the system from ActionScript for tasks like sound playback, saving and loading, and orientation handling.
13. **utils** – Utility classes, functions, and definitions used throughout the codebase, but not limited to a particular package.

3.2.3 Frameworks

3.2.3.1 Save/Load Framework

The save and load framework, defined in com.scaleform.std.system.SaveSystem.as, is a sample implementation for writing to and reading data from disk. Although the code used to store the player’s progress is simplistic, it is provided as a basis for a more complex implementation.

The ActionScript interface for this functionality leverages the SharedObject class. SharedObject is a standard ActionScript 3 class that allows developers to store and retrieve data from an ActionScript Object on disk. Scaleform’s implementation of the SharedObject class supports primitive types and Arrays.

Each SharedObject is defined by a unique string identifier used for reading and writing at runtime. In the code below, the SharedObject’s unique string identifier is “sb_player_data”.

```
// Local reference to the galaxy progress. Populated from the PlayerData.
public static var galaxyProgress:Array = null;
// Constant String for the property of the SharedObject we'll modify.
protected static const GALAXY_PROGRESS_SO_PROP:String = "galaxyProgress";
```

```
// Reference to the SharedObject once we retrieve it.
protected static var _playerDataSO:SharedObject = null;

public static function load( pd:PlayerData ):void {
    _playerDataSO = SharedObject.getLocal("sb_player_data");
    galaxyProgress = _playerDataSO.data[ GALAXY_PROGRESS_SO_PROP ];
    pd.setGalaxyProgress( galaxyProgress );
}

public static function save( pd:PlayerData ):void {
    // Write PlayerData into SharedObject.
    _playerDataSO = SharedObject.getLocal("sb_player_data");
    galaxyProgress = pd.getGalaxyProgress();
    _playerDataSO.data[ GALAXY_PROGRESS_SO_PROP ] = galaxyProgress;
    _playerDataSO.flush();
}
```

Note that developers are not limited to a single SharedObject instance. Multiple SharedObjects can be stored on disk as long as each uses a unique name. This is a convenient way to segment data into categories.

More detailed information on the SharedObject class can be found online here:
http://help.adobe.com/en_US/FlashPlatform/reference/actionscript/3/flash/net/SharedObject.html

3.2.3.2 Sound Framework

The sound framework, defined in com.scaleform.std.system.SoundSystem, is a sample implementation for handling sound effect and music playback within your game. The framework loads and plays external sounds, .mp3s in this case, from disk based on SoundEvents dispatched from the game. SoundEvents can be dispatched from any class with a reference to the stage.

```
dispatchEvent( new SoundEvent( SoundEvent.PLAY_SOUND, true, true, "infantry_attack.mp3",
                               1, false ) );
```

The sound framework uses two sound channels, one for the background and one for the foreground. SoundEvents designate which channel the target sound should play on. All sound effects for the game are played on the foreground channel while background music is played in the background channel. This allows each to be muted or toggled independently using a SoundEvent.

```
dispatchEvent( new SoundEvent( SoundEvent.MUTE_SOUNDFX, true, true );
```