Core Dump Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 About This Document

This document describes the core dump feature and core file upload feature.

The core dump feature outputs information that is useful for identifying the cause when a program generates an exception. This output information includes the memory content, context, and thread state at the time of the exception. By using this feature and analyzing the core file, the status of the program can be examined at the timing at which the exception occurred, without requiring the debugger to be connected to the executed program.

The core file upload feature performs a supplementary feature for the core dump feature that sends the generated core file to a specified web server. By using this feature, it will be possible to omit the procedure for obtaining the core file from the memory card, and constructing a system where core files are collected in a test environment with multiple Development Kits (DevKits) or Testing Kits (TestKits), etc. will be possible.

Reference Materials

This document requires a basic understanding of the kernel. For information regarding the processes and threads created by the kernel, PRX, and the parameters used by synchronous objects, refer to the following documents.

- Kernel Overview
- Kernel Reference

In addition, with the core dump a library is provided for embedding arbitrary data from a program as part of the core file. For details, refer to the following documents.

- Coredump Library Overview
- Coredump Library Reference

2 Setup

If the setup of the DevKit or TestKit is complete, there is no special setup required for using the core dump feature.

For details regarding the setup of the DevKit or TestKit, refer to the following documents.

- DevKit/TestKit Setup Guide
- Development Kit Neighborhood Settings Guide

3 Core Dump Feature

This chapter explains the core dump feature.

Purpose and Features

The core dump feature obtains information that is useful for analyzing the cause of a detectable exception - processor exceptions occurring in applications (game applications and system applications) as well as processor exceptions occurring in the kernel - and outputs the information as a core file. This feature is provided as a debugging feature, to be used mainly from the latter stages to the final stages of application development. With this feature, you can do more than just analyze bugs that occur during QA in game applications being developed. Because crash data for the system software is output as a core file protected by the system's encryption and electronic signature, you can request analysis of system problems by sending the core file of the system application to the technical support on the PlayStation®Vita Developer Network (https://psvita.scedev.net/) ("PlayStation®Vita DevNet") and receive feedback. A core file will still be generated after title launch when specific exceptions for which a core file should be generated occur on PlayStation®Vita. This core file is protected by the system's encryption and electronic signature and will be stored on the server via the Crash Reporting System. For details on the Crash Reporting System, refer to the "Crash Reporting System Overview" document.

Also, because the core dump feature is always active, you do not need to enable or disable the feature. However, depending on whether DevKit is set to **Development Mode** or **Release Mode**, the conditions by which the core dump feature outputs a core file differ. Moreover, the conditions by which a core file is output on the TestKit are the same as the **Release Mode** of DevKit

*__Development Mode__ and **Release Mode** settings can be changed with Neighborhood for PlayStation®Vita (Neighborhood). For details of the each mode, refer to the "Development Kit Neighborhood Settings Guide" document.

Setting to Output Core Files

When DevKit is set to Development Mode

In order to output core files, the following five methods are provided.

- **Select Save as from the Debug menu of Visual Studio**
  When a program is being executed in debug mode from Visual Studio and the attached process is already stopped, select **Save as** from the **Debug** menu to save the core file to a specified location. **Minidump (*.**psp2dmp**)** and **Fulldump (*.**psp2dmp**)** can be selected as save options from the dropdown list.
- **Use pdump of psp2ctrl**
  The **pdump** command of psp2ctrl, which is a command line utility, can be used to forcibly trigger the core dump feature for the process ID specified with that command and output the core file. When executing this command, the specified process must be attached and stopped.
- **Use `InitiateCoredump()` of the Target Manager API**
  `InitiateCoredump()`, which is an API provided by Target Manager Server, can be used to create an independent program that can dump the core from the development host computer at the timing desired. Use this API, for example, to load a program automatically from the development host computer for an automatic test tool on the development host computer, create a core file automatically when an exception occurs, and collect it by the development host computer.

- **Enabling "Forced Dump on Devmode" in Neighborhood**
  When this feature is enabled, the core dump feature behaves the same as when the DevKit is set to **Release Mode**, even when DevKit is set to **Development Mode**.
  Therefore, in a running application that is not attached by the debugger, the core file is output automatically when an exception that is detectable by the core dump feature occurs.
  The progress bar is displayed on the screen during core file output, and the core file is output automatically with the same behavior as **Release Mode**.
  When a running program is attached by the debugger, even when this feature is enabled, control is passed to the debugger, therefore, the core dump feature does not output the core file automatically.
- Select ★**Generate Core File** from the ★**Debug Utility** menu
  If the application is in the suspended state, the ★**Generate Core File** feature provided by ★**Debug Utility** can be used to output the application's core file at any time.
  The file path of the output core file will be displayed in a dialog after the core file output process completes.

When DevKit is set to Release Mode or for TestKit

In order to output core files, the following two methods are provided.

- When specific exceptions that can be detected by the core dump feature occur while the application is running
  When specific exceptions that can be detected by the core dump feature occur while an application is running, the core file is output to a specified location in the order the output destination was detected. At that time, the progress bar is displayed on the screen during core file output, and if output is successful, the core file path at output is displayed on the screen. To return to LiveArea™ after confirmation, press the PS button. The PS button can also be used to cancel core file output.
  For file output destination of the core file and exceptions that can be detected by the core dump feature, refer to the sections "Output Destination of the Core File" and "Exceptions for Which a Core File Will Be Output" in this document.
- Select ★**Generate Core File** from the ★**Debug Utility** menu
  This method is the same as the one explained in the "When DevKit is set to Development Mode" item.

Setting the Core Dump Feature

The default operation of the core dump feature can be changed by changing the following settings.

Dump Level

Use Dump Level to change the dump level (content level) of core files output by the core dump feature. For details on each Dump Level, refer to the "Dump Level" section in this document.

Item	Description
Mini Dump	Records only the bare essential information and memory content. (Default value)
Full Dump	Records all information and memory content that can be output by the core dump feature.

This setting can be changed with the debug settings of Neighborhood and the system software.

Host Path

Use Host Path to specify the output path of the core file on the development host computer. (When this is not specified, core files are output to the location indicated by host0:.)

This setting can be changed only with the debug settings of Neighborhood. Specify up to 255 characters in UTF-8 for the path. Specify the directory path of the development host computer to which to output the core files.

Specification example:

```
c:\work\corefile\
```

This setting can only be used to specify the output destination on the development host computer. It cannot be used to specify other output destinations, such as an internal flash drive on the target. It also cannot be used to specify the core file name. This setting is valid only when DevKit is set to **Development Mode**. This setting is always disabled on TestKit.

Output Destination of the Core File

The output destination of the core file is determined automatically by checking the usable output destinations in the following order.

When a development host computer is connected

When a development host computer is connected to the DevKit and a file system on the development host computer can be used, the output destination is determined in the following order.

- When the path of the development host computer is specified to the "Host Path" debug setting: Files are output to the directory path of the specified development host computer.
- When the path of the development host computer is not specified to the "Host Path" debug setting: the core file is output to the following location.

```
host0:psp2core-A-B-C.psp2dmp
```

*To use the development host computer as the output destination of the core file, not only do the DevKit and the development host computer have to be connected by USB, but an explicit connection must be established with **Connect** of Neighborhood to the DevKit, or the connect command from psp2ctrl. Use the "Connection Status" of Neighborhood or the list command of psp2ctrl to check whether the DevKit is connected to the development host computer.

When a memory card is inserted

When a memory card is inserted in the DevKit or TestKit, the following location is used as the output destination of the core file.

```
ux0:data/psp2core-A-B-C.psp2dmp
```

*To reference a core file recorded in a memory card, use psp2ctrl, which is a command line utility or use Windows File Explorer by installing a file system driver. For details on commands related to access to a memory card, refer to the "Neighborhood and Utilities User's Guide" document included in the Neighborhood Help package and Target Manager Server Help package. In addition, for the memory card, refer to the "Application Development Process Overview" document.

*If the core file is output to the memory card on TestKit, insert that memory card to DevKit to apply the above procedure.

When an internal memory card is used in a TestKit (PTEL-2000 series)

When a memory card is not inserted in a TestKit (PTEL-2000 series), a core file will be saved to the internal memory card. In order to use the core file saved to the internal memory card, it must be copied to an external memory card. The copy procedure for memory cards is as follows.

(1) Insert a memory card into the TestKit (PTEL-2000 series) where a core file has been saved to the internal memory card and reboot the TestKit

(2) Start the Settings application

(3) Select ★**Debug Settings** > **Core Dump** > **Copy**

(4) Check the checkbox for the core file to be copied to the external memory card

(5) Tap the **Copy** button

The core file copied to the memory card can be used with the procedure explained in the "When a memory card is inserted" section.

If neither of the above can be used, a core file is not output.

Core File

The core file output by the core dump feature may differ in a filename or being security protected at output depending on whether the target application is application being developed, an authored application, or the system software. This section identifies and describes both of these in the case that these are different.

The core files of an authored application or the system software are output to the same location as the core files of the application being developed. The core files are security protected with the original encryption, electronic signature etc., and cannot be read by a debugger. To have system software issues analyzed, send the core file to the PlayStation®Vita DevNet technical support.

Filename

The core file of the application being developed is output with a filename in the "**psp2core-A-B-C.psp2dmp**" format. **A** represents the elapsed time (in seconds) from Epoch (1970/01/01 00:00:00 UTC), **B** represents the core dump target process ID (hexadecimal notation), and **C** represents the name of the program loaded onto the process.

Example: psp2core-1167609793-0x00020a5d-EBOOT.BIN.psp2dmp

The core filename changes according to the execution environment (timing of the exception, process ID, and program name), and a new file is created each time. Take note of the free space at the core file output destination.

While the core file is being created, the "**.tmp**" extension will be attached to the end of the filename, as in, "psp2core-A-B-C.psp2dmp**.tmp**". This extension will be removed once the creation of the core file completes.

On the other hand, the core file of an authored application or the system software is output with a filename in the "**psp2core-D.spsp2dmp**" format. **D** represents the application name. This core file differs from the core file of the application being developed in that only one core file is output for each application. While the core file is being created, the "**.tmp**" extension will be attached to the end of the filename, as in, "psp2core-D.spsp2dmp**.tmp**" as well as the core file of the application being developed.

When a GPU exception occurs, however, a core file with a filename that differs from the above rule is created. For details, refer to the "Core Dump When a GPU Exception Occurs" section in this document.

File Format

The file format is extended uniquely based on the ET_CORE-type ELF file format and is then compressed. A single core file is created per process. The core file contains memory content that was mapped to the process address space; information of the register, and synchronization object; the information of thread. In addition, the system core file is output while being security protected by the system's encryption and electronic signature.

The host debugger supports reading the core file of the application being developed. In addition, unique host applications supported for this core file format can be created by using TMAPI provided by the Target Manager.

The host debugger does not support reading the core file of an authored application or the system software.

File Size

The file size is usually between several 100 kilobytes and several megabytes for Mini Dump and between several 10 megabytes and several 100 megabytes for Full Dump, but it varies depending on the compression effect and program status (resource usage of memory, threads, and synchronous objects) when exceptions detected by the core dump feature occur.

Exceptions for Which a Core File Will Be Output

The core dump handler outputs a core file when it detects the following exceptions.

Exceptions occurring in the game application

- Occurrence of prefetch abort
- Occurrence of data abort
- Execution of undefined command
- Occurrence of fpu/VFP interrupt
- Occurrence of fpu/NEON interrupt
- Detection by the system of an invalid system call
- Detection by the system of a system call from an invalid context (for example, from within a callback)
- Occurrence of a stack overflow
- Detection by the system of a breakpoint
- Self-induced process termination
  - Call `exit()`, `_Exit()`, `abort()`, `terminate()`, `unexpected()`
  - `assert()` failure
  - Return from `main()`
  - Does not receive a thrown exception object by a catch statement
- Occurrence of GPU exception

Core Dump When a GPU Exception Occurs

The core dump when a GPU exception occurs differs from other core dumps in the following ways.

Core file output screen

When a GPU exception occurs, "GPU Driver detects GPU Crash" appears on the game screen without displaying the screen that usually appears during core file output. After a core file is output, the system enters standby state.

Core file name

Since it is possible that the application that caused the GPU exception cannot be identified, the application name, program name, and process ID are not be included in the core file name. The core file is output with the following format.

- When an application being developed was active when the exception occurred

```
psp2core-A-GPUCRASH.psp2dmp
  "A" is the time elapsed (sec) since the epoch (1970/01/01 00:00:00 UTC).
```

- When an authored application or system application was active when the exception occurred

```
psp2core-GPUCRASH.spsp2dmp
```

Core file analysis

The core file created when a GPU exception occurs can be read by a debugger in the same way as other generated core files, and this can be used to obtain information about the processes that were active when the exception occurred. Currently, however, there is no supported method for obtaining the information related to the GPU from the core file. To obtain more detailed information related to the GPU for debugging, send the core file using DevNet Private support.

Reporting Security-Protected Core Files to SCE

If an authored application or system application crashes, when a core file protected by security processing is generated, dialog for selecting whether or not to report this file to SCE will be displayed. Moreover, a security-protected core file will also be generated when the kernel crashes, and a dialog to select whether to report the core file to SCE will be displayed after restart. When reporting the file is selected, the core file will be sent to SCE. Any core file sent to SCE may be used by SCE to improve system software quality. SCE uses the core file generated upon a system application crash or kernel crash to fix a bug of the system application or kernel. In a similar manner, the core file of a game application can be used by the game developer via the Crash Reporting System to fix a bug or to improve quality. Regarding the Crash Reporting System, refer to the "Crash Reporting System Overview" document.

There may be data on the title currently being developed included in the core file that is sent. SCE will use this core file only for the system software quality improvement purposes.

The reported data will be used by SCE for learning/analyzing the frequency of occurrence for the problem. To request problem analysis for the system, separately send the core file using PlayStation®Vita DevNet Private support.

Data in the Core File

A core file contains the following core dump data.

Core Dump Data	Description
Core file data	Format version of the core file, etc.
System data	PSID of the system that created the core file, SDK version of the system software, etc.
Process data	State of the process, parent process ID, path name of program loaded in the process, Fingerprint data, etc.
Thread data	ID, state, name, stack address, exception information, etc. of each thread in the process, etc.
Thread register data	Register content of each thread in the process
PRX data	Names, attributes, versions, segment data, Fingerprint data of each PRX in the process, etc.
Library data	Names of libraries being used by the process, functions being exported, information on variables, etc.
Process memory data	Memory content that was mapped to the process address space (memory dump data)
Semaphore data	ID and attribute of each semaphore in the process, maximum and current values of the semaphores, list of thread IDs waiting for the semaphores, etc.
Event flag data	ID and attribute of each event flag in the process, event flag bit pattern, list of all waiting threads, etc.
Mutex data	ID and attribute of each mutex in the process, ID of the thread with the mutex lock, list of thread IDs waiting for the lock, etc.
Lightweight mutex data	ID and attribute of each lightweight mutex in the process, ID of the thread with the lightweight mutex lock, list of thread IDs waiting for the lock, etc.
Message pipe data	ID and attribute of each message pipe in the process, list of thread IDs waiting for sending/receiving, etc.
Callback data	ID and attribute of each callback in the process, notification count of delayed callback, etc.
Timer data	ID, attribute and current value of each timer in the process, etc.
Condition variable data	IDs and attributes of the condition variables in the process, ID of the associated mutex, list of waiting thread IDs, etc.
Lightweight condition variable data	IDs and attributes of the lightweight condition variables in the process, ID of the associated lightweight mutex, list of waiting thread IDs, etc.
Reader/writer lock data	ID and attribute of reader/writer lock in the process, list of waiting read thread and write thread IDs, etc.
Simple event data	IDs for each simple event included in the process, attributes, list of thread IDs waiting for an event, etc.
Stack data	Maximum stack size used, current stack size used, etc., of each thread in the process, etc.
Memory block data	ID, name, type, base address, size of each memory block in the process, etc.
File data	File descriptor of a file whose process is open, open mode, created mode, file pointer value, file size, file path, etc.
Application data	Title ID of the process, title name, application version, and version of the SDK that did the build
External process data	State of other processes that were running at the same time of the crash, parent process ID, pathnames of programs loaded by the process, fingerprint information, etc.

Core Dump Data	Description
User data	Arbitrary data that the process wrote with the core dump handler in the program
User level thread synchronization object data	ID, name, attributes, etc., of each user level thread synchronization object included in a process
TTY output log data	Of the TTY output by the system, log data of the last several KB outputs
Screenshot data	Screenshot of the process upon outputting core dump
GPU data	GPU-related information

Dump Level

The following are the two types of core files according to the output level (content volume).

- **Mini Dump**
  This is the default output level. It records only the bare essential information and memory content. The output time is faster and the output file size is smaller than that of a full dump, but because the output information is limited, some information may not be able to be used during debugging. In particular, a memory dump outputs only a part of an area. Refer to "Comparison of Information Output for a Mini Dump and Full Dump" in this document for details.
- **Full Dump**
  It records all information and memory content that can be output by the core dump feature. Because nearly all mapped memory areas of the process memory are dumped, this is useful for debugging; however, the output time and output size is larger than of a mini dump.

Comparison of Information Output for a Mini Dump and Full Dump

The following table provides a comparison of the information output for a mini dump and full dump.

Core Dump Data	Mini Dump	Full Dump	Note
Core file data	Yes	Yes	No difference
System data	Yes	Yes	No difference
Process data	Yes	Yes	No difference
Thread data	Yes	Yes	No difference
Thread register data	Yes	Yes	No difference
PRX data	Yes	Yes	No difference
Library data	Yes	Yes	No difference
Process memory data	Yes*	Yes	In a mini dump, the stack areas of all threads included in a process and related section information are recorded. In addition, some of the areas (mapped in processes) which were possibly referred to using pointers by threads that were the cause of core file output will also be dumped. In a full dump, in addition to the information output in a mini dump, the areas mapped in a process, including the text, data area, and heap area, are also dumped.
Semaphore data	Yes	Yes	No difference
Event flag data	Yes	Yes	No difference
Mutex data	Yes	Yes	No difference
Lightweight mutex data	Yes	Yes	No difference
Message pipe data	Yes	Yes	No difference
Callback data	Yes	Yes	No difference
Timer data	Yes	Yes	No difference
Condition variable data	Yes	Yes	No difference
Lightweight condition variable data	Yes	Yes	No difference

Core Dump Data	Mini Dump	Full Dump	Note
Reader/writer lock data	Yes	Yes	No difference
Simple event data	Yes	Yes	No difference
Stack data	Yes	Yes	No difference
Memory block data	Yes	Yes	No difference
File data	Yes	Yes	No difference
Application data	Yes	Yes	No difference
External process data	Yes	Yes	No difference
User data	Yes	Yes	No difference
User level thread synchronization object data	Yes	Yes	No difference
TTY output log data	Yes	Yes	No difference
Screenshot data	No	Yes	Recorded only in a full dump
GPU data	Yes	Yes	No difference

Host Tool Version Required for Core File Reading

Target Manager Server

Update the Target Manager Server version to the same system software version or later as that of the DevKit or TestKit that created the core file. If the same system software version or later has not been released, update to the latest version.

The core dump may not be able to be read on the Target Manager Server with an older version.

4 Core File Upload Feature

By using the core file upload feature, the generated core file can be sent to a specified web server. In addition, arbitrary comments input by developers can be sent at the same time as the core file.

With this feature, it will be possible to omit the procedure for obtaining the core file from the memory card, and it will also be possible to construct a system where core files are collected in a test environment with multiple DevKits or TestKits, etc.

Only the protocol (HTTP Post) that the DevKit or TestKit uses to send the core file is regulated with this feature. Developers must set up a web server for receiving sent core files.

Setting the Core File Upload Feature

The settings of the core file upload feature can be changed by changing the following items. These settings can be changed with the debug settings of Neighborhood and the system software.

Enable Uploader

Sets the core file upload feature to enabled/disabled.

Item	Description
On	Enables the core file upload feature.
Off	Disables the core file upload feature. (Default value)

Uploader URL

Specifies the URL of the web server for sending the core file. Up to 255 UTF-8 characters can be specified for the URL.

Specification example:

```
http://uploadserver.com/corefile_receiver.php
```

Auto Upload

By turning **Auto Upload** to **On**, the confirmation dialog and comment input dialog will not be displayed before sending, and the core file will be automatically sent after the core dump.

Item	Description
On	Automatically sends the core file without displaying the send confirmation prompt.
Off	Displays the send confirmation prompt before sending. (Default value)

Core File Sending Method

(1) When an application crashes and a core file is generated, dialog for whether or not to send the core file will be displayed, and the two buttons **Report This Problem** and **Do Not Report** will be displayed. To send the core file, press the **Report This Problem** button. If the core file upload feature is disabled, the **Report This Problem** button will be disabled.

(2) When **Report This Problem** is selected, the comment input dialog will be displayed. Comment input is optional. Input comments will be sent to the web server at the same time as the core file. Click the **Yes** button in the displayed dialog to perform the sending.

Core File Sending Format

The sent data is multipart data with up to two parts and is sent using HTTP POST. The following indicates the MIME headers for the total POST data and both parts, and indicates the data content.

Total POST Data

Item	Description
Content-type header	multipart/form-data; boundary=(arbitrary character string)

First Part

The core file is stored in the first part. The core file name is specified in the Content-Disposition filename parameter.

Item	Description
Content-Disposition header	form-data; name="corefile"; filename="psp2core-xxxxx.psp2dmp"
Content-type header	application/octet-stream
Data	Sent core file

Second Part

Comments are stored in the second part as UTF-8 format text. A character string with the extension ".psp2dmp" subtracted from the corresponding core file name and "-comment.txt" added is specified in the Content-Disposition filename parameter. This part will not be generated if no comments are input.

Item	Description
Content-Disposition header	form-data; name="comment"; filename="psp2core-xxxxx-comment.txt"
Content-type deader	text/plain
Data	Comment