Basic TRC Compliant Tutorial

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Document serial number: 000004892117

Table of Contents

About This Document

This document explains techniques used in the Basic TRC Compliant Tutorial sample code. The sample code is located in the PlayStation®Vita SDK package at:

```
%SCE_PSP2_SDK_DIR%/samples/sample_code/system/Basic_TRC_Compliant/
minimum_autosave
```

Typographic Conventions

This document uses the following typographic conventions.

Hyperlinks

Hyperlinks (underlined and in blue) are available to help you to navigate around the document. To return to where you clicked a hyperlink, select **View > Toolbars > More Tools** from the Adobe Reader main menu, and then enable the **Previous View** and **Next View** buttons.

Hints

A GUI shortcut or other useful tip for gaining maximum use from the software is presented as a Hint surrounded by a box. For example:

> **Hint:** This hint provides a shortcut or tip.

Notes

Additional advice or related information is presented as a Note surrounded by a box. For example:

> **Note:** This note provides additional information.

Text

- Names of keyboard functions or keys are formatted in a bold serif font. For example, **Ctrl**, **Delete**, **F9**.
- File names, source code, and command-line text are formatted in a fixed-width font. For example:

```
AppEventHandler::onEventArrived()
```

1 Overview

The sample shows a simple game-like application. It has a minimum feature set that is implemented in a Technical Requirements Checklist (TRC) compliant way.

This sample was developed with SDK 1.650/SDK1.800 and the *Technical Requirements Checklist for PlayStation®Vita*, version 1.2, and was checked by SCEE QA Support. Although new TRC versions may introduce new issues or add changes, it is also possible that something that may have resulted in a QA bug in the past may no longer be an issue. However, TRC changes should be incremental and relatively small. The recommended procedure is to consider the TRC from the beginning of the development cycle and then to keep up to date with version changes as they arise.

The sample code is an example of implementing a touch menu flow and basic game-related features, such as model rendering and audio. It supports a single-player fixed auto-save system, small-scale trophies, and a live-area page. The menu and game assets should be considered placeholders for real artwork, and the actual game screen is very simple because it is essentially a placeholder game. In the game screen, the score is increased by tapping the middle of the display; game-specific content and game-play mechanics were not considered in judging TRC issues. The project includes a package-building post process that runs in release build configuration. An already built package is also included with the source code.

The Master Submission Requirements information on the Developer Network (DevNet) home page specifies the correct version of the TRC and SDK to use when you are ready to submit your game for testing. You can install the most recent version of the *Technical Requirements Checklist for PlayStation®Vita* via SDK Manager, from the Publishing category.

The remainder of this document explains the implementation of TRC-related topics in the Basic TRC Compliant sample. Although this document provides explanations of some TRC requirements, they are only brief examples in the context of this particular application. Therefore, the relevant TRC requirement numbers (for example, R3002) are provided so that you can look them up in the TRC. For complete details about current requirements, always check the *Technical Requirements Checklist for PlayStation®Vita*.

Figure 1 Screenshots of Basic TRC-Compliant Sample

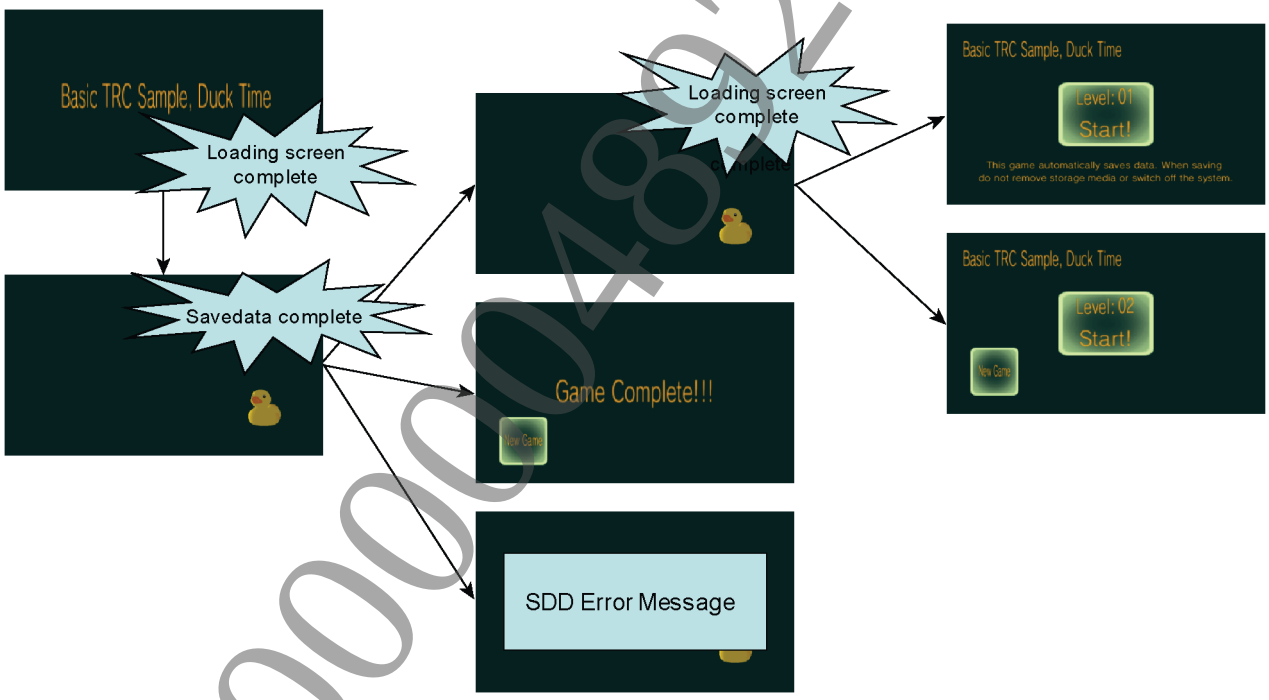
2 Application and Package

Application Overview

The application's main flow of control is contained within a single event handler, making it simple to track the interaction between the main systems, such as graphical user interface (GUI) touch buttons, save, trophy, and scene processing. The rendering-specific code outside of the `sample_utilities` and skeleton application class is accessed through a render layer. The goal of this architecture is to make the high-level flow and TRC-compliant functionality as clear as possible.

For example, Figure 2 shows the sequence of events and display when the sample application is run from Visual Studio. The initial game title screen is displayed while some assets are loaded and initialized. When the load thread is complete, the "Loading screen complete" event is dispatched (`EVENT_LOADINGSCREEN_COMPLETE`). The event is picked up in the event handler, `AppEventHandler::onEventArrived()`, which controls what to do next. In this case it calls the `SaveData` class in an attempt to load or create a new save game. Similarly when the savedata processing is complete, a "Savedata complete" event is received in the same event handler. This time the appropriate menu screen and processing are started depending on whether the savedata was successfully loaded, and whether the user has already reached the end of the game.

Figure 2 Loading Screens of Basic TRC-Compliant Sample



User Interface and System Dialogs

The application-specific GUI is contained in `gui.c/h` files and it is initialized and rendered by the scene class. Each instance of the `MenuScreen` class represents the layout and contents to be displayed on screen; this class supports text display, touch buttons and animated loading graphics. Sometimes system software dialogs also need to be displayed – for example, when there is a problem accessing the memory card. The Common Dialog Library provides dialogs that interface with other system or network libraries; in this sample the dialog library may be used by NP Toolkit and the `Savedata` class. A single instance of the Application Utility and message dialogs is required; therefore, the functionality is encapsulated in the `SystemDialogNP` class, and the dependent modules assume that the shared utility is initialized and that any dialogs are updated in the render loop, allowing them to appear in each frame.

System and network-related topics are described later in more detail. There are only a few TRC issues that relate to the application-specific GUI, as described below.

R3032/R3033 : ...display of important info...

The basic TRC-compliant sample implements the correct display of text labels, so that they are not too close to the edge of the screen. Although multiple languages are supported internally, the proper thing to check is the display of the final strings.

R3019/R3020: ...time required for loading.../unresponsive screens...

The loading screens that are implemented by the sample allow animation events to be triggered. Animation should be used if the waiting time on a particular screen could be long.

In this sample, because the loading times are not very long, an artificial delay was added. However, even in this case, animation was not strictly required. It is not a TRC issue to show animation if load times are short. The GUI system supports both short and long loading times. You can use `#define NO_LOADING_ANIM` to disable animation and the artificial delay. In this case, a blank screen is displayed for a very short time instead.

R3129: ...master contains no data created with a system-based font, font data is not saved...

The sample does use a system font at runtime to generate the GUI text; however, the data is not in the master package and is never saved.

Game Package and Publishing Tools

In addition to building and running sample code with Visual Studio, you can also build a package file or run an existing package for this sample. A pre-built copy of the package is in `minimal_autosave\PSP2_Package\package_01.00`.

For information about how to install package files, see the first appendix in the *Application Development Process Overview* (part of the SDK documentation).

To create a package from this sample, build it in Visual Studio, using the Release configuration. In release build, there is a post-build step that creates a package using the Package Generator tool. You can install this tool from the Publishing category of SDK Manager.

The build step calls `.bat` to create the package. It uses the Package Generator tool, a `.gp4p` input file, and the contents of the directory `minimal_autosave\PSP2_Package\app`. This directory contains all the data required by the game at runtime, and the same data is used when the game is run via Visual Studio. The `*.gp4p` file contains package metadata and file hierarchy; it is created using the Package Generator application.

To build a package for your own game, you will also need to:

- Edit the `param.sfo` file using the Param File Editor tool
- Create a trophy pack with the Trophy Pack File Utility

To begin development and edit your own package content, it is simplest to install the complete set of publishing documents and tools (including user's guide documentation) via the SDK Manager.

After your package is generated, you can verify it using the Package Generator Tool or by using the system software (hold the installed package icon; then tap the icon to access settings and select ***Debug Utility > *Check**).

The TRC issues regarding package creation are related to having correctly formatted IDs and filling in your version numbers and other data correctly when using the publishing tools.

R3002 : ...content ID is in the correct format ...

For this sample, the NP title ID, service ID, and trophy related data were obtained using the PSN℠ forms on DevNet. A valid service ID is required to construct a TRC compliant content ID. This can be obtained by adding the Commerce (Regional Store) Service for your title.

A title is initially registered on DevNet by a representative of the developer or publisher, who takes responsibility for adding required network services, and granting access to other DevNet users on their team. For information about using the PSN℠ forms, use the context-sensitive help in the forms as well as referring to https://psvita.scedev.net/docs/psnconfig.

The sample uses a title ID suitable for the **Test** product type; you will need to use a non-test type for production code. The service ID for the sample is EP9000-NPXB01693_00; therefore the content ID was set as EP9000-NPXB01693_00-DUCKTIMEMINIMALA. The arbitrary user data was chosen by taking the first 16 acceptable digits from the title name: Duck Time Minimal Autosave.

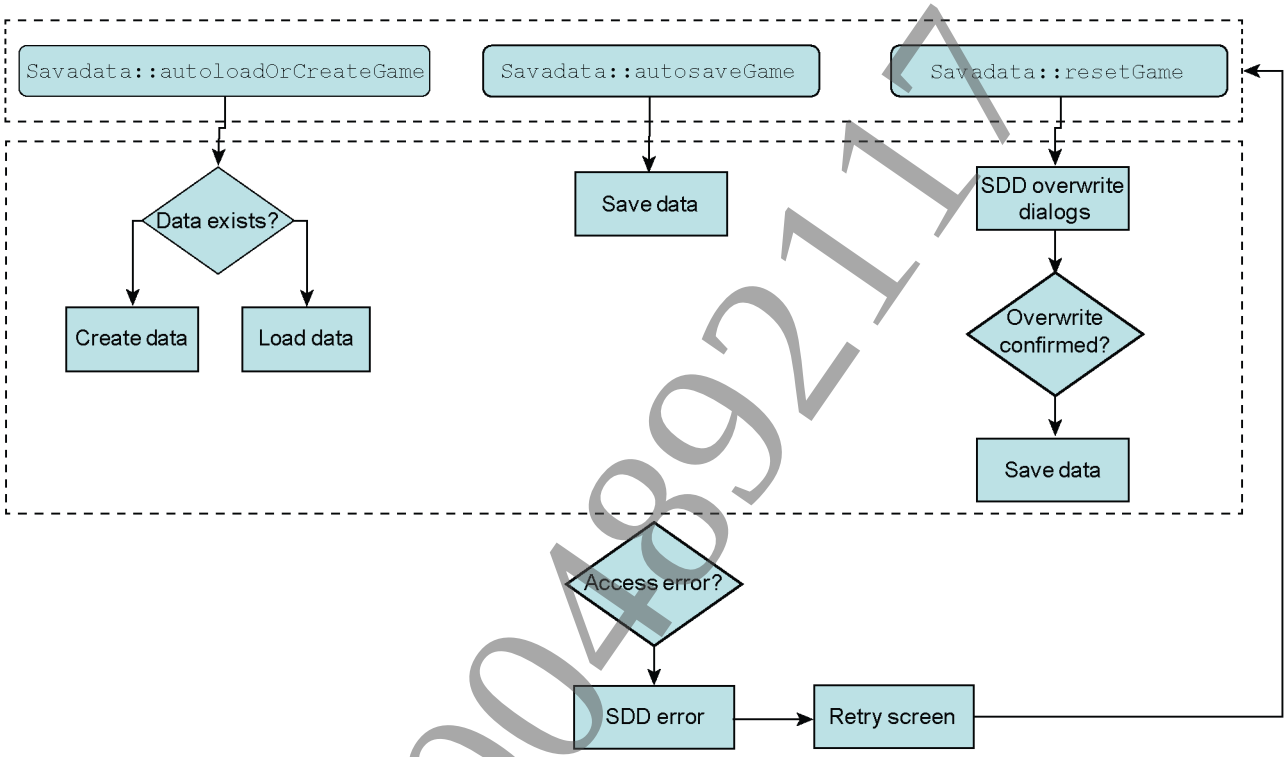
3 System and Network

Savedata

In this sample, the user interaction for savedata processes is very simple because only one save game/slot is allowed. Therefore loading, saving, and initial game creation do not require user intervention.

The **New Game** button will restart the game from the beginning (total score=0 level=0). In this case, because the existing saved progress is overwritten, a dialog asking for user confirmation is displayed.

Figure 3 Savedata Flow Diagram



The sample code implements the required behavior mainly within the `Savedata` class. Internally the `Savedata` functions in Application Utility are used, and a single fixed save is automatically created, loaded, and updated without any user input. The exception is when a problem accessing data causes the game to loop, or when the user selects **New Game**. Because only one savedata slot is used in this application, existing save data will be overwritten with the new data; therefore the user is asked for confirmation via the Save Data Dialog utility.

R3070: ...error messages via Save Data Dialog and retry...

All error messages are handled with the Save Data Dialog. When the game cannot continue, a dialog is displayed that allows the user to tap a button to retry the savedata access.

R3072: ...size used for save data and Save Data Quota...

This sample implements method (A) detailed in the TRC documentation for this issue; that is, it implements a single fixed-size save.

R3073: ...shortage of free space and retry...

This sample implements method (A) detailed in the TRC documentation for this issue; that is, it does not allow the game to continue until the free space error is resolved. As before, all relevant messages are handled with the Save Data Dialog. The user needs to make space on the storage media before the game

can progress. The application displays the "retry" screen, which has a single button so that the user can try again.

There are several test cases and detailed instructions in the TRC. In this sample, you can test the "not-enough space" scenario by doing the following:

- Select ***Debug Settings > Game > Fake Free Space > On**.
- Delete the existing savedata (the only time this application cannot have enough space is when there is no existing savedata).
- Run the application, which automatically tries to create a new saved game during start-up.

Trophy

This sample implements Trophy and NP-related features by using the NP Toolkit library.

The `TrophyUnlocker` class handles the Trophy-specific functions and assumes that shared functionality such as Common Dialog usage is already handled in the `SystemDialogNP` class.

Although Trophies are a network-related topic, this sample does not implement any functionality that requires a network connection or PSN℠ sign-on. Therefore the trophy class does not need to ask the user to sign in. If network functionality is used, a PSN℠ sign-in dialog should be displayed at relevant points.

R3037: ... PSN℠ sign in...

This requirement is not applicable to this sample. Network functionality is not used if a title only uses NP to support trophies.

R3012/R3013 : ...trophy pack file rules for small/large scale game points...

You can use the Trophy Pack File Utility to create and check the trophy pack file. Check the TRC for details as several are relevant to the contents of the sample's trophy pack.

LiveArea™

Note: The "click paths" in the following two paragraphs begin with Windows **Start** menu **> All Programs**.

Similar to the sample game's content, the example LiveArea™ content is a placeholder. General development information about LiveArea™ is in **SCE > PSP2 > SDK_doc > System > LiveArea™ Specifications**, and the content guidelines are in **SCE > PSP2 > Publishing > LiveArea(TM)_Content-Guidelines**.

The sample also contains a digital manual, which can be viewed via the LiveArea™ icon. For details, see **SDK_doc > Overview and Changes > Content Information Specifications**, as well as **SCE > PSP2 > Publishing > SCEE Region > PSVita Digital Manual Guidelines**. Software manual contents requirements can differ across regions. Currently a digital manual is required in the SCEE region, and all the template images for mandatory pages in the manual are installed as a part of the publishing package. This sample includes all the images from:

```
SCE\PSP2\Publishing\Regional\SCEE\PNG_Files\3rd_Party\18_English\Hi-Res
```

The images for the manual are simply added to the sample's package build directory at `minimal_autosave\PSP2_Package\app\sce_sys\manual`. They can be viewed from the LiveArea™ after the package file is installed.

R3030: ...display of title and information correct...

Because this sample implements a very basic LiveArea™ with no links and no publisher, there is only a placeholder message to add publisher details. This requirement also mentions that if a manual is supplied, the content of the manual must adhere to R3136.

R3136: ...add mandatory health and safety warnings and other messages...

The sample manual implemented is again very minimal; it contains all the mandatory pages with a single application-specific page.