Programming Startup Guide

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Introduction

Scope of This Document

This document provides the basic information required for application development and programming environments.

Reference Materials

When documents with detailed information are available, this document contains references to these documents for each item. Here, other documents are introduced for reference when starting SDK development.

Application Development Process Overview

This document explains requirements for developing an application using the SDK.

DevKit/TestKit Setup Guide

This document contains information related to the setup of the Development Kit (DevKit hereafter) and Testing Kit (TestKit hereafter).

Publishing Tools Overview

This document contains information related to software included in the Publishing Tools, which carries out master package creation and verification.

Hardware Overview

This document describes the main specifications and features of the PlayStation®Vita hardware.

Graphics Programming Tutorial

This document provides basic knowledge on graphics programming for PlayStation®Vita.

USSE Performance and Shader Optimizations: Tutorial

This document explains how to optimize shaders and provides tips for optimizing graphics performance.

Audio Tutorial

This document provides an overview of the five sound libraries prepared by the PlayStation®Vita SDK and instructions on how to use the NGS, which serves a major role in creating sound for game applications.

2 Basic Information on Development Environments

This chapter provides stage by stage explanations for the overall configuration of the various tools provided by the SDK for application development. The stages are as follows.

- Building
- DevKit Operation From the Development Host Computer
- Application Execution
- Debugging and QA
- Performance Tuning
- Testing Operation for the Retail Unit Using DevKit/TestKit

Building

The SNC Toolchain is provided as a tool to use for building an application. The Compiler and C++ standard libraries support C++03 and C++11. Default setting is C++03. C++11 can be used by specifying it as a compiler option upon build. Various tools of the SNC Toolchain will start up via the Visual Studio (with Visual Studio Integration installed) to build an application. For details, refer to the "Visual Studio Integration for PlayStation®Vita User's Guide" document.

SNC Toolchain

SNC Toolchain comprises tools for building source codes (compiler, linker, etc.).

SNC Compiler

The SNC Compiler is a C/C++ Compiler provided by SN Systems. High-performance, advanced optimizing controls are provided. For details, refer to "SNC Compiler User's Guide" document.

SN Linker

The SN Linker is a linker provided by SN Systems. High-performance, advanced features and settings are provided. For details, refer to "Linker User's Guide" document.

Build Utilities

Various build utilities are provided to support game development. For details on provided build utilities, refer to "Build Utilities User's Guide" document.

SN-DBS (SN-Distributed Build System)

When carrying out a distributed build over multiple computers, compile time can be greatly reduced. SN-DBS is provided as a tool to support distributed builds. Download SN-DBS from the PlayStation®Vita Developer Network site (https://psvita.scedev.net/) and install it.

Visual Studio Integration (VSI hereafter)

VSI is a Visual Studio add-in to integrate and manage the compiler, debugger and the performance analysis tool, Razor for PlayStation®Vita (Razor); these are also provided as Visual Studio add-ins.

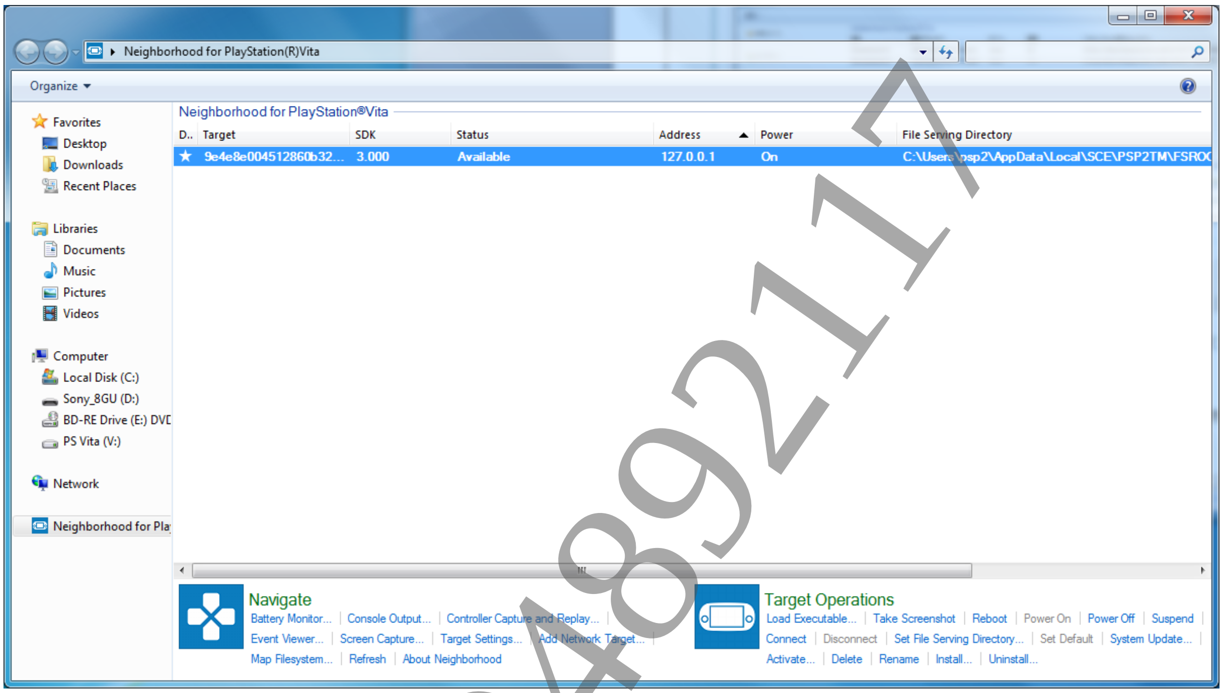
DevKit Operation From the Development Host Computer

The following methods are provided to operate DevKit from the development host computer.

Neighborhood for PlayStation®Vita (Neighborhood hereafter)

Operate DevKit from the development host computer using Neighborhood, which is integrated with the Windows Explorer.

Figure 1 Neighborhood


Neighborhood for PlayStation(R)Vita
Organize
Favorites
Desktop
Downloads
Recent Places
Libraries
Documents
Music
Pictures
Videos
Computer
Local Disk (C:)
Sony_8GU (D:)
BD-RE Drive (E:) DVD
PS Vita (V:)
Network
Neighborhood for Pla
Navigate
Battery Monitor... | Console Output... | Controller Capture and Replay...
Event Viewer... | Screen Capture... | Target Settings... | Add Network Target...
Map Filesystem... | Refresh | About Neighborhood
Target Operations
Load Executable... | Take Screenshot | Reboot | Power On | Power Off | Suspend
Connect | Disconnect | Set File Serving Directory... | Set Default | System Update...
Activate... | Delete | Rename | Install... | Uninstall...

Neighborhood for PlayStation®Vita

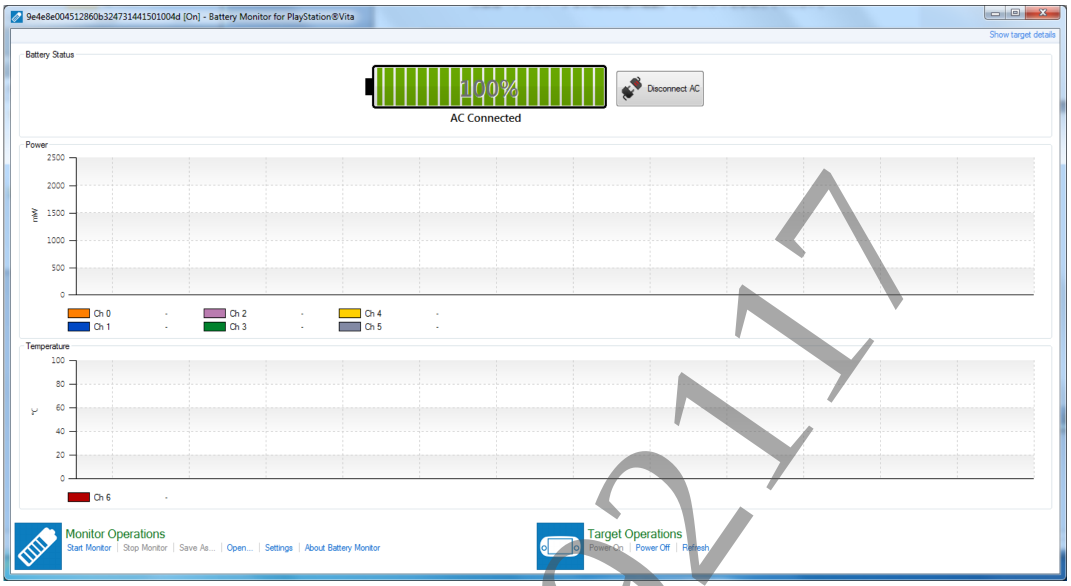
D..	Target	SDK	Status	Address	Power	File Serving Directory
★	9e4e8e004512860b32...	3.000	Available	127.0.0.1	On	C:\Users\psp2\AppData\Local\SCE\PSP2TM\FSROC

Connection between the development host computer and DevKit, as well as the DevKit power state, can be controlled on Neighborhood. The following features also are provided.

- Settings feature:
  Settings on DevKit can be changed using the **Target Settings** feature of Neighborhood. For details on the items that can be changed, refer to the "Development Kit Neighborhood Settings Guide" document.
- Console input/output confirmation feature:
  TTY output that is output to and from the DevKit can be confirmed on the development host computer using the **Console Output** feature of Neighborhood. In addition, standard input can also be performed with **Console Input**. For details, refer to the "Neighborhood and Utilities User's Guide" document.
- File system map feature:
  Files on the DevKit memory card can be accessed from Windows Explorer using the **Map Filesystem** feature of Neighborhood. For details, refer to the "Neighborhood and Utilities User's Guide" document. To access game data and save data installed on the memory card, refer to the "Application Development Process Overview" document.

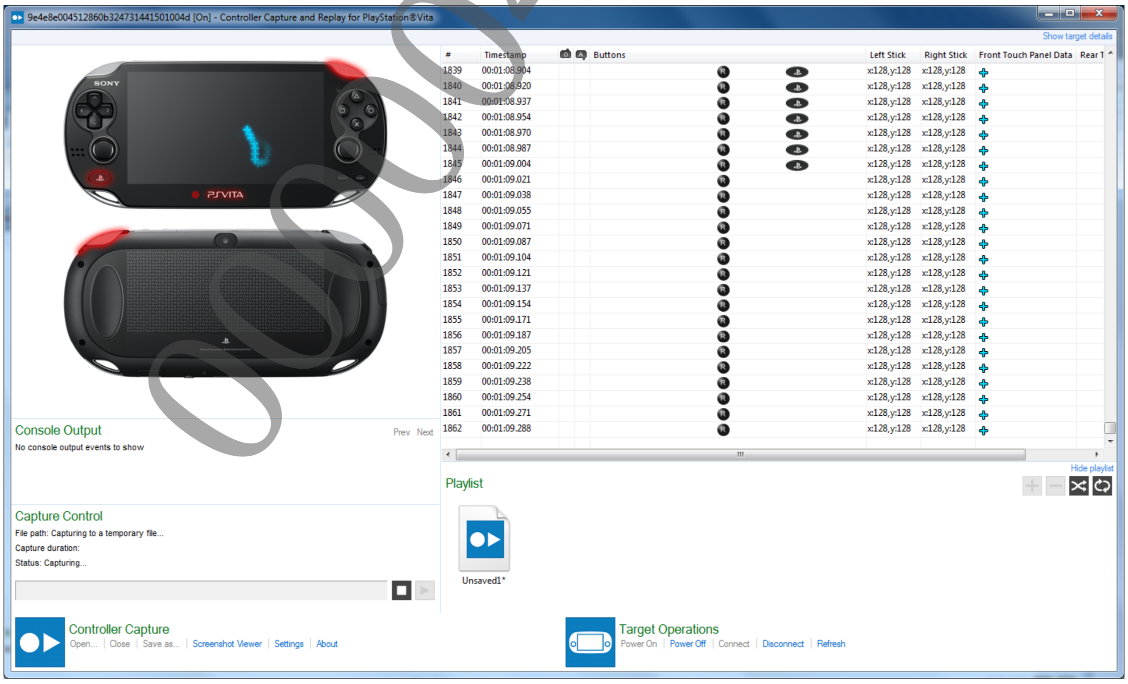
- Battery monitor feature:
  Battery emulation - including the virtual battery status (of the battery emulator) on DevKit and its consumed power, monitoring of the battery emulator's temperature, etc. - can be controlled using the **Battery Monitor** feature of Neighborhood. For details, refer to the "Battery Monitor User's Guide" document.

Figure 2 Battery Monitor


9e4e8e004512860b324731441501004d [On] - Battery Monitor for PlayStation®Vita
Show target details
Battery Status
100%
AC Connected
Disconnect AC
Power
mW
2500
2000
1500
1000
500
0
Ch 0
Ch 1
Ch 2
Ch 3
Ch 4
Ch 5
Temperature
°C
100
80
60
40
20
0
Ch 6
Monitor Operations
Start Monitor | Stop Monitor | Save As... | Open... | Settings | About Battery Monitor
Target Operations
Power On | Power Off | Refresh

- Controller capturing and replays feature:
  Controller input data from DevKit can be captured using the **Controller Capture and Replay** feature of Neighborhood. Save the captured input data as a file and replay the user's controller input from the stored data. For details, refer to the "Controller Capture and Replay User's Guide" document.

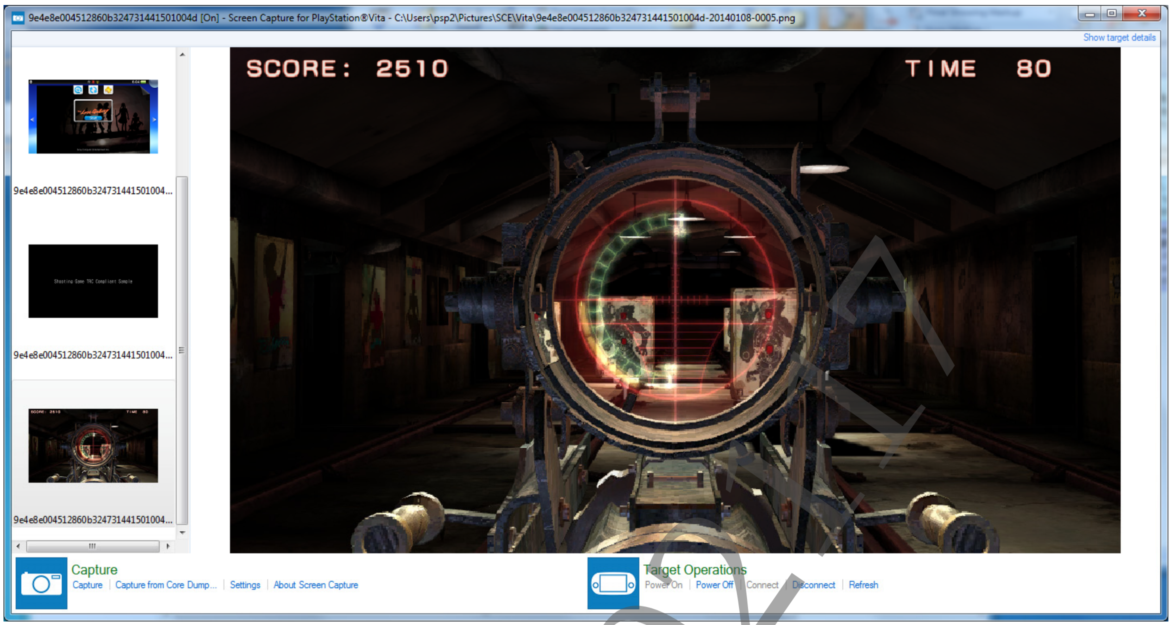
Figure 3 Controller Capture and Replay


9e4e8e004512860b324731441501004d [On] - Controller Capture and Replay for PlayStation®Vita
Show target details
SONY
PS VITA
Console Output
Prev Next
No console output events to show
Capture Control
File path: Capturing to a temporary file...
Capture duration:
Status: Capturing...
Hide playlist
Playlist
Unsaved1*
Controller Capture
Open... | Close | Save as... | Screenshot Viewer | Settings | About
Target Operations
Power On | Power Off | Connect | Disconnect | Refresh

#	Timestamp	Buttons		Left Stick	Right Stick	Front Touch Panel Data	Rear T
1839	00:01:08.904	Ⓡ	●	x:128,y:128	x:128,y:128	+	
1840	00:01:08.920	Ⓡ	●	x:128,y:128	x:128,y:128	+	
1841	00:01:08.937	Ⓡ	●	x:128,y:128	x:128,y:128	+	
1842	00:01:08.954	Ⓡ	●	x:128,y:128	x:128,y:128	+	
1843	00:01:08.970	Ⓡ	●	x:128,y:128	x:128,y:128	+	
1844	00:01:08.987	Ⓡ	●	x:128,y:128	x:128,y:128	+	
1845	00:01:09.004	Ⓡ	●	x:128,y:128	x:128,y:128	+	
1846	00:01:09.021	Ⓡ		x:128,y:128	x:128,y:128	+	
1847	00:01:09.038	Ⓡ		x:128,y:128	x:128,y:128	+	
1848	00:01:09.055	Ⓡ		x:128,y:128	x:128,y:128	+	
1849	00:01:09.071	Ⓡ		x:128,y:128	x:128,y:128	+	
1850	00:01:09.087	Ⓡ		x:128,y:128	x:128,y:128	+	
1851	00:01:09.104	Ⓡ		x:128,y:128	x:128,y:128	+	
1852	00:01:09.121	Ⓡ		x:128,y:128	x:128,y:128	+	
1853	00:01:09.137	Ⓡ		x:128,y:128	x:128,y:128	+	
1854	00:01:09.154	Ⓡ		x:128,y:128	x:128,y:128	+	
1855	00:01:09.171	Ⓡ		x:128,y:128	x:128,y:128	+	
1856	00:01:09.187	Ⓡ		x:128,y:128	x:128,y:128	+	
1857	00:01:09.205	Ⓡ		x:128,y:128	x:128,y:128	+	
1858	00:01:09.222	Ⓡ		x:128,y:128	x:128,y:128	+	
1859	00:01:09.238	Ⓡ		x:128,y:128	x:128,y:128	+	
1860	00:01:09.254	Ⓡ		x:128,y:128	x:128,y:128	+	
1861	00:01:09.271	Ⓡ		x:128,y:128	x:128,y:128	+	
1862	00:01:09.288	Ⓡ		x:128,y:128	x:128,y:128	+	

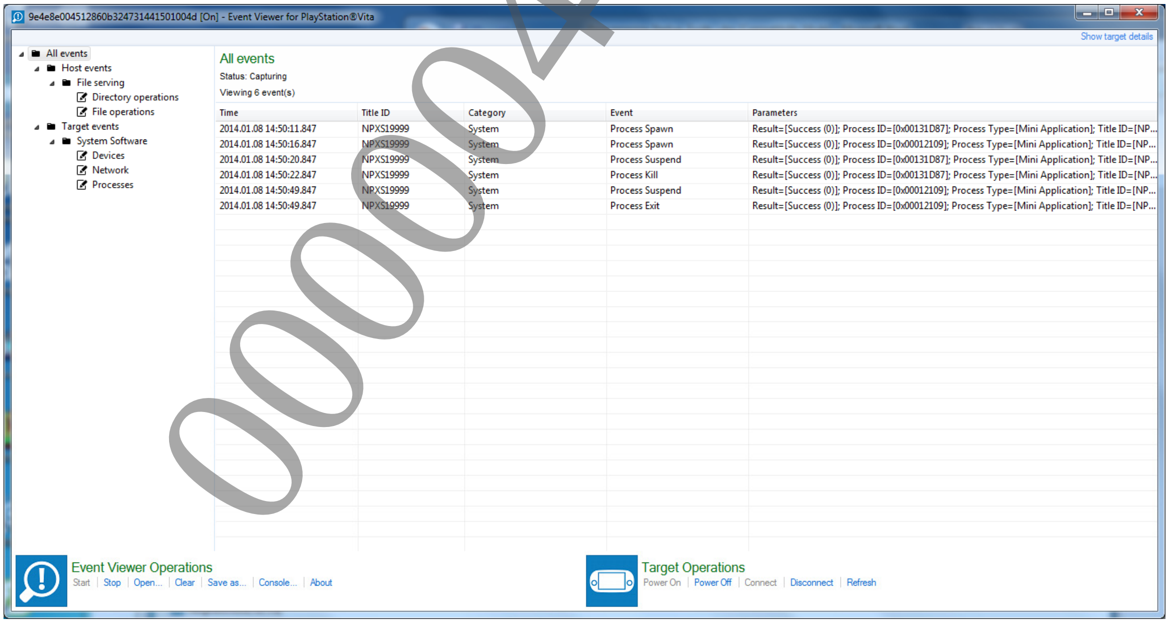
- Screen capturing feature:
  The DevKit screen can be captured as an image file using the **Screen Capture** feature of Neighborhood. For details, refer to the "Screen Capture User's Guide" document.

Figure 4 Screen Capture


9e4e8e004512860b324731441501004d [On] - Screen Capture for PlayStation®Vita - C:\Users\psp2\Pictures\SCE\Vita\9e4e8e004512860b324731441501004d-20140108-0005.png
Show target details
SCORE: 2510
TIME 80
9e4e8e004512860b324731441501004...
Capture
Capture | Capture from Core Dump... | Settings | About Screen Capture
Target Operations
Power On | Power Off | Connect | Disconnect | Refresh

- Event viewer feature:
  The **Event Viewer** feature of Neighborhood can be used to view events on DevKit. For details, refer to the "Event Viewer User's Guide" document.

Figure 5 Event Viewer


9e4e8e004512860b324731441501004d [On] - Event Viewer for PlayStation®Vita
Show target details
All events
Host events
File serving
Directory operations
File operations
Target events
System Software
Devices
Network
Processes
Event Viewer Operations
Start | Stop | Open... | Clear | Save as... | Console... | About
Target Operations
Power On | Power Off | Connect | Disconnect | Refresh

All events

Status: Capturing

Viewing 6 event(s)

Time	Title ID	Category	Event	Parameters
2014.01.08 14:50:11.847	NPXS19999	System	Process Spawn	Result=[Success (0)]; Process ID=[0x00131D87]; Process Type=[Mini Application]; Title ID=[NP...
2014.01.08 14:50:16.847	NPXS19999	System	Process Spawn	Result=[Success (0)]; Process ID=[0x00012109]; Process Type=[Mini Application]; Title ID=[NP...
2014.01.08 14:50:20.847	NPXS19999	System	Process Suspend	Result=[Success (0)]; Process ID=[0x00131D87]; Process Type=[Mini Application]; Title ID=[NP...
2014.01.08 14:50:22.847	NPXS19999	System	Process Kill	Result=[Success (0)]; Process ID=[0x00131D87]; Process Type=[Mini Application]; Title ID=[NP...
2014.01.08 14:50:49.847	NPXS19999	System	Process Suspend	Result=[Success (0)]; Process ID=[0x00012109]; Process Type=[Mini Application]; Title ID=[NP...
2014.01.08 14:50:49.847	NPXS19999	System	Process Exit	Result=[Success (0)]; Process ID=[0x00012109]; Process Type=[Mini Application]; Title ID=[NP...

Command Line Tool psp2ctrl

Operation of DevKit from the development host computer is possible using the command line tool psp2ctrl. For details on the commands provided by psp2ctrl, refer to the "Neighborhood and Utilities User's Guide" document.

Creating a Custom Tool Using Target Manager (TM hereafter) APIs and libdeci4p

A custom tool to operate a program on DevKit from the development host computer can be created using TM APIs and the libdec4p library.

TM APIs

TM APIs are provided for operating DevKit from a program on the development host computer. TM APIs are provided as C# APIs, and by using them it will be possible to create custom host tools.

psp2ctrl and psp2run are command line tools created using TM APIs. Their source code is provided in %SCE_ROOT_DIR%\PSP2\Tools\Target Manager Server\samples. For details on TM APIs, refer to the "Target Manager API Reference" document.

libdeci4p Library

The libdeci4p library is provided as a method for an application executed on DevKit to obtain data sent to DevKit using TM APIs. For details, refer to the "libdeci4p Overview" document.

Application Execution

The following methods are provided for executing applications on DevKit from the development host computer.

Visual Studio

By selecting **Debug > Start Debugging** or **Debug > Start Without Debugging** in Visual Studio, applications on DevKit can be started.

Neighborhood

Applications can also be executed from Neighborhood. Select DevKit, execute **Load Executable**, and specify the ELF file of the application. In addition, execution is also possible by dragging-and-dropping the ELF file onto the Neighborhood target list.

psp2run

It is possible to execute applications on DevKit by using the psp2run command at the command prompt. For details on the psp2run command, refer to the "Neighborhood and Utilities User's Guide" document.

Debugging and QA

For application debugging, use the debugger integrated onto Visual Studio. Similar to Windows programs, applications executed on DevKit can be debugged through the debugger interface on Visual Studio. For debugger settings, refer to the "Debugger User's Guide" document.

Debugger for PlayStation®Vita

The debugger for PlayStation®Vita is a debugger that has been integrated with Visual Studio. It can debug PlayStation®Vita applications. For details, refer to the "Debugger User's Guide" document.

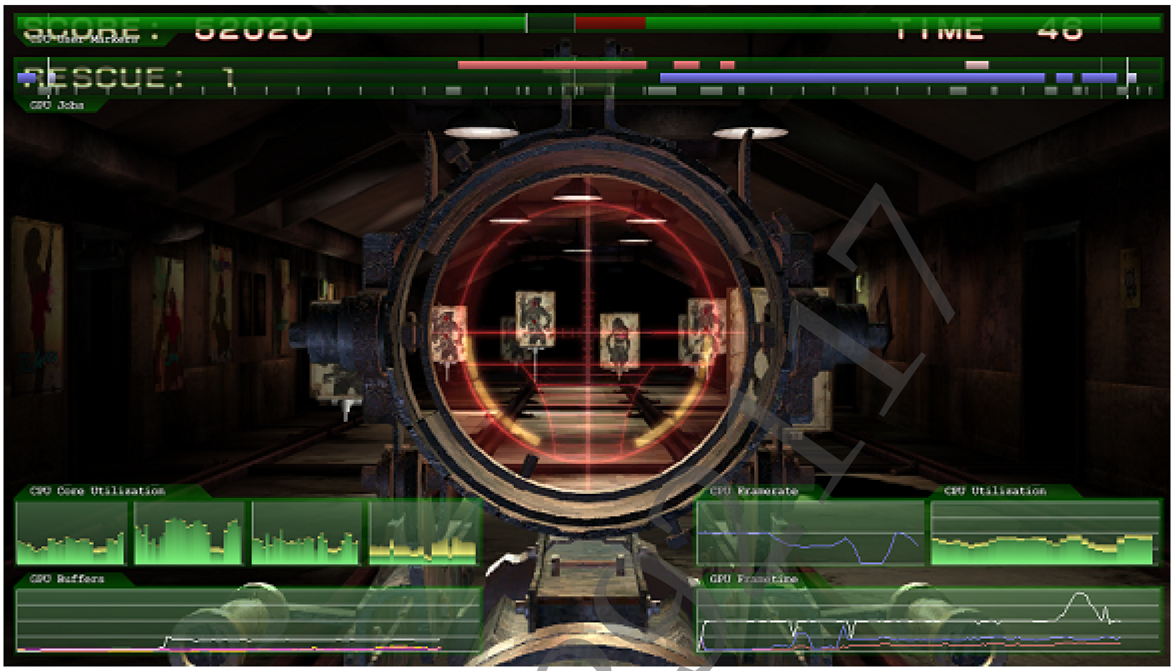
Performance Tuning

Various tools are provided for tuning performance. First, use Razor to tune overall performance. After problems have been narrowed down and to look deeper into a certain problem, consider the use of libperf or DTrace.

Razor

Razor is a performance analysis tool integrated to Visual Studio. By using Razor, CPU data and GPU data during application execution can be captured, and various measurements can be obtained such as, which processing is being performed in the CPU/GPU and how much processing time is entailed for it.

Figure 6 Razor Usage Example (CPU HUD Performance Mode)



For details, refer to the "Razor User's Guide" document. Moreover, the utilization of Razor is exemplified in the "USSE Performance and Shader Optimizations: Tutorial" document.

libperf

libperf is a library that supports a program's performance analysis by using the performance monitor feature embedded in the ARM processor. Features to enable coordination with Razor are also provided - such as, a feature to set markers that can be checked by Razor. For details, refer to the "libperf Overview" document. For the usage example of libperf, refer to the "Algorithms for Sorting on PlayStation®Vita: Tutorial" document.

DTrace

DTrace can be used to check how a game application is running, track and narrow-down performance problems, and identify the cause of abnormal operation. Various monitoring points, referred to as probes, are used to sample system states in specified intervals in order to narrow-down performance problems. For details, refer to the "DTrace Overview" document.

Sulpha Tool

Sulpha Tool works in coordination with NGS to capture and analyze audio information on DevKit.

NGS is a sound synthesizer library for PlayStation®Vita. Sound system catered to the game can be created with flexibility, and various effects - such as, reverberation and filters - are provided.

By utilizing Sulpha Tool, NGS in the application can be used to detect errors of the sound system in games and to swiftly analyze performance. For details, refer to the "Sulpha Tool User's Guide" document.

Testing Operation for the Retail Unit Using DevKit/TestKit

Package File Creation

A package file (pkg file) must be created to install the developed application to DevKit/TestKit. The Publishing Tools are provided to create a package. For details, refer to the "Package Generator User's Guide" document. For instructions on how to install the package, refer to the "Application Development Process Overview" document. For an actual packaging example, refer to the "TRC Compliant Shooting Game Creation Tutorial" and "Basic TRC Compliant Tutorial" documents.

Test Execution in Release Mode

Applications are required to use TestKit and ensure that their implemented features run as expected.

Within the application development process, before testing on TestKit, first test operation on DevKit with **Release Check Mode** set to **Release Mode** in order to detect problems that will occur when running the application on TestKit. DevKit set to **Release Mode** has the same limitations as TestKit regarding reads/writes of host0: and ux0: and DECI communication with the development host computer (for example). If these codes are included in the application, related problems can be detected in advance on DevKit. Moreover, because messages output by the application to TTY can be reviewed on DevKit even in **Release Mode** if **Release Mode Console** is set to **Enable**, it is easier to review problem causes on DevKit rather than on TestKit.

Core Dump

Because the debugger cannot be connected to TestKit, when looking for a cause of an application's abnormal termination on TestKit, review the core file created by core dump. For details on the core dump, refer to the "Core Dump Overview" document.

3 Basic Information and Notes on Programming Environments

This chapter provides basic information and notes on programming environments.

File Format

The file formats of the source files provided with the SDK are as follow.

- Character code: UTF-8 BOM
- Line break code: CR+LF

Because the compiler only supports UTF-8 encoding, UTF-8 with BOM is recommended for the source code in application development.

Variable Size

The size and alignments of the variables provided by the SDK are shown below.

Datatype	Size (Bytes)	Alignment (Bytes)
char	1	1
short	2	2
wchar_t	2	2
int	4	4
long	4	4
long long	8	8
float	4	4
double	8	8
64-bit vector	8	8
128-bit vector	16	8
Pointer Types	4	4

Note
char datatypes without signed/unsigned modifiers are defined, by default, as signed datatypes.

Program Module Format

Single programs loaded onto memory are referred to as program modules, or simply modules.

This platform supports the following two program module formats.

ELF Format

These program modules are loaded first upon process startup. They stay resident in memory from process startup to completion. They are also referred to as main modules. Programs always starts by the `main()` function in this module.

The address to which these modules are loaded is set statically at the build stage.

PRX Format

This is a unique object file defined by kernel specifications, and can be loaded dynamically in processes when desired.

Refer to "libsysmodule Overview" document for details.

SELF/SPRX

The SELF/SPRX file format is an executable file format based on the ELF/PRX file format, which has been digitally signed by SCE and encrypted. When a build is performed using VSI, a SELF format file is automatically created. Before execution, the kernel decrypts a SELF/SPRX format program, verifies digital signature, and confirms it to be a program that has been created by an official licensee and that it has not been tampered with. Thereafter, execution is performed.

Temporary digital signatures are appended to SELF files created during development. The official digital signature is attached by SCE after the master disc has been submitted.

Library Formats

The following two library formats are provided.

Static Link Format

The static link format library is linked to applications at the build phase. The library is statically linked to the application, and reserves a memory area during execution.

Dynamic Link Format (PRX Format)

PRX format libraries are dynamically linked upon execution. When the library is used, it is loaded into memory and linked. When no longer necessary, it can be unloaded to free up memory.

When using PRX format libraries, a stub library (_stub.a) must be statically linked in order to relay function calls.

Handling Cases When the Blue Screen (BSOD) Is Displayed during Application Development

The blue screen (BSOD) is displayed when a fatal error occurs on the system side; for example, a kernel panic. When BSOD occurs, the cause cannot be searched for by the application developer. Save the displayed blue screen (BSOD) by taking a photograph of it (on a digital camera, for example and saving it as an image file). Because the font size on BSOD is small, use a high-resolution digital camera and make sure the text in the saved image file is legible. Furthermore, the display of the blue screen (BSOD) can be toggled using the **L** and **R** buttons; make sure all displays are saved as image files. Note that the system software's screenshot feature cannot be used when the blue screen (BSOD) is displayed.

Subsequently, send the image files to Technical Support along with a description of the situation when the BSOD occurred, how to reproduce it, and if possible, the program in which the BSOD is reproduced. SCE will analyze the problem.

If the program crashes when an invalid data access (for example) occurs on the application side, the problem can be analyzed by using the debugger or core dump in the usual manner.

Error Handling that Must Be Implemented When Calling File Access Functions

When a PlayStation®Vita card is removed, the system will terminate the application. However, if a PlayStation®Vita card is removed at the same time a file-accessing function is called, the application may be notified that this function calling has failed. This means there is a possibility that any function performing file access will fail.

When an application calls a function for performing file access, even if in theory there is no possibility of the function call failing, consider the possibility of failure and implement error processing so that exceptions and runaways (data destruction and unauthorized action towards other players) due to applications do not occur. In particular, be careful when calling the file access functions of the following libraries.

- Kernel IO/File manager (`sceIoXXXX()` functions)
- C/C++ standard libraries
- FIOS2 library

As an error processing implementation method, when function calling for reading data required for an application to continue running fails, it is presumed that it is impossible for the application to continue running. In such cases, the application can be stopped or infinite retries can be implemented. Use a method that can be implemented by the application easily so that the application does not crash.

Media and Access Performance

The media that can be used by PlayStation®Vita are PlayStation®Vita cards and memory cards.

Media performance will vary based on the type of PlayStation®Vita card or memory card and on the time of their launch; however, you can test the access performance of PlayStation®Vita cards or memory cards by setting ★**Debug Settings** > **System** > **Slow Card Mode** in the home screen's Settings application to **Enable**.

Process Time

Each process has a "process time" that is continually counted while the process is running with the start point of the corresponding process as the start (0).

The process time is in microseconds, and it can be obtained using functions such as `sceKernelGetProcessTime()` and used in the programs of applications.

In addition, the process time will also count toward timeout times, etc. set by various synchronization functions.

The process time count will stop when a process is suspended due to the **PS button** or **power button** being pressed.

On the other hand, when a process is paused (debug suspension) by a debugger operation or by the occurrence of an exception, the process time count will not stop.

Therefore, note that timeouts may occur during debugging that do not occur during standard application operation.

PRX Restriction

With the PRX modules that can be loaded in this platform, the segment alignment is a positive power of 2 and must be within a range from 16 bytes to 4096 bytes. Therefore, when using a PRX format module, do not allow the alignment declaration in a program to exceed 4096 bytes.

4 Application Termination Processing

The termination processing of applications is described below.

Application Termination Processing Procedure

In the PlayStation®Vita, the only termination method allowed for applications consists of flipping through the executable LiveArea™ screens all the way to the end, with the following procedure:

(1) When the PS button is pressed, the system suspends the application.

(2) After the application is suspended, the screen changes back to the LiveArea™ screen. The LiveArea™ screen can be flipped through by flicking the LiveArea™ screen from the top right to the bottom left of the screen while LiveArea™ of the application is displayed. Once the screen has been flipped all the way to the end, the application ends.

The above termination method is implemented as a kernel process kill for suspended game processes. Applications cannot detect this termination. Moreover, the following specifications should be noted in the C and C++ standard libraries.

- If the application is suspended while the file is opened, the file may not be accessed by use of the stream after resume, depending on how the file system is mounted.
- After writing to a file, if the application is suspended prior to flashing or closing, buffered data may not be written depending on how the file system is mounted.
- Even in the above case, if the stream is closed with `fclose()` the resources that were in use with the C and C++ standard libraries will be recovered correctly.
- After writing to a file, if the application is killed before flashing or closing buffered data will not be written.
- Calling functions (including global destructor) registered by using `atexit()` is only supported in the DevKit operated in **Development Mode**.

Forbidding Self-induced Application Termination

Technical Requirements Checklist (TRC) forbids applications to perform self-induced processing terminating the application, such as the following:

- Calls `exit()`, `_Exit()`, `abort()`, `terminate()`, `unexpected()`
- `assert()` failure
- Returns from `main()`
- Does not receive a thrown exception object by a catch statement

If the application performs any of the above processing with the **Release Check Mode** of **Boot Parameters** set to **Release Mode**, the system will determine that unexpected behavior has occurred in the application, and display an on-screen message stating that "The application has stopped unexpectedly.", as when an application crashes.

If the **Release Check Mode** of **Boot Parameters** is **Development Mode**, even if the above-mentioned processing leading to termination is performed the system will not treat it as an error, and will not display the message stating that "The application has stopped unexpectedly.". Therefore, it is possible to have self-induced termination of application for testing purposes, etc.

Necessary Measures in Relation to the Application's Termination Processing

It is not possible to detect whether application suspension processing has been performed. However, it is possible to obtain the return of the application itself from suspension as a Resume event. For details, refer to the following documents.

- Application Manager Overview
- Application Manager Reference

Also, calling `sceAppMgrSetNetworkDisconnectionWarningDialogState()` and activating Network Disconnection Warning Dialog will result in a warning dialog being displayed when the application is suspended by pressing the **PS** button while using the network feature.

Even if the application is suspended while a file is being written, the system will guarantee that the file is written completely by using `sceAppUtilSaveDataDataSave()` to save the save data. However, this does not mean that the system guarantees that the save data will not change to broken status; it is therefore necessary to implement separate means of dealing with broken status.

5 Resources that Can Be Used by an Application

The resources that can be used by an application are described below.

CPU

An application can use three of the four provided ARM Cortex-A9 processor cores.

The processing of threads for which low or medium priority is specified is automatically allocated by the thread scheduler of the kernel to the three cores that can be used by the application. Threads for which high priority is specified must be executed by specifying the core to be used for execution. For details on the behavior of the thread scheduler, refer to the following document.

- Kernel Overview

In principle, kernel/system threads are designed to be executed using one core other than the three cores that can be used by an application, and thus an application can use the three cores mostly. However, since some types of processing (e.g. system thread execution, interruption processing) may occur using the three cores that can be used by an application, it is not possible to implement an application on the assumption that the application's thread is never be preempted.

Main Memory (LPDDR2 DRAM)

Main memory that can be used by the application (LPDDR2 DRAM) changes according to **Memory Size** of **Boot Parameters** and the memory expansion mode set by param.sfo.

Boot Parameters

- When **Memory Size** of **Boot Parameters** is **Console Size**
  The main memory that can be used by an application is 282 MiB. 282 MiB is the sum of 256 MiB of ordinary memory and 26 MiB of physically continuous memory.
- When **Memory Size** of **Boot Parameters** is **DevKit Size**
  The main memory that can be used by an application is 538 MiB. 538 MiB is the sum of 512 MiB of ordinary memory and 26 MiB of physically continuous memory.

Moreover, when the **Release Check Mode** is **Release Mode**, the size of main memory that can be used by the application will be set to the same size as when **Console Size** is specified, regardless of this memory size setting.

For details on **Boot Parameters**, refer to the "Development Kit Neighborhood Settings Guide" document.

The memory specified in libcamera or decoding-related libraries must be physically continuous memory. Normal memory and physically continuous memory can each be allocated by specifying the respective memory type to *`type`* of `sceKernelAllocMemBlock()`

For details on memory types, refer to the "Kernel Reference" document.

Memory Expansion Mode

The memory size of the main memory's physical memory to be used by the application can be expanded by setting **Large Memory Mode** of **App Setting(2)** for param.sfo. The memory expansion mode has a +29MiB mode and +77MiB mode. One memory expansion mode is set to an application and cannot be changed dynamically.

Limitations When Using the Memory Expansion Mode

While an application using the memory expansion mode is running, no application besides the Settings application can be started from the home screen. Even if the application can be simultaneously executed with another application such as, the "near", Messages, or Party application, such an application cannot be started from the home screen while the application using the expansion mode is running.

If an application using the memory expansion mode is started while a certain application is running, the running application will be forced to terminate. Moreover, when starting another application (besides the Settings application) while an application using the memory expansion mode is running, a dialog confirming termination of the running application will be displayed.

Limitations When Using the +77MiB Mode

When an application uses the +77MiB mode, it is possible to call the Title Store application from within the application and to execute it simultaneously with the application using +77MiB mode; however, the Modal Internet Browser cannot be called from within the application for simultaneous execution. Do not call `sceAppUtilLaunchWebBrowser(SCE_APPUTIL_WEBBROWSER_LAUNCH_APP_MODAL)` from an application using the +77MiB mode. When starting the Modal Internet Browser from within the application, a dialog confirming application termination will be displayed.

Simultaneous Execution Possibility of Features in Each Memory Expansion Mode

	+29MiB Mode	+77MiB Mode
Title Store application	Yes	Yes
Modal Internet Browser	Yes	No

Main Memory Size that Can Actually Be Allocated by Applications

Although the main memory size that can be used by applications is as described above, this does not mean that applications can use the above memory in its entirety. Note the following memory areas are allocated by the system from main memory before the application's `main()` function is called.

- The necessary memory to load the program of the application (eboot.bin) to be started up
- The system memory reserved by the kernel (3 MiB)
- The memories for libc (312 KiB), libfios (160 KiB) and apputil (44 KiB), preloaded by the system
- The memory allocated to the heap area of C and C++ Standard Libraries (an arbitrary size can be specified)

Also, after start-up, the application cannot allocate all of the remaining memory by using `sceKernelAllocMemBlock()`. Note that the following memory areas will be allocated from the application's memory area:

- The stack area and TLS memory used when creating threads (size specification, 4 KiB units)
- The necessary memory for loading PRX with libsysmodule (depends on the module to be loaded)

Refer to the document below for more information on the memory size to be allocated to C and C++ Standard Libraries.

- Description of "malloc" in "C and C++ Standard Libraries Overview and Reference"

Refer to the document below for the memory size used by the PRX loaded using libsysmodule:

- "libsysmodule Reference"

Refer to the document below for more information on the stack area and TLS memory used when creating threads:

- "Kernel Overview"

Video Memory (CDRAM)

The maximum size of the video memory (CDRAM) that can be used by an application is 112 MiB.

Note
For memory types and sizes, debug techniques related to memory, and comparison of memory performances, refer to the "Memory Management Tutorial" document.

Exclusive Operation Devices

The front camera and rear camera cannot be used simultaneously by an application. Besides this, by using the power configuration control provided by the power service, GPU clock frequency can be accelerated, whereas the use of specific devices is not possible. For details on the power configuration control, refer to the following document.

- Power Service Overview

Number of File Descriptors that Can Be Opened Simultaneously

The number of file descriptors that can be opened simultaneously by an application is 64.

Number of Memory Blocks that Can Be Created

The number of memory blocks that can be created simultaneously by an application is 2048.

Number of Threads that Can Be Created

The number of threads that can be created simultaneously by an application is 128. Do not carry out start/stop processing of the PRX when the number of threads has reached this maximum of 128.

When PRX start processing or stop processing is performed, the kernel creates one thread in order to execute a PRX entry function. When PRX start processing or stop processing is performed in a state where the number of threads has reached the maximum value of 128, the thread creation will fail, and the PRX start/stop processing itself will fail. The behavior when PRX start/stop processing is carried out with the maximum value of 128 reached for the number of threads will be as follows.

(a) When attached with a debugger
`SCE_KERNEL_ERROR_UID_RL_OVERFLOW` will return as an error code. Moreover, "Syscall Critical Usage Error" will be output to the console.

(b) When not attached with a debugger
A core dump will be output. A cause code, "a system call with invalid system call number issued," will be stored in the core dump.

Note
The cause code stored in the core dump may be changed in a future version.

Number of Synchronization Objects that Can Be Created

The number of synchronization objects that can be created simultaneously by an application is as follows:

- Objects without the `SCE_KERNEL_ATTR_OPENABLE` attribute added: 2048
- Objects with the `SCE_KERNEL_ATTR_OPENABLE` attribute added: 256

Number of Identifiers (UIDs) that Can Be Created

The number of identifiers (UIDs) that can be simultaneously created in a process is 6144.

This number represents the total number of identifiers represented as an SceUID type for objects provided by the kernel including file descriptors, threads, memory blocks, synchronization objects, and modules.

Note that identifiers created by system-provided modules that run within processes are also included in this upper limit.

Thus, the actual number of identifiers that can be created by an application is less than 6144.

Note also, that the number of objects will not always match the number of identifiers.

For example, n number of references may be created for one synchronization object. In this case, there will be 1 + n number of identifiers that indicate this synchronization object (one identifier created upon object creation and n number of references).

Regarding object references, refer to the "Kernel Overview" document.

Number of Loadable Modules

An application can simultaneously load 128 modules. This module number also includes modules loaded by the system.

Note that the number of modules loaded can be checked from the debugger.

6 Application Support for PlayStation®TV

Library Operations for PlayStation®TV

Library operations for PlayStation®TV are explained below.

Devices and system applications implemented on PlayStation®TV differ from those on PlayStation®Vita. Because of this, applications supporting PlayStation®TV must take note that there are libraries that behave differently on PlayStation®TV than on PlayStation®Vita as follows.

- Library that returns error codes unique to PlayStation®TV when executed on PlayStation®TV
- Libraries that return error codes that can also occur for PlayStation®Vita when executed on PlayStation®TV
- Libraries that return dummy data upon execution on PlayStation®TV

To check application operation for PlayStation®TV on a DevKit, enable **PS TV Emulation** (**On**) from the Settings application. For details, refer to the "System Software Overview" and "DevKit/TestKit Setup Guide" documents.

Library that Returns Error Codes Unique to PlayStation®TV

When an application is running on PlayStation®TV, error codes unique to PlayStation®TV return when using functions of the following library.

- libcamera

libcamera returns an error code indicating that a camera is not implemented. This error code can be used for the application to determine that it is being executed on PlayStation®TV; however, considering future compatibility, only use this error code to determine the existence/lack of a camera and do not use it to determine other factors.

Libraries that Return Error Codes that Can Also Occur for PlayStation®Vita

When an application is running on PlayStation®TV, functions of the following libraries return error codes that can also occur for PlayStation®Vita.

- "near" utility/"near" Dialog utility
- liblocation

The "near" application and location information cannot be used on PlayStation®TV. Because of this, some functions of the above libraries will return error codes. Handle these errors without differentiating from when they are returned in an application running on PlayStation®Vita.

The features of these libraries cannot, as a result, be used on PlayStation®TV.

Libraries that Return Dummy Data

When an application is running on PlayStation®TV and the following libraries are used, data differing from when the application is executed on PlayStation®Vita is obtained.

- Controller Service
- Touch Service
- libmotion

Some restrictions are exemplified below.

- The Touch Service can obtain inputs from the wireless controller as touch operations when the touchscreen pointer feature of PlayStation®TV is used. The maximum value for the number of points that can be obtained at one time with the touchscreen pointer feature is two.
- Valid data cannot be obtained with libmotion.

Note
In PlayStation®TV, DUALSHOCK®3 wireless controllers, DUALSHOCK®4 wireless controllers, and SIXAXIS™ wireless controllers can be used without distinction (it is not possible for applications to distinguish between such controllers). Therefore, this document refers to them as "wireless controller" unless otherwise required.

Handling the Call of a Feature that Cannot be Used with PlayStation®TV

When a feature that cannot be used with PlayStation®TV is called and an error code is returned from a function, interpret the meaning of the error code as is. The library does not return error codes by which device types - PlayStation®Vita or PlayStation®TV - can be determined; do not implement a design to determine the device type from the returned error code.

For example, if the AR feature is implemented, a camera is required to use this feature. When a libcamera API is called in the PlayStation®TV environment, the `SCE_CAMERA_ERROR_NOT_MOUNTED` (0x802E0010) error code will return. At this time, do not make an announcement such as, "The AR feature cannot be used with PlayStation®TV," which assumes the device type is PlayStation®TV because there is no camera. Interpret the error code as is, and display a message indicating that "The AR feature cannot be used because there is no camera."

UI Design Supporting PlayStation®TV

Requirements and Policies

The main difference between PlayStation®Vita and PlayStation®TV in designing the UI is the existence/lack of touch operations. The screen (touchscreen) and rear touch pad are not installed on PlayStation®TV; thus, applications supporting PlayStation®TV are required to allow control and completion of the game with just a wireless controller and without touch operations (TRC R3171).

On the other hand, in UI design policy, consider provide UIs that do not have clear advantages/disadvantages in PlayStation®Vita or PlayStation®TV. In particular, make sure that clear advantages/disadvantages are not present in competitive games or games with competition for high scores. For example, it is recommended to provide a UI that allows for control with a wireless controller similar to touch operations, as in the system software home screen.

Displaying Control Instructions

Applications must explain operation methods using PlayStation®Vita; explaining operation methods using a PlayStation®TV and a wireless controller are optional for applications. Even in applications supporting PlayStation®TV, an explanation of operation methods using the wireless controller is not required as long as operation methods using PlayStation®Vita is explained.

If operation methods with a wireless controller are displayed with images, display both images of PlayStation®Vita and the wireless controller. Do not just display images of the wireless controller.

Note that it is recommended to change the UI within the scope that can be evaluated from the error code when an error is returned due to calling a feature that cannot be used with PlayStation®TV. For example, it is recommended that changes such as disabling the button for the onscreen AR feature when an error code returns stating that a camera is not present. However, limit changes to a scope that corresponds to the meaning of the error code.

Handling of Touch Operation

To replace the screen (touch screen) or rear touch pad operations in a PlayStation®Vita system with controls performed using a wireless controller, it is recommended to use buttons that do not exist on a PlayStation®Vita system as in the following example.

Example: Assign a zoom operation that uses the rear touch pad on a PlayStation®Vita system to the R3 button on a wireless controller.

Handling of Buttons Unique to Wireless Controllers

By using `sceCtrlGetWirelessControllerInfo()`, it will be possible to determine if a wireless controller is connected or not. In addition, by using `sceCtrlReadBufferPositive2()`, etc., it will be possible to obtain operation of buttons unique to wireless controllers (L2 button/R2 button/L3 button/R3 button). However, make sure that there are not any clear advantages/disadvantages between control with a wireless controller when these buttons are used and control with a PlayStation®Vita.

Handling of Multiple Wireless Controllers

Handling of multiple wireless controllers with PlayStation®TV is performed as follows.

- The data from wireless controllers with Controller Numbers 1, 2, 3, and 4 can be obtained as data for players 1, 2, 3, and 4. It will be possible for users to switch Controller Numbers as needed in the menu that appears by pressing and holding the PS button.
- The touchscreen pointer feature can only be used by the wireless controller with Controller Number 1.
- There are no differences in the data that can be obtained from DUALSHOCK®3 wireless controllers, DUALSHOCK®4 wireless controllers, and SIXAXIS™ wireless controllers. Any type can be used by users. In addition, applications cannot distinguish between types of wireless controllers.

The following two functions are available as methods for applications to detect connection states for multiple wireless controllers.

- `sceCtrlIsMultiControllerSupported()`
  This function determines whether the current environment supports multiple wireless controllers or not. In accordance with an environment that supports multiple wireless controllers or not, it will be possible to perform processing such as switching between a UI for single players and a UI for multiple players.
- `sceCtrlGetWirelessControllerInfo()`
  This function allows for checking if a wireless controller is connected or not for each Controller Number.
  After confirming that a wireless controller is connected, it will be acceptable to provide control operation methods for wireless controllers.

For details on the specifications and usage of these functions, refer to the "Controller Service Reference" and "Controller Service Overview" documents.

How to Enable Application Execution on PlayStation®TV

Whether an application can be started up on PlayStation®TV is determined by a parameter set to param.sfo. When "PS Vita: Bootable, PS TV: Bootable" is set to param.sfo, the application can be executed on PlayStation®TV.

Even if the initially released application package is not bootable on PlayStation®TV, it can be enabled by distributing a patch package later on. Distribute a patch package with "PS Vita: Bootable, PS TV: Bootable" set to param.sfo to enable the application to run on PlayStation®TV. For details on param.sfo, refer to the "Param File Editor User's Guide" document.

The conditions under which an application can be enabled to run on PlayStation®TV are defined in R3171 of TRC 1.5 or later. Moreover, the parameter that must be set is defined in R3172 of TRC 1.5 or later; make sure to check the TRC.

7 Notes on Application Development with Consideration for the Retail Unit PCH-2000 Series

This chapter explains notes on application development with consideration for the PCH-2000 series, the new retail unit of PlayStation®Vita. An application cannot evaluate whether the environment in which it is being booted is the PCH-2000 series; therefore, an implementation where operation automatically changes according to the evaluated device type is not possible. However, implementation can be carried out with consideration for the PCH-2000 series in some respects.

Difference in Screen Color Tonality

The OLED display has been changed to the LCD display for the PCH-2000 series and there is a slight difference in the color tonality of the screen when compared to the PCH-1000 series.

To check the color tonality of the PCH-2000 series during application development, use TestKit (PTEL-2000 series). To check the color tonality of the PCH-1000 series, use DevKit or TestKit (PTEL-1000 series).

There is no feature provided to convert the color tonality of the system software. To set a different color tonality for the PCH-1000 series and the PCH-2000 series, implement a color tonality adjustment feature in, for example, the options menu of the application.

Difference in the Size of the Rear Touch Pad

The size of the rear touch pad differs on the PCH-1000 series than that on the PCH-2000 series. For details on size, refer to the "Hardware Overview" document.

The application cannot obtain size information of the rear touch pad. Implement operation methods that are not problematic regardless of which rear touch pad is used (PCH-1000 series or PCH-2000 series).

To check operability of the PCH-2000 series rear touch pad, use TestKit (PTEL-2000). To test the operability of the PCH-1000 series rear touch pad, use DevKit or TestKit (PTEL-1000).

Handling of the Built-in Memory Card

The PCH-2000 series, TestKit (PTEL-2000 series), and PlayStation®TV are installed with a built-in memory card of 1 GiB. The application cannot determine whether the data storage destination is the built-in memory card or a removable memory card.

The removable memory card and built-in memory card operate mutually exclusively. When a removable memory card is set, the built-in memory card cannot be used.

To move data on the built-in memory card to a removable memory card, use the Content Manager Assistant for PlayStation®Vita DevKit with the removable memory card removed and copy data to the development host computer. Subsequently set the removable memory card and copy data from the development host computer. Regarding data transfer from the built-in memory card of the retail kit, refer to the following "PlayStation®Vita User's Guide".

- http://manuals.playstation.net/document/en/psvita/basic/internalmemory.html

8 Appendix A: Application's Enter Button Assignment and Multilingual Support

Application's enter button assignment and multilingual support are described below.

Application's Enter Button Assignment

Depending on the country/region, the enter button is assigned to either the cross button or the circle button. In the system software, the enter button's assignment is decided uniquely for each country/region where the PlayStation®Vita is sold; in applications, however, the enter button's assignment is left to the application. The assignment of the enter button inside applications should nonetheless maintain consistency – avoid specifications whereby it changes while the application is running.

Note
The assignment of the enter button of Common Dialog called by the application is specified when creating the application's system files or package files. It cannot be specified inside the application. Make sure that the application's assignment of the enter button and Common Dialog's assignment of the enter button coincide.
Refer to the "Param File Editor User's Guide" document for more information on how to determine Common Dialog's assignment of the enter button.

Languages Supported by Applications

An application can freely select the language to be used. However, language display within an application must be consistent. Avoid inconsistent specifications that for example switch the notation from Japanese to English suddenly without explicit user action. Also, an application can obtain the language setting of the system by specifying `SCE_SYSTEM_PARAM_ID_LANG` in the first argument of `sceAppUtilSystemParamGetInt()`. For details, refer to the following document.

- Application Utility Reference

Note
The language of Common Dialog called by the application cannot be specified by the application, as display will change automatically based on the system's language settings and on the Sony Entertainment Network account's language settings.

Implementation examples of the language settings that can be used by an application

Enabling language switching from an option within the application

A language switching menu is provided in the options menu in the application. The application does not reference at all the system's language setting.

Obtaining the system's language setting and using this setting

The application's language display is switched according to the system's language setting obtained using `sceAppUtilSystemParamGetInt()`. If the system's language setting is a language that is not supported by the application, the application selects an appropriate language.

Using the language setting of the system as the default setting value of the language setting menu in the application upon initial startup

A language switching menu is implemented in the options menu in the application. In addition, the first time the application is started up, the language setting of the system obtained with `sceAppUtilSystemParamGetInt()` is used as the default language setting value. The system's language setting being the setting that was selected by the user, the specification matching the default language of the application to the system's means that usually the user does not have to change the language. If the application does not support the system's language setting, the application selects an appropriate language, as described in the previous implementation example.

9 Appendix B: Feature Supporting TRC Compliance of Applications

TRC Check Notifications is provided as a feature that supports verification that applications comply with TRC. This feature's setting method and the displayed system messages are described below.

Setting Method

Start the Settings application on the home screen of the DevKit/TestKit and set ★**Debug Settings** > **System** > **TRC Check Notifications** to **Enable**.

Feature Details

When **TRC Check Notifications** is set to **Enable**, the following system messages are displayed.

Error Code Display Mode of Message Dialog and Save Data Dialog

If Message Dialog or Save Data Dialog is started in the error code display mode, the following system messages are displayed in the top left of the screen.

```
SaveDataDialog:
ErrorCode -
0x[hexadecimal error code set from the application here]

MsgDialog:
ErrorCode -
0x[hexadecimal error code set from the application here]
```

System Defined Message Display Mode of Message Dialog and Save Data Dialog

If Message Dialog or Save Data Dialog is started up in the system defined message display mode, the following system messages are displayed on the top left of the screen.

```
SaveDataDialog:
System Message - [shortened character string of a macro supported in the system
defined message display mode]

MsgDialog:
System Message - [shortened character string of a macro supported in the system
defined message display mode]
```

Any of the following character strings is displayed for each [shortened character string of a macro supported in the system defined message display mode].

Macros Supported in the System Defined Message Display Mode	Character Strings To Be Displayed
`SCE_MSG_DIALOG_SYSMSG_TYPE_TRC_MIC_DISABLED`	MIC DISABLED
`SCE_MSG_DIALOG_SYSMSG_TYPE_TRC_WIFI_REQUIRED_OPERATION`	WIFI REQUIRED OPERATION
`SCE_MSG_DIALOG_SYSMSG_TYPE_TRC_WIFI_REQUIRED_APPLICATION`	WIFI REQUIRED APPLICATION
`SCE_MSG_DIALOG_SYSMSG_TYPE_TRC_EMPTY_STORE`	EMPTY STORE
`SCE_MSG_DIALOG_SYSMSG_TYPE_TRC_PSN_AGE_RESTRICTION`	PSN AGE RESTRICTION
`SCE_MSG_DIALOG_SYSMSG_TYPE_TRC_PSN_CHAT_RESTRICTION`	PSN CHAT RESTRICTION

Clip Board

When the `sceClipboardSetText()` function for setting character strings of the clipboard is called, the following system message is displayed in the top left of the screen.

```
Clipboard:
Set Text
```

Shutter Sound

When the `sceShutterSoundPlay()` function for playing back the still image shutter sound, video start sound, and video end sound is called, the following system message is displayed in the top left of the screen.

```
ShutterSound:
Play "XXXXX" sound
```

During playback of the still image shutter sound, video start sound, and video end sound, IMAGE, VIDEO_START, and VIDEO_END are displayed for the XXXXX part, respectively.

Ad Hoc Connection Mode

When connection is established in ad hoc mode, the following system message is displayed to check for NP Communication ID.

```
Adhoc:
NpCommId=[NP Communication ID]
```

When connection is established in PSPNET ad hoc connection mode, the following system message is displayed to check for ad hoc ID.

```
PSPNETAdhoc:
AdhocId=[ad hoc ID]
```

File System Free Space Insufficiencies

When file system free space becomes insufficient, the following system message is displayed.

```
AppUtil:
Free Space Error -
FS : XXXX KiB
```

When there is no enough free space for Save Data Quota, the following system message is displayed.

```
AppUtil:
Free Space Error -
Quota : XXXX KiB
```

Messages Displayed with Message Dialog and Save Data Dialog When Free Space Becomes Insufficient

When a message indicating insufficient save data free space is displayed with Message Dialog and Save Data Dialog that are started up in the system defined message display mode, the following system messages are displayed on the top left of the screen.

When Specifying SCE_MSG_DIALOG_SYSMSG_TYPE_NOSPACE:

```
MsgDialog:
System Message -
NOSPACE : XXXX KiB
```

When Specifying SCE_SAVEDATA_DIALOG_SYSMSG_TYPE_NOSPACE:

```
SaveDataDialog:
System Message -
NOSPACE : XXXX KiB
```

When Specifying SCE_SAVEDATA_DIALOG_SYSMSG_TYPE_NOSPACE_CONTINUABLE:

```
SaveDataDialog:
System Message -
NOSPC_CONTINUABLE : XXXX KiB
```

Displaying the Location Where savedata0: is to be Mounted

When the application starts up, the location where savedata0: is to be mounted is displayed on the top left of the screen as system messages as follows.

```
AppMgr:
savedata0: [OK,NG] -
[<Path where savedata0: is to be mounted>, SCE_ERROR_ERRNO_ENOENT,
SCE_ERROR_ERRNO_ENODEV, <0x8XXXXXXX>]
```

For `[OK,NG]`, when savedata0: is successfully mounted, "`OK`" will be displayed. When the mount processing fails, "`NG`" will be displayed.

With regard to `[<Path where savedata0: is to be mounted >, SCE_ERROR_ERRNO_ENOENT, SCE_ERROR_ERRNO_ENODEV, <0x8XXXXXXX>]`, the directory path will be displayed when savedata0: is successfully mounted, otherwise the error code macro name or the error code will be displayed.

Display of Save Data Slot Access

If `sceAppUtilSaveDataSlotGetParam()` is called while ★**Debug Settings** > **Game** > **Debug Info** is set to **On**, the following system message will be displayed on the top left of the screen:

```
AppUtil:
Read Slot Status [slot id]
```

The save data slot ID whose parameters are to be obtained will be displayed in `[slot id]`. By setting the slot ID verified here to broken status by using ★**Debug Settings** > **Game** > **Fake Save Data Slot Broken**, it is possible to test the application's behavior when it finds broken save data.

Screenshot Disabled/Enabled Setting

When an application disables screenshot, the following system message is displayed.

```
ScreenShot: Disable
```

When an application enables screenshot, the following system message is displayed. The default value at the time of start-up is set to Enable.

```
ScreenShot: Enable
```

When Functions Requiring Testing Regarding TRC R3146 Are Called

TRC R3146 requires that an error code of a function entailing write process to a memory card be notified using Message Dialog. When an applicable function is called, the corresponding system message will be displayed as follows.

When `scePhotoExportFromData()` is Called:

```
Photo:
Require TRC R3146 test
scePhotoExportFromData()
```

When `scePhotoExportFromFile()` is Called:

```
Photo:
Require TRC R3146 test
scePhotoExportFromFile()
```

When `sceAppUtilPhotoMount()` is Called:

```
AppUtil:
Require TRC R3146 test
sceAppUtilPhotoMount()
```

When `sceAppUtilMusicMount()` is Called:

```
AppUtil:
Require TRC R3146 test
sceAppUtilMusicMount()
```

When `sceMusicExportFromFile()` is Called:

```
Music:
Require TRC R3146 test
sceMusicExportFromFile()
```

When `sceMp4RecCreateRecorder()` is Called:

```
Mp4Rec:
Require TRC R3146 test
sceMp4RecCreateRecorder()
```

When `sceMp4RecInit()` is Called:

```
Mp4Rec:
Require TRC R3146 test
sceMp4RecInit()
```

When `sceMp4RecAddVideoSample()` is Called:

```
Mp4Rec:
Require TRC R3146 test
sceMp4RecAddVideoSample()
```

When `sceMp4RecAddAudioSample()` is Called:

```
Mp4Rec:
Require TRC R3146 test
sceMp4RecAddAudioSample()
```

When `sceMp4RecTerm()` is Called:

```
Mp4Rec:
Require TRC R3146 test
sceMp4RecTerm()
```

When Network Check Dialog is Called

The following system message will be displayed upon calling Network Check Dialog.

```
NetCheckDialog:
[String corresponding to the mode]
```

One of the following will be displayed for [String corresponding to the mode]

Mode	Displayed string
Ad hoc connection mode	MODE_AdhocMode
PSPNET ad hoc connection mode	MODE_PSPAdhocMode
PSN℠ mode	MODE_PSNMode
PSN℠ online mode	MODE_PSNOnlineMode
PlayStation®3 connection mode	MODE_PS3ConnMode

When Enabling/Disabling Network Disconnection Warning Dialog

When calling `sceAppMgrSetNetworkDisconnectionWarningDialogState()` to enable or disable Network Disconnection Warning Dialog, the following system messages will be displayed, respectively.

When Network Disconnection Warning Dialog Is Enabled:

```
AppMgr:
NetworkDisconnection
WarningDialog -
Enable
```

When Network Disconnection Warning Dialog is Disabled:

```
AppMgr:
NetworkDisconnection
WarningDialog -
Disable
```

Set Mode B to Power Configuration of Power Service

When setting Mode B to the power configuration of Power Service, the following system message is displayed.

```
Power:
The network features
have been disabled by
mode B.
```

libgxm

When the following functions of libgxm return a value other than `SCE_OK`, a system message is displayed in the top left of the screen. Note that libgxm only supports DevKits.

- `sceGxmDraw()`
- `sceGxmDrawInstanced()`
- `sceGxmDrawPrecomputed()`
- `sceGxmPrecomputedVertexStateSetTexture()`
- `sceGxmPrecomputedVertexStateSetAllTextures()`
- `sceGxmPrecomputedFragmentStateSetAllTextures()`
- `sceGxmPrecomputedFragmentStateSetTexture()`

The displayed system message is as follows.

```
GXM:
<Name of function for which error occurred>
-[Hexadecimal error code returned by the function]
```

In order for this system message to be displayed, **Use Debug Version of libgxm for Game** must be set to **Enable** along with **TRC Check Notifications**. Set as follows.

- Set from Setting node of Neighborhood
- Start Settings application on the DevKit home screen, and set from ★**Debug Settings > Graphics Library (libgxm)**

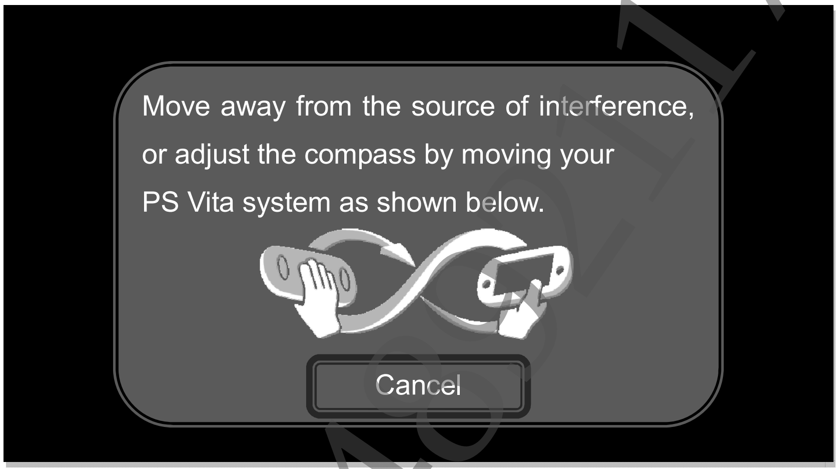
10 Appendix C: Implementation Guidelines for Magnetometer Sensor Calibration Message Dialog

The guidelines for the implementation of the "magnetometer sensor calibration message dialog" by applications are described below.

Overview

Both libmotion and liblocation offer compass (magnetic field) information. Compass information is automatically calibrated by the system software, but the "magnetometer sensor calibration message dialog" shown below is provided for effective calibration.

Figure 7 Screen Shot of Magnetometer Sensor Calibration Message Dialog


Move away from the source of interference, or adjust the compass by moving your PS Vita system as shown below.

For applications that require the accurate use of compass information, it is recommended to implement the "magnetometer sensor calibration message dialog" according to these guidelines. The "magnetometer sensor calibration message dialog" is provided as a system defined message of Message Dialog, and can be displayed by specifying `SCE_MSG_DIALOG_SYSMSG_TYPE_MAGNETIC_CALIBRATION` for the message type.

For details on Message Dialog, refer to the "Message Dialog Overview" and "Message Dialog Reference" documents.

Implementation Guidelines

The stability of the magnetometer sensor is classified into three levels for libmotion and liblocation. Message Dialog display must be controlled according to these state transitions. The stability of the magnetometer sensor is defined in the respective libraries as follows.

Stability of Magnetometer Sensor that Can Be Acquired from libmotion

Constant	Description
SCE_MOTION_MAGNETIC_FIELD_UNSTABLE	The output of the magnetometer sensor is unstable
SCE_MOTION_MAGNETIC_FIELD_STABLE	The output of the magnetometer sensor is stable
SCE_MOTION_MAGNETIC_FIELD_VERYSTABLE	The output of the magnetometer sensor is very stable

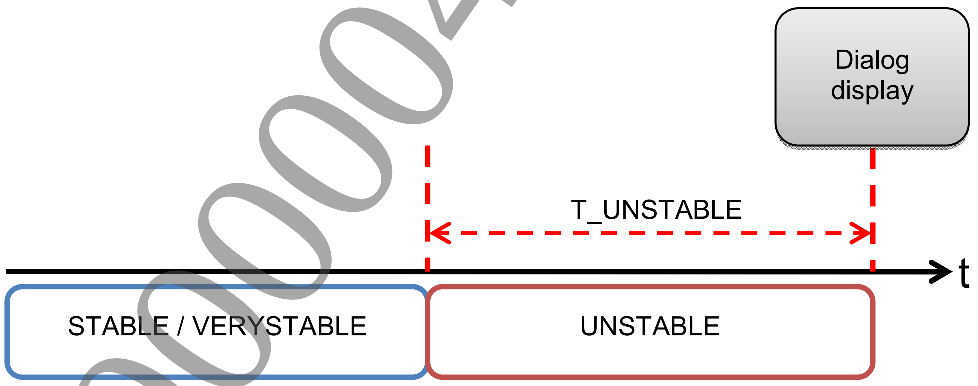
Stability of Magnetometer Sensor that Can Be Acquired from liblocation

Constant	Description
SCE_LOCATION_HEADING_STABILITY_UNSTABLE	The output of the magnetometer sensor is unstable
SCE_LOCATION_HEADING_STABILITY_STABLE	The output of the magnetometer sensor is stable
SCE_LOCATION_HEADING_STABILITY_VERYSTABLE	The output of the magnetometer sensor is very stable

For the method to acquire the stability information of the magnetometer sensor for the respective libraries, refer to the "libmotion Overview", "libmotion Reference", "liblocation Overview" and "liblocation Reference" documents.

Starting Display of the "Magnetometer Sensor Calibration Message Dialog"

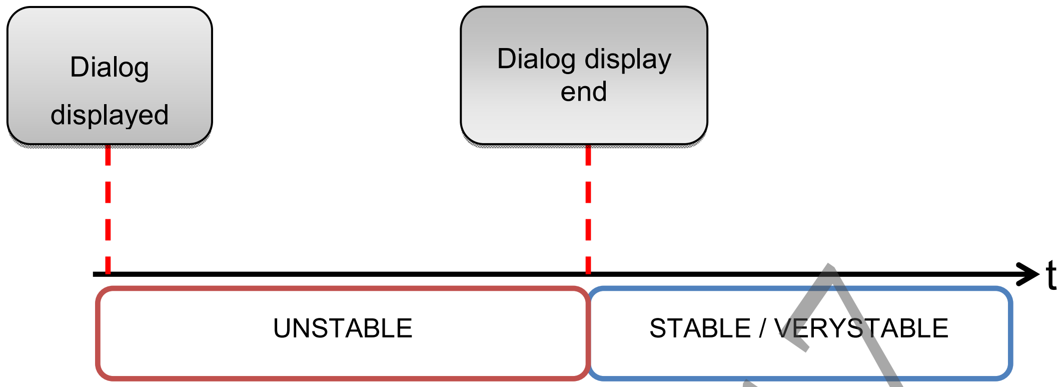
Display of the "magnetometer sensor calibration message dialog" is recommended when the stability of the magnetometer sensor state becomes unstable (UNSTABLE) for a period of T_UNSTABLE or longer.



The recommended value for the T_UNSTABLE period is 1.5 seconds.

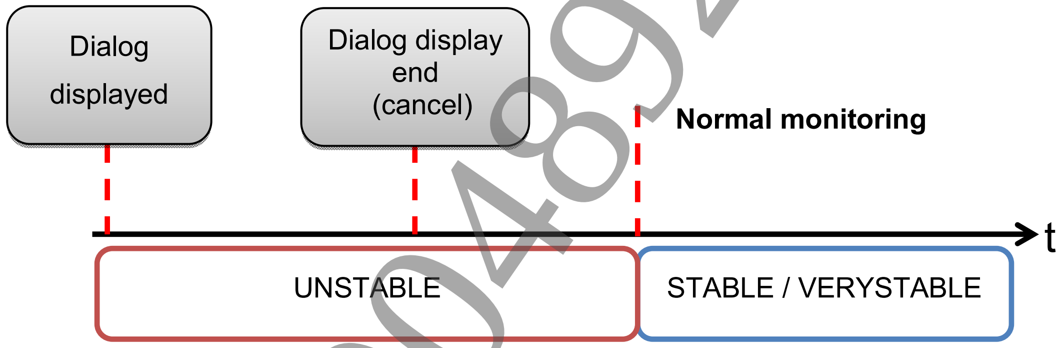
Ending Display of the "Magnetometer Sensor Calibration Message Dialog"

End display of Message Dialog when the stability of the magnetometer sensor becomes stable (STABLE or VERYSTABLE).



Behavior at Cancellation of the "Magnetometer Sensor Calibration Message Dialog"

A **Cancel** button is provided for the "magnetometer sensor calibration message dialog" that allows the user to cancel display of this dialog. In order to prevent chattering after display of the dialog is cancelled, it is recommended to enter again the normal monitoring state after the stable state (STABLE or VERYSTABLE) is returned to.


Dialog displayed
Dialog display end (cancel)
Normal monitoring
t
UNSTABLE
STABLE / VERYSTABLE