

DTrace Reference

© 2013 Sony Computer Entertainment Inc.
All Rights Reserved.
SCE Confidential

Table of Contents

DTrace Command Line Interface	3
CLI Options	4
SCE DTrace Script Actions and Variables	8
SCE DTrace Script Actions	9
SCE DTrace Script Variables	15
DTrace Provider Probes.....	17
Summary of Probes and Associated Providers	18
interrupt Provider Probes	20
entry	21
return	22
vblank	23
sched Provider Probes	25
wait	26
User-Level Statically Defined Tracing Provider	27
usdt.....	28

DTrace Command Line Interface

CLI Options

DTrace dynamic tracing compiler and tracing command

Definition

```
dtrace [-eFlSvVwZ] [-b bufsz]
        [-o output] [-s script]
        [-x opt[=val]]

        [-P provider[[predicate]action]]
        [-m [provider:]module[[predicate]action]]
        [-f [[provider:]module:]func[[predicate]action]]
        [-n [[provider:]module:]func:]name[[predicate]action]]
        [-i probe-id[[predicate]action]] [args...]

predicate -> '/' D-expression '/' action -> '{' D-statements '}'
```

Options

Option	Argument	Description
-b	<i>bufsz</i>	Set trace buffer size. The trace buffer size can include any of the size suffixes k, m, g, or t. If the buffer space cannot be allocated, dtrace attempts to reduce the buffer size or exit depending on the setting of the buffer size.
-e		Exit after compiling request but prior to enabling probes. This option can be combined with D compiler options to verify that the programs compile without actually executing them and enabling the corresponding instrumentation.
-f	[[<i>provider:</i>] <i>module:</i>] <i>func</i> [[<i>predicate</i>] <i>action</i>]]	Enable or list probes matching the specified function name. The corresponding argument can include any of the probe description forms <i>provider:module:function</i> , <i>module:function</i> , or <i>function</i> . Unspecified probe description fields are left blank and match any probe regardless of the values in those fields. If no qualifiers other than <i>function</i> are specified in the description, all probes with the corresponding <i>function</i> are matched. The -f option can be suffixed with an optional D probe clause. You can specify more than one -f option on the command line at a time. (See below.)
-F		Coalesce trace output by function. Function entry probe reports are indented and their output is prefixed with ->. Function return probe reports are not indented and their output is prefixed with <-. System call entry probe reports are indented and their output is prefixed with =>. System call return probe reports are not indented and their output is prefixed with <=.
-i	<i>probe-id</i> [[<i>predicate</i>] <i>action</i>]	Enable or list probes matching the specified probe ID. You can specify probe IDs using decimal integers; for example: dtrace -1. The -i option can be suffixed with an optional D probe clause. You can specify more than one -i option at a time. (See below.)

Option	Argument	Description
-l		List (rather than enable) all probes that match specified criteria.
-m	<i>[provider:]module[[predicate]action]</i>	Enable or list probes matching the specified module name. The corresponding argument can include any of the probe description forms <i>provider:module</i> or <i>module</i> . Unspecified probe description fields are left blank and match any probe regardless of the values in those fields. If no qualifiers other than <i>module</i> are specified in the description, all probes with the corresponding <i>module</i> are matched. The -m option can be suffixed with an optional D probe clause. You can specify more than one -m option on the command line at a time. (See below.)
-n	<i>[[[provider:]module:]function:]name[[predicate]action]</i>	Enable or list probes matching the specified probe name. The corresponding argument can include any of the probe description forms <i>provider:module:function:name</i> , <i>module:function:name</i> , <i>function:name</i> , or <i>name</i> . Unspecified probe description fields are left blank and match any probe regardless of the values in those fields. If no qualifiers other than <i>name</i> are specified in the description, all probes with the corresponding <i>name</i> are matched. The -n option can be suffixed with an optional D probe clause. You can specify more than one -n option on the command line at a time. (See below.)
-o	<i>output</i>	Set the output file for the -l option or for the traced data itself. The default output file is <i>d.out</i> .
-P	<i>provider[[predicate]action]</i>	Enable or list probes matching the specified provider name. The remaining probe description fields <i>module</i> , <i>function</i> , and <i>name</i> are left blank and match any probes regardless of the values in those fields. The -P option can be suffixed with an optional D probe clause. You can specify more than one -P option on the command line at a time. (See below.)
-q		Set quiet mode so that <i>dtrace</i> only outputs explicitly traced data. When this option is set, <i>dtrace</i> suppresses messages such as the number of probes matched by the specified options and D programs. It also does not print column headers, the CPU ID, the probe ID, nor does it insert newlines into the output. Only data traced and formatted by D program statements, such as <i>trace()</i> and <i>printf()</i> , is displayed to <i>stdout</i> .
-s		Compile the specified D program source file. If the -e option is present, the program is compiled but instrumentation is not enabled. If the -l option is present, the program is compiled and the set of probes matched by it is listed, but instrumentation is not enabled. If neither -e nor -l option is present, the instrumentation specified by the D program is enabled and tracing begins.
-S		Print D compiler intermediate code. The D compiler produces a report of the intermediate code generated for each D program and outputs the report to <i>stderr</i> .

Option	Argument	Description
-v		Set verbose mode. When verbose mode is specified, <code>dtrace</code> reports stability attributes and arguments.
-V		Report the highest DTrace D language API version supported by <code>dtrace</code> . The version information is printed to <code>stdout</code> and the <code>dtrace</code> command exits.
-w		Permit destructive actions in D programs specified using the <code>-s</code> , <code>-P</code> , <code>-m</code> , <code>-f</code> , <code>-n</code> , or <code>-i</code> options. If the <code>-w</code> option is not specified, <code>dtrace</code> does not permit the compilation or enabling of a D program that contains destructive actions.
-x	<code>arg[=val]</code>	Enable or modify D compiler options and DTrace runtime tracing options. Boolean options are enabled by specifying their name. Options requiring other values are set by separating the option name and value with an equals sign (=). (See list of options and values below.)
-Z		Permit probe descriptions that match zero probes. If this option is not specified, <code>dtrace</code> reports an error and exits if any probe descriptions specified in D program files (<code>-s</code> option) or on the command line (<code>-P</code> , <code>-m</code> , <code>-f</code> , <code>-n</code> , or <code>-i</code> options) contain descriptions that do not match any known probes.

The arguments accepted by the `-P`, `-m`, `-f`, `-n`, and `-i` options can include an optional D language predicate enclosed in slashes `//` and optional D language action statement list enclosed in braces `{}`.

D program code specified on the command line must be appropriately quoted to avoid interpretation of meta-characters by the shell.

You can specify zero or more additional arguments on the `dtrace` command to define a set of macro variables (`$1`, `$2`, and so forth). The additional arguments can be used in D programs specified using the `-s` option or on the command line.

All argument parameters can also be set in a D script via `#pragma`, for example:

```
#pragma D option bufsize=512k
```

The table below shows the D compiler and runtime tracing options set via the `-x` option:

Option Name	Value	Description
<code>aggrate</code>	<i>time</i>	Rate of aggregation reading.
<code>aggsz</code>	<i>size</i>	Aggregation buffer size. The default size for this buffer is reduced to 128 KB on PlayStation®Vita from 4 MB on Solaris.
<code>bufpolicy</code>	<i>fill</i>	Record to a single, large in-kernel buffer rather than periodically swapping multiple buffers at the <code>switchrate</code> . Allow tracing to continue until one or more of the per-CPU buffers has filled.
<code>bufresz</code>	<i>auto or manual</i>	Buffer resizing policy.
<code>bufsz</code>	<i>size</i>	Principal buffer size. (See <code>-b</code> option above.) The default size for this buffer is reduced to 128 KB on PlayStation®Vita from 4 MB on Solaris.
<code>cleanrate</code>	<i>time</i>	Cleaning rate. Must be specified in number-per-second with the <code>hz</code> suffix.
<code>defaultargs</code>		Allow references to unspecified macro arguments.
<code>destructive</code>		Allow destructive actions. (See <code>-w</code> option above.)
<code>dynvarsz</code>	<i>size</i>	Dynamic variable space size. The default size for this buffer is reduced to 128 KB on PlayStation®Vita from 1 MB on Solaris.

Option Name	Value	Description
flowindent		Indent function entry and prefix with ->; unindent function return and prefix with <-. (See -F option above.)
nspec	<i>scalar</i>	Number of speculations
quiet		Output only explicitly traced data. (See -q option above.)
specsize	<i>size</i>	Speculation buffer size.
strsize	<i>size</i>	String size.
stackframes	<i>scalar</i>	Number of stack frames.
stackindent	<i>scalar</i>	Number of whitespace characters to use when indenting <code>stack()</code> or <code>ustack()</code> output.
statusrate	<i>time</i>	Rate of status checking.
switchrate	<i>time</i>	Rate of buffer switching.
ustackframes	<i>scalar</i>	Number of user stack frames.

Return Values

The following exit values are returned:

Value	Description
0	Successful completion. For D program requests, an exit status of 0 indicates that programs were successfully compiled, probes were successfully enabled, or anonymous state was successfully retrieved. <code>dtrace</code> returns 0 even if the specified tracing requests encountered errors or drops.
1	An error occurred. For D program requests, an exit status of 1 indicates that program compilation failed or that the specified request could not be satisfied.
2	Invalid command line options or arguments were specified.

Description

The `dtrace` command is the generic command-line interface front end to the DTrace facility.

The command implements a simple interface to invoke the D language compiler, including the ability to retrieve buffered trace data from the DTrace kernel facility along with a set of basic routines to format and print traced data.

The `dtrace` command provides an interface to the following DTrace facility services:

- Options that list the set of probes and providers currently published by DTrace
- Options that enable probes to directly use any of the probe description specifiers (provider, module, function, name)
- Options that run the D compiler to compile one or more D program files or programs written directly on the command line
- Options that generate anonymous tracing programs
- Options that generate program stability reports
- Options that modify DTrace tracing and buffering behavior and enable additional D compiler features

Using `dtrace` with the `-e` option, you can compile D programs and determine their properties without actually enabling tracing.

SCE DTrace Script Actions and Variables

SCE DTrace Script Actions

The SCE DTrace platform functionality closely follows the Sun standard DTrace functionality. There are some differences between them. The following table lists the DTrace script actions, their prototypes and arguments, and a description of each action. The status column in the table indicates one of the following for each action:

- **Identical:** SCE DTrace action should behave like any other DTrace implementation of this action.
- **Untested:** The SCE DTrace action is untested or partly working.
- **Not Supported:** The SCE DTrace action is currently not supported. It may or may not be implemented in a future SDK release.
- **Not Applicable:** The action is not applicable on the PlayStation®Vita platform.
- **Unique:** The action is unique to the PlayStation®Vita platform.

Action	Status	Prototype/Arguments	Description
alloca	Identical	<code>void *alloca(size_t size)</code>	Allocates size bytes out of scratch space and returns a pointer to the allocated memory.
avg	Identical	scalar expression	The arithmetic average of the specified expressions.
basename	Identical	<code>string basename(char *str)</code>	Creates a string that consists of a copy of the specified string, but without any prefix that ends in /.
bcopy	Not Supported	<code>void bcopy(void *src, void *dest, size_t size)</code>	Copies size bytes from the memory pointed to by src, to the memory pointed to by dest. All source memory must lie outside of scratch memory, and all destination memory must lie within it.
breakpoint	Unique	<code>void breakpoint(void)</code>	Generates a kernel breakpoint for a normal DTrace. On the other hand, for SCE DTrace, a software break unique to DTrace is generated for a user process. If a debugger (for example,) is attached, an exception is notified from the target to the host. This prompts the debugger to stop at the point of the <code>breakpoint()</code> action occurrence. Two probes are supported: <code>return</code> probe of the <code>syscall</code> provider and the <code>usdt</code> provider. Also, note this is a destructive action. Either use the aforementioned "-w" option or use with <code>destructive</code> set by <code>#pragma</code> . For a usage example, refer to the "Coordination between DTrace and the Host Tool" chapter of the "DTrace Overview" document.

SCE CONFIDENTIAL

Action	Status	Prototype/Arguments	Description
chill	Not Supported	<code>void chill(int nanoseconds)</code>	Causes DTrace to spin for the given nanoseconds. For system safety, DTrace will refuse to execute the <code>chill</code> action for more than 500 milliseconds in each 1-second interval on any CPU.
cleanpath	Identical	<code>string cleanpath(char *str)</code>	Creates a string that consists of a copy of the path indicated by <code>str</code> , but with redundant elements eliminated. This might result in shorter invalid paths being returned.
clear	Identical	aggregation	Clears only the aggregation's values; the aggregation's keys are retained.
commit	Untested	speculative buffer ID	Commits the speculative buffer associated with ID.
copyin	Identical	<code>void *copyin(uintptr_t addr, size_t size)</code>	Copies the specified size in bytes from the specified user address into a DTrace scratch buffer and returns the address of this buffer. The resulting buffer pointer is 8-byte aligned.
copyinstr	Identical	<code>string copyinstr(uintptr_t addr)</code>	Copies a null-terminated C string from the specified user address into a DTrace scratch buffer, and returns the address of this buffer. The <code>strsize</code> option limits the string length.
copyinto	Identical	<code>void copyinto(uintptr_t addr, size_t size, void *dest)</code>	Copies the specified size in bytes from the specified user address into the DTrace scratch buffer specified by <code>dest</code> .
copyout	Not Supported	<code>void copyout(void *buf, uintptr_t addr, size_t nbytes)</code>	Copies <code>nbytes</code> from the buffer <code>buf</code> to the address <code>addr</code> in the address space of the process associated with the current thread.
copyoutstr	Not Supported	<code>void copyoutstr(string str, uintptr_t addr, size_t maxlen)</code>	Copies the string <code>str</code> to the address <code>addr</code> in the address space of the process associated with the current thread. The string length is limited to the value set by the <code>strsize</code> option.
count	Identical	none	The number of times called.
denormalize	Identical	aggregation	Undoes previous <code>normalize()</code> operations on an aggregation - restores the original <code>raw</code> data.
dirname	Identical	<code>string dirname(char *str)</code>	Creates a string that consists of all but the last level of the path name specified by <code>str</code> .
discard	Untested	speculative buffer ID	Discards the speculative buffer associated with ID.

SCE CONFIDENTIAL

Action	Status	Prototype/Arguments	Description
exit	Identical	void exit(int status)	Immediately stops tracing, notifies DTrace consumer to cease tracing, performs any final processing, and calls exit with the specified status.
jstack	Not Applicable	void jstack (int nframes, int strsize) void jstack (int nframes) void jstack(void)	Alias for uestack() that uses the jstackframes option for the stack frame value and jstackstrsize for the string space size.
lquantize	Identical	scalar expression, lower bound, upper bound, step value	A linear frequency distribution, sized by the specified range, of the values of the specified expressions. Increments the value in the highest bucket that is less than the specified expression.
max	Identical	scalar expression	The largest value among the specified expressions.
min	Identical	scalar expression	The smallest value among the specified expressions.
msgdsize	Not Applicable	size_t msgdsize(mblk_t *mp)	Returns the number of bytes in the data message pointed to by mp.
msgsize	Not Applicable	size_t msgsize(mblk_t *mp)	Returns the number of bytes in the message pointed to by mp.
mutex_owed	Not Supported	int mutex_owed(kmutex_t *mutex)	Returns non-zero if the calling thread currently holds the specified kernel mutex, or zero if the specified adaptive mutex is currently unowned.
mutex_owner	Not Supported	kthread_t *mutex_owner(kmutex_t *mutex)	Returns the thread pointer of the current owner of the specified adaptive kernel mutex. Returns NULL if the specified adaptive mutex is currently unowned, or if the specified mutex is a spin mutex.
mutex_type_adaptive	Not Supported	int mutex_type_adaptive(kmutex_t *mutex)	Returns non-zero if the specified kernel mutex is of type MUTEX_ADAPTIVE, or zero if it is not.
mutex_type_spin	Not Supported	int mutex_type_spin(kmutex_t *mutex)	Returns non-zero if the specified kernel mutex is of type MUTEX_SPIN, or zero if it is not.
normalize	Identical	aggregation and a normalization factor	Normalize aggregation data with respect to some constant factor. The output of the aggregation shows each value divided by the normalization factor.

Document serial number: 000004892117

SCEI CONFIDENTIAL

Action	Status	Prototype/Arguments	Description
offsetof	Not Supported	size_t offsetof(type-name, member-name)	Return the byte offset of the specified member of the specified struct or union type. The value of the function is computed at compile time by the D compiler. The offsetof function may not be applied to bit-field members.
panic	Not Supported	void panic(void)	Causes a kernel panic. Should be used to force a system crash dump at a time of interest.
printa	Identical	void printa(aggregation) void printa(string format, aggregation)	Enables displaying and formatting of aggregations. If a format is not specified, the default format is used.
printf	Identical	void printf(string format,...)	The arguments are a format string followed by a variable number of arguments. The arguments are formatted for output according to the specified format string.
proc_game	Unique	int proc_game(pid_t pid)	Returns non-zero if the specified process ID is a game process.
proc_kernel	Unique	int proc_kernel(pid_t pid)	Returns non-zero if the specified process ID is a kernel process.
proc_system	Unique	int proc_system(pid_t pid)	Returns non-zero if the specified process ID is a system process.
progenyof	Identical	int progenyof(pid_t pid)	Returns non-zero if the calling process is among the progeny of the specified process ID.
quantize	Identical	scalar expression	A power-of-two frequency distribution of the values of the specified expressions. Increments the value in the highest power-of-two bucket that is less than the specified expression.
raise	Not Applicable	void raise(int signal)	Sends the specified signal to the currently running process.
rand	Identical	int rand(void)	Returns a pseudo-random integer.
rw_iswriter	Not Supported	int rw_iswriter(krwlock_t *rwlock)	Returns non-zero if the specified reader/writer lock is either held or desired by a writer. Returns zero if the lock is held only by readers, no writer is blocked, or the lock is not held at all.
rw_read_held	Not Supported	int rw_read_held(krwlock_t *rwlock)	Return non-zero if the specified reader/writer lock is currently held by one or more readers, or zero otherwise.
rw_write_held	Not Supported	int rw_write_held(krwlock_t *rwlock)	Returns non-zero if the specified reader/writer lock is currently held by a writer. Returns zero if the lock is held only by readers or not held at all.
setopt	Identical	void setopt(const char *, [const char *])	Set D runtime option (tuneable).

SCE CONFIDENTIAL

Action	Status	Prototype/Arguments	Description
speculate	Untested	speculative buffer ID	Denotes that the remainder of the clause should be traced to the speculative buffer specified by ID
speculation	Untested	int speculation (void)	Reserves a speculative trace buffer for use with speculate() and returns an identifier for this buffer.
stack	Not Supported	void stack(int nframes) void stack (void)	Records a kernel stack trace, nframes in depth. The number specified by the stackframes option is used if nframes is not specified. May also be used as a key to an aggregation.
stddev	Identical	scalar expression	The standard deviation of the specified expressions.
stop	Not Supported	void stop (void)	Forces the process that fires the enabled probe to stop when it next leaves the kernel.
strlen	Identical	size_t strlen(string str)	Returns the length of the specified string in bytes, excluding the terminating null byte.
strjoin	Identical	string strjoin(char *str1 char *str2)	Creates a string that consists of str1 concatenated with str2.
sum	Identical	scalar expression	The total value of the specified expressions.
system	Untested	void system(string program, ...)	Causes program to be executed as if it were given to the shell as input. Program may contain any of the printf/printa formats. Other arguments must match the specified format in program.
trace	Identical	void trace (expression)	Takes a D expression as argument and traces the result to the directed buffer.
tracemem	Not Supported	void tracemem (address, size_t nbytes)	Takes the memory address specified by address into the directed buffer for the length specified by nbytes. Address is a D expression.
trunc	Identical	aggregation and an optional truncation value	Without the truncation value, trunc discards both aggregation values and aggregation keys for the entire aggregation. When a truncation value n is present, trunc discards aggregation values and keys except for those values and keys associated with the highest n values.
uaddr	Not Supported	string uaddr(uintptr_t uaddr)	Prints the symbol for a specified user address, including hexadecimal offset.

SCE CONFIDENTIAL

Action	Status	Prototype/Arguments	Description
ustack	Identical	<code>void ustack(int nframes)</code> <code>void ustack(void)</code>	Records a user stack trace, <code>nframes</code> in depth. The number specified by the <code>ustackframes</code> option is used if <code>nframes</code> is unspecified.
usym	Not Supported	<code>string usym(uintptr_t uaddr)</code>	Returns the symbol for a specified address. This is analogous to how <code>uaddr</code> works, but without the hexadecimal offsets.

SCE DTrace Script Variables

The SCE DTrace platform types closely follow the Sun standard DTrace types, although there are some differences between the PlayStation®Vita and Sun script variables. The following table lists the DTrace variables and a description of each variable. The status column in the table indicates one of the following for each variable:

- **Identical:** SCE DTrace type should behave like any other DTrace implementation of this variable.
- **Unique:** Unique functionality for the PlayStation®Vita platform.
- **Not Supported:** The SCE DTrace variable is currently not supported. It may or may not be implemented in a future SDK release.
- **Not Applicable:** The variable is not applicable on the PlayStation®Vita platform.

Variable	Status	Description
uint32_t affinity	Unique	The CPU affinity setting for the current thread.
int32_t arg0, ..., arg9	Unique	The first 10 input arguments to a probe represented as raw 32-bit integers. 64-bit arguments are split across two 32-bit variables. If fewer than 10 arguments are passed to the current probe, the remaining variables return zero. For the syscall provider, only arg0 - arg3 are valid.
args[]	Not Supported	The typed arguments to the current probe, if any. args[] array is accessed using an integer index, but each element is defined to be the type corresponding to the given probe argument.
uintptr_t caller	Not Supported	The kernel program counter location of the current thread just before entering the current probe.
chipid_t chip	Identical	The CPU chip identifier for the current physical chip.
processorid_t cpu	Identical	The CPU identifier for the current CPU.
cpuinfo_t *curcpu	Not Supported	The CPU information for the current CPU.
lwpsinfo_t+ *curlwpsinfo	Not Supported	The lightweight process (LWP) state of the LWP associated with the current thread.
psinfo_t *curpsinfo	Not Supported	The process state of the process associated with the current thread.
kthread_t *curthread	Not Supported	The address of the operating system kernel's internal data structure for the current thread, the kthread_t. kthread_t is defined in <sys/thread.h>.
string cwd	Not Supported	The name of the current working directory of the process associated with the current thread.
uint_t epid	Identical	The enabled probe ID (EPID) for the current probe. This integer uniquely identifies a particular probe that is enabled with a specific predicate and set of actions.
int errno	Not Applicable	The error value returned by the last system call executed by this thread.
string execname	Identical	The name that was passed to exec(2) to execute the current process.
gid_t gid	Not Applicable	The probe ID for the current probe. This ID is the system-wide unique identifier for the probe as published by DTrace.
int hparamvdd	Unique	VDD power consumption (units: milliwatts) Includes IFTU, DMAC, internal bus, and SPM 32 KB/128 KB.
int hparamvdda	Unique	VDDA power consumption (units: milliwatts) Includes ARM cores and L2 cache.
int hparamvddc	Unique	VDDC power consumption (units: milliwatts) Includes CodecEngine and AVC decoder.
int hparamvddg	Unique	VDDG power consumption (units: milliwatts)

SCE CONFIDENTIAL

Variable	Status	Description
		Includes GPU cores.
int hparamvin	Unique	VIN power consumption (units: milliwatts) Includes everything except CP.
int hparamtemp	Unique	Custom SoC external substrate temperature (units: degrees Celsius)
uint_t id	Identical	The probe ID for the current probe. This ID is the system-wide unique identifier for the probe as published by DTrace.
uint_t ipl	Not Supported	The interrupt priority level (IPL) on the current CPU at probe firing time.
lgrp_id_t lgrp	Not Applicable	The latency group ID for the latency group of which the current CPU is a member.
long NULL	Identical	The value 0 as an integer constant.
pid_t pid	Identical	The process ID of the current process.
pid_t ppid	Identical	The parent process ID of the current process.
int32_t priority	Unique	The priority setting for the current thread.
string probefunc	Identical	The function name portion of the current probe's description.
string probemod	Identical	The module name portion of the current probe's description.
string probename	Identical	The name portion of the current probe's description.
string probeprovider	Identical	The provider name portion of the current probe's description.
string procname	Unique	The name of the current process.
psetid_t pset	Not Applicable	The processor set ID for the processor set containing the current CPU.
string root	Not Supported	The name of the root directory of the process associated with the current thread.
self	Identical	Reference thread-local variable.
uint_t stackdepth	Not Supported	The current thread's stack frame depth at probe firing time.
this	Identical	Reference probe-local variable.
string threadname	Unique	The name of the current thread.
id_t tid	Identical	The thread ID of the current thread. For threads associated with user processes, this value is equal to the result of a call to <code>topthread_self</code> .
uint64_t timestamp	Identical	The current value of a nanosecond timestamp counter. This counter increments from an arbitrary point in the past and should only be used for relative computations.
uint64_t ucaller	Identical	The user program counter location of the current thread just before entering the current probe.
uid_t uid	Not Applicable	The real user ID of the current process.
uint64_t uregs[]	Not Supported	The current thread's saved user-mode register values at probe firing time.
uint_t ustackdepth	Identical	User stack frame depth at probe firing time.
uint64_t vtimestamp	Identical	The current value of a nanosecond timestamp counter that is the amount of time the current thread has been running on a CPU, minus the time spent in DTrace predicates and actions.
uint64_t walltimestamp	Identical	The current number of nanoseconds since 00:00 Universal Coordinated Time, January 1, 1970.
string zonename	Not Applicable	The zone name associated with the current process.

DTrace Provider Probes

000004892117

Summary of Probes and Associated Providers

Provider	Probe	Description
dtrace	<i>BEGIN</i>	Fires before any other probe, and can be used to initialize state needed by other probes. See Oracle web page .
	<i>END</i>	Fires after all other probe clauses in a script have completed. See Oracle web page .
	<i>ERROR</i>	Fires when run-time errors occur. See Oracle web page .
interrupt	<i>vblank</i>	Fires at the start of a vertical blanking interval. See the “entry” interrupt provider probe.
proc	<i>create</i>	Fires when a process is created. See Oracle web page .
	<i>exec</i>	Fires when a process loads a new process image. See Oracle web page .
	<i>exec-failure</i>	Fires when loading a process fails. See Oracle web page .
	<i>exec-success</i>	Fires when loading a process succeeds. See Oracle web page .
	<i>exit</i>	Fires when a process exits. See Oracle web page .
	<i>lwp-create</i>	Fires when a thread is created. Differs from the standard DTrace implementation in the following aspect: The first argument is not a pointer to a process or thread structure. Instead, the first two arguments are the thread ID and the process ID. See Oracle web page .
	<i>lwp-start</i>	Fires immediately after a process begins execution. Differs from the standard DTrace implementation in the following aspect: The first argument is not a pointer to a process or thread structure. Instead, the first two arguments are the thread ID and the process ID. See Oracle web page .
	<i>lwp-exit</i>	Fires when a thread exits. Differs from the standard DTrace implementation in the following aspect: The first argument is not a pointer to a process or thread structure. Instead, the first two arguments are the thread ID and the process ID. See Oracle web page .
profile	<i>start</i>	Fires within the context of a newly created process. Differs from the standard DTrace implementation in the following aspect: The first argument is not a pointer to a process or thread structure. Instead, the first two arguments are the thread ID and the process ID. See Oracle web page .
	<i>profile-n</i>	Probes that fire at fixed intervals on all CPUs. See Oracle web page .
	<i>tick-n</i>	Probes that fire at fixed intervals on one CPU. See Oracle web page .

SCE CONFIDENTIAL

Provider	Probe	Description
sched	<i>change-pri</i>	Fires when the priority of a thread is about to change. Differs from the standard DTrace implementation in the following aspect: The first argument is not a pointer to a process or thread structure. Instead, the first two arguments are the thread ID and the process ID. See Oracle web page .
	<i>dequeue</i>	Fires before a runnable thread is dequeued. Differs from the standard DTrace implementation in the following aspect: The first argument is not a pointer to a process or thread structure. Instead, the first two arguments are the thread ID and the process ID. Also, the third argument is the CPU index for the processor runqueue to be added to or removed from. See Oracle web page .
	<i>enqueue</i>	Fires before a runnable thread is enqueued. Differs from the standard DTrace implementation in the following aspect: The first argument is not a pointer to a process or thread structure. Instead, the first two arguments are the thread ID and the process ID. Also, the third argument is the CPU index for the processor runqueue to be added to or removed from. See Oracle web page .
	<i>off-cpu</i>	Fires when a CPU is about to end thread execution. Differs from the standard DTrace implementation in the following aspect: The first argument is not a pointer to a process or thread structure. Instead, the first two arguments are the thread ID and the process ID. See Oracle web page .
	<i>on-cpu</i>	Fires when a CPU has begun executing a thread. See Oracle web page .
	<i>preempt</i>	Fires just before a thread is preempted. See Oracle web page .
	<i>remain-cpu</i>	Fires when a scheduling decision is made but thread continues running. See Oracle web page .
	<i>sleep</i>	Fires before a thread sleeps on a synchronization object. See Oracle web page .
	<i>wait</i>	Fires just before a thread that has blocked is taken off the CPU. See the "wait" probe.
	<i>wakeup</i>	Fires before the current thread wakes a sleeping thread. Differs from the standard DTrace implementation in the following aspect: The first argument is not a pointer to a process or thread structure. Instead, the first two arguments are the thread ID and the process ID. See Oracle web page .
syscall	<i>entry</i>	Fires before a system call is entered. See Oracle web page .
	<i>return</i>	Fires after a system call completes. See Oracle web page .

(The above reference destination has been confirmed as of November 20, 2014. Note that pages may have been subsequently moved or its contents modified.)

interrupt Provider Probes

SCE CONFIDENTIAL

entry

Fires when the interrupt handler is entered.

Definition

```
entry  
interrupt:::entry
```

Arguments

None.

Return Values

None.

Description

The `entry` probe fires when the interrupt handler is entered. The function name is the name of the instrumented interrupt handler. The module name is undefined.

SCE CONFIDENTIAL

return

Fires when the interrupt handler has completed.

Definition

```
return  
interrupt:::return
```

Arguments

None.

Return Values

None.

Description

The `return` probe fires when the interrupt handler has completed and before control transfers back to the previous context. The function name is the name of the instrumented interrupt handler. The module name is undefined.

SCE CONFIDENTIAL

vblank

Fires at the start of a vertical blanking interval.

Definition

```
vblank
interrupt::vblank
```

Arguments

None.

Return Values

Returns the internal probe identifier and CPU identifier.

Description

The `vblank` probe fires at the start of the vertical blanking interval at the end of a frame.

Examples

The example below uses the `vblank` probe as part of a larger script:

```
syscallTimePerFrame.d:
int gFrameCount; /* globals initialized with 0 */
inline string gGameProcess = "GameOfTheYear";

syscall:::entry
/ execname == gGameProcess /
{
    self->start = timestamp;
}

syscall:::return
/ execname == gGameProcess&& self->start != 0 /
{
    this->elapsed = timestamp - self->start;
    @syscallTime[probefunc] = sum(this->elapsed);
    self->start = 0;
}

interrupt::vblank
{
    gFrameCount++;
}

END
{
    printf("%s - system call time (in nanoseconds) per frame (%d frames)\n",
           gGameProcess, gFrameCount);
    normalize(@syscallTime, gFrameCount); /* divide by number of frames */
    /* printa(@syscallTime); */ /* implicit */
}
```

SCE CONFIDENTIAL

}

dtrace.exe -s syscallTimePerFrame.d

000004892117

Document serial number: 000004892117

sched Provider Probes

000004892117

SCE CONFIDENTIAL

wait

Fires just before a thread that has blocked is taken off the CPU.

Definition

```
wait  
sched:::wait
```

Arguments

None.

Return Values

None.

Description

The `wait` probe fires when a thread that has blocked is about to be taken off the CPU. The probe fires just before the thread is taken off the CPU.

User-Level Statically Defined Tracing Provider

usdt

User-Level Statically Defined Tracing (usdt) provider

Description

The usdt provider lets you define your own static probes in application code. These static probes add very little runtime overhead. These probes are dynamically enabled by an instruction patching mechanism. Probes you define can trace user space state including strings.

The sections below explain the complete process to add a probe to an application, covering defining a provider and probes, generating the header for the probes, compiling and linking the application with the probes so that the probes are added to the application, and finally running the application.

Note that SCE DTrace does not yet support is-enabled probes.

Define Provider and Probes

To add probes to your application, first define a provider and its probes in a .d file. For example, the code below defines the provider primes:

```
provider primes {
/* Start of the prime calculation */
probe primecalc__start(long prime);
/* End of the prime calculation */
probe primecalc__done(long prime, int isprime);
/* Exposes the size of the table of existing primes */
probe primecalc__tablesize(long tablesize);
};
```

The .d file is converted by dtrace.exe into a header file that is included by source files that use the probes. The .d file is also used to generate a special ELF section (.SUNW_dof), which is added to your application binary in order to describe the provider and probes.

Generate the Probes

Header

After defining your provider and probes in a d file, the next step is to generate a header using dtrace.exe, as follows:

```
%SCE_PSP2_SDK_DIR%\host_tools\bin\dtrace.exe -h -s probes.d -o probes.h
```

You can automate the above step by adding a Pre-Build Event to your Visual Studio project. Add this event under Build Events, as shown below:

```
%SCE_PSP2_SDK_DIR%\host_tools\bin\dtrace.exe -h -s $(ProjectDir)\probes.d
-o $(ProjectDir)\probes.h
```

The above step generates the header file shown below for this example:

```
/*
 * Generated by dtrace(1M).
 */

#ifndef _PROBES_H
#define _PROBES_H

#ifdef __cplusplus
extern "C" {
#endif

#ifndef NDTRACE
```

SCE CONFIDENTIAL

```

#define PRIMES_PRIMECALC_DONE(arg0, arg1) \
__dtrace_primes__primecalc__done(arg0, arg1)
#define PRIMES_PRIMECALC_START(arg0) \
__dtrace_primes__primecalc__start(arg0)
#define PRIMES_PRIMECALC_TABLESIZE(arg0) \
__dtrace_primes__primecalc__tablesize(arg0)

extern void __dtrace_primes__primecalc__done(long, int);
extern void __dtrace_primes__primecalc__start(long);
extern void __dtrace_primes__primecalc__tablesize(long);

#else

#define PRIMES_PRIMECALC_DONE(arg0, arg1)
#define PRIMES_PRIMECALC_START(arg0)
#define PRIMES_PRIMECALC_TABLESIZE(arg0)

#endif

#ifdef __cplusplus
}
#endif

#endif/* _PROBES_H */

```

Add Probes to Application

Include the generated header file wherever you want to instrument your code with one of the probes you defined. For the example above, the following code might be used:

```

#include "probes.h"

void calcprimes(void)
{
    long primes[1000000] = { 3 };
    long primecount = 1;

    long divisor = 0;
    long currentprime = 5;
    long isprime = 1;

    while (currentprime < 1000000) {
        isprime = 1;
        PRIMES_PRIMECALC_START(currentprime);
        for (divisor = 0; divisor < primecount; divisor++) {
            if (currentprime % primes[divisor] == 0) {
                isprime = 0;
            }
        }
        PRIMES_PRIMECALC_DONE(currentprime, isprime);
        if (isprime) {
            primes[primecount++] = currentprime;
            PRIMES_PRIMECALC_TABLESIZE(primecount);
        }
        currentprime = currentprime + 2;
    }
}

int main(int argc, char **argv)
{
    calcprimes();
    return 0;
}

```

©SCEI

SCE CONFIDENTIAL

}

Note that you can disable calls to USDT probes by defining the NDTRACE macro prior to including the header.

Compile the Application

You can now compile the application. Since dtrace.exe must process any object files containing probes before your application is linked, you do the following:

```
del /F probes.o
%SCE_PSP2_SDK_DIR%\host_tools\bin\dtrace.exe -G -s probes.d -o probes.o *.o
```

The above code patches each probe call point to be disabled by default, and also generates a new object file (probes.o) that contains a special .SUNW_dof section describing your provider, probes, and any probe argument types. It is not necessary to process all object files; only process those files which use USDT probes.

To automate the above step you can add a Pre-Link Event to your Visual Studio project under Build Events, as follows:

```
if exist $(TargetDir)\probes.o del /F $(TargetDir)\probes.o
%SCE_PSP2_SDK_DIR%\host_tools\bin\dtrace.exe -G -s $(ProjectDir)\probes.d
-o $(TargetDir)\probes.o $(TargetDir)\*.o
```

Link the Application

You must link your application with the SceDTrace_stub library as well as the probes object. To do so, in your Visual Studio project, add the following under **Linker -> Additional Dependencies**:

```
-lSceDTrace_stub
```

Then, under **Linker -> Command Line**, add your probes object, as follows:

```
$(TargetDir)\probes.o
```

Run the Application

The provider name you chose will be appended with the process identifier of your application. For our example, if the binary is launched with process id XXXXX, the provider name will be primesXXXXX:

```
>psp2run /r game.self
>psp2ctrl plist
PID          PPID          Name          Path
0x00050d57 0xffffffff NPXS10999 host0:Z:\game\PSP2_Debug\game.self
>
```

You can list the available probes for your provider:

```
>%SCE_PSP2_SDK_DIR%\host_tools\bin\dtrace.exe -P primes* -l
ID PROVIDER          MODULE          FUNCTION NAME
2703 primes331095     game.self       calcprimes primecalc-done
2704 primes331095     game.self       calcprimes primecalc-start
2705 primes331095     game.self       calcprimes primecalc-tablesize
>
```

At this point, your provider and its probes can be used in DTrace scripts in the same way as any other probe.

```
BEGIN
{
    self->begin_timestamp = -1;
}

primes*:::primecalc-start
/self->begin_timestamp == -1/
{
```

SCE CONFIDENTIAL

```

        self->begin_timestamp = timestamp;
    }

primes*:::primecalc-done
/arg1 != 0 && self->begin_timestamp != -1/
{
    /* arg0: value */
    /* arg1: 0:not prime !=0:prime */

    @time[probename] = quantize(timestamp - self->begin_timestamp);

    self->begin_timestamp = -1;
}

```

Here is the output from the above code:

```

>%SCE_PSP2_SDK_DIR%\host_tools\bin\dtrace.exe -s primes.d
dtrace.exe: script 'Z:\primes.d' matched 3 probes

```

```

primecalc-done
      value  ----- Distribution ----- count
      262144 |                                0
      524288 |@@                            254
     1048576 |@@@                          528
     2097152 |@@@@@@@@@@@               1124
     4194304 |@@@@@@@@@@@@@@@@@           1568
     8388608 |@@@@@@@@@@@@@               1028
    16777216 |@@                            266
    33554432 |                                12
     67108864 |                                0
    134217728 |                                1
    268435456 |                                0
>

```