libdbg Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

About This Document

The purpose of this document is to provide an overview of the libdbg library and describe its use from a developer's point of view. The libdbg library provides a centralized facility for outputting debug logging messages and testing assertions.

Typographic Conventions

The typographic conventions used in this guide are explained in this section.

Text

File names, source code, and command-line text are formatted in a fixed-width font. For example:

```
samples/sample_code/developer_tools/api_libdbg/basic
```

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:** Example hint.

Notes

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

> **Note:** Example note.

Related Documentation

You can find any updates or amendments to this guide in the release notes that accompany the SDK release packages.

1 Library Overview

Characteristics

libdbg is a library that provides a centralized facility for outputting debug logging messages and testing assertions. The library is intended for use by the licensee but is also utilized internally by SDK libraries to alert the licensee to invalid or potentially problematic conditions.

The library is available for both PSP2 and Windows platforms. Whenever possible, the behavior of the Windows versions of the functions matches the PSP2 versions; however, in some cases the behavior of functions may be different.

All messages output through libdbg (both logging and asserts) are limited to a total length of 512 characters. Messages longer than this will be truncated and either an error will be returned or (where break on error is enabled or if there is a trapping assert) execution will terminate.

Files

The files required to use the libdbg library are as follows:

Table 1 Prerequisite Files

File Name	Description
`target/include_common/libdbg.h`	Header file
`target/lib/libSceDbg_stub.a`	PSP2 Library file

Sample Programs

The following sample program demonstrates the basic usage of libdbg:

`samples/sample_code/developer_tools/api_libdbg/basic`

2 Using the Library

Basic Procedure

(1) Initialization

- PSP2: libdbg is automatically loaded and initialized by the kernel at program startup; therefore no explicit initialization is required by the user.
- Windows: libdbg is implemented entirely in inline functions. No explicit initialization is required.

(2) Logging

Logging messages can be output using the function `sceDbgLoggingHandler`. Users can either directly call this function or use the associated macros to implement their own logging functionality. The function `sceDbgSetMinimumLogLevel` can be used to specify the minimum severity level that should be output through the logging interfaces.

(3) Assertions

Assertion handling can be triggered using the function `sceDbgAssertionHandler`. Users can either directly call this function or use the associated macros to implement their own assertion functionality.

Functionality List

Functions and Macros Related to Logging

Users can pre-define `_SCE_DBG_LOG_BASE` to a suitable function in order to provide custom handling in source-release components.

Table 2 Functions and Macros – Logging

Identifier	Description
`sceDbgSetMinimumLogLevel`	Function to set the minimum logging severity level that the user wishes to view.
`sceDbgLoggingHandler`	Base function to output a logging message.
`sceDbgSetBreakOnErrorState`	Function to control whether a message with severity level `SCE_DBG_LOG_LEVEL_ERROR` should break execution.
`SCE_DBG_LOG_TRACE`	Outputs a verbose logging message.
`SCE_DBG_LOG_DEBUG`	Outputs a diagnostic logging message.
`SCE_DBG_LOG_INFO`	Outputs an informational logging message.
`SCE_DBG_LOG_WARNING`	Outputs a logging message that warns of a potentially problematic condition.
`SCE_DBG_LOG_ERROR`	Outputs a logging message that reports an erroneous condition.

Functions and Macros Related to Asserts

Note: In the current SDK, there is no significant difference in the behavior of `SCE_DBG_ASSERT` and `SCE_DBG_STOP_ASSERT`. However, it is planned eventually to expose differences in debugger behavior for these asserts. In later SDK releases, `SCE_DBG_STOP_ASSERT` will require the user to move the Program Counter manually before resuming execution; `SCE_DBG_ASSERT` will allow the user to resume execution immediately.

Users can pre-define `_SCE_DBG_ASSERT_PRIVATE` to a suitable function in order to provide custom handling in source-release components.

Table 3 Functions and Macros – Asserts

Identifier	Description
sceDbgAssertionHandler	Base function that provides assertion functionality.
SCE_DBG_ALWAYS_ASSERT	If the supplied condition evaluates to false, always halts execution regardless of the value of SCE_DBG_ASSERTS_ENABLED.
SCE_DBG_SIMPLE_ASSERT	If SCE_DBG_ASSERTS_ENABLED is true and the supplied condition evaluates to false, halts execution. No message is output.
SCE_DBG_ASSERT	If SCE_DBG_ASSERTS_ENABLED is true and the supplied condition evaluates to false, halts execution and outputs a simple message based on the condition.
SCE_DBG_STOP_ASSERT	If SCE_DBG_ASSERTS_ENABLED is true and the supplied condition evaluates to false, halts execution and outputs a simple message based on the condition.
SCE_DBG_WARN_ASSERT	If SCE_DBG_ASSERTS_ENABLED is true and the supplied condition evaluates to false, outputs a simple message based on the condition without halting execution.
SCE_DBG_ASSERT_MSG	If SCE_DBG_ASSERTS_ENABLED is true and the supplied condition evaluates to false, halts execution and outputs a user-supplied message.
SCE_DBG_VERIFY	If SCE_DBG_ASSERTS_ENABLED is true and the supplied condition evaluates to false, halts execution and outputs a simple message. If SCE_DBG_ASSERTS_ENABLED is false, the supplied condition is evaluated anyway.
SCE_DBG_VERIFY_MSG	If SCE_DBG_ASSERTS_ENABLED is true and the supplied condition evaluates to false, halts execution and outputs a user-supplied message. If SCE_DBG_ASSERTS_ENABLED is false, the supplied condition is evaluated anyway.
SCE_DBG_STATIC_ASSERT	If the supplied compile-time condition evaluates to false, produces a compiler error.

Macros Related to Breakpoints

Users can pre-define SCE_BREAK and related macros in order to provide custom breakpoint behavior in source-release components.

Table 4 Macros – Breakpoints

Identifier	Description
SCE_BREAK	Breaks program execution. If a debugger is attached, the user can resume execution immediately.
SCE_STOP	Stops program execution. If a debugger is attached, the user must move the Program Counter (PC) before resuming execution.
SCE_NORETURN_STOP	Stops program execution. If a debugger is attached, the user must move the Program Counter (PC) before resuming execution. Currently this behaves the same as SCE_STOP, but in a future release the compiler may generate code based on the assumption that execution will not resume after reaching the resultant breakpoint.