



Red Hat Enterprise Linux 7 SystemTap Tapset Reference

For SystemTap in Red Hat Enterprise Linux 7

Red Hat Enterprise Linux Documentation William Cohen
Don Domingo

Red Hat Enterprise Linux 7 SystemTap Tapset Reference

For SystemTap in Red Hat Enterprise Linux 7

William Cohen
Red Hat Software Engineering
wcohen@redhat.com

Don Domingo
Red Hat Customer Content Services
ddomingo@redhat.com

Red Hat Enterprise Linux Documentation

Edited by

Jacquelynn East
Red Hat Customer Content Services
@redhat.com

Legal Notice

Copyright © 2010 Red Hat, Inc. and others.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, MetaMatrix, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

The Tapset Reference Guide describes the most common tapset definitions users can apply to SystemTap scripts.

Table of Contents

Chapter 1. Introduction	25
1.1. Documentation Goals	25
Chapter 2. Tapset Development Guidelines	26
2.1. Writing Good Tapsets	26
2.2. Elements of a Tapset	27
Chapter 3. Context Functions	30
Name	30
Synopsis	30
Arguments	30
General Syntax	30
Description	30
Name	30
Synopsis	30
Arguments	31
General Syntax	31
Description	31
Name	31
Synopsis	31
Arguments	31
General Syntax	31
Description	31
Name	32
Synopsis	32
Arguments	32
General Syntax	32
Description	32
Name	32
Synopsis	32
Arguments	32
General Syntax	33
Description	33
Name	33
Synopsis	33
Arguments	33
General Syntax	33
Description	33
Name	33
Synopsis	34
Arguments	34
General Syntax	34
Description	34
Name	34
Synopsis	34
Arguments	34
General Syntax	34
Description	35
Name	35
Synopsis	35
Arguments	35
General Syntax	35

Description	35
Name	35
Synopsis	35
Arguments	36
General Syntax	36
Description	36
Name	36
Synopsis	36
Arguments	36
General Syntax	36
Description	36
Name	37
Synopsis	37
Arguments	37
General Syntax	37
Description	37
Name	37
Synopsis	37
Arguments	37
General Syntax	38
Description	38
Name	38
Synopsis	38
Arguments	38
General Syntax	38
Description	38
Name	38
Synopsis	39
Arguments	39
General Syntax	39
Description	39
Name	39
Synopsis	39
Arguments	39
General Syntax	39
Description	40
Name	40
Synopsis	40
Arguments	40
General Syntax	40
Name	40
Synopsis	40
Arguments	40
General Syntax	41
Description	41
Name	41
Synopsis	41
Arguments	41
General Syntax	41
Description	41
Name	42
Synopsis	42
Arguments	42

-	
General Syntax	42
Description	42
Name	42
Synopsis	42
Arguments	42
General Syntax	43
Description	43
Name	43
Synopsis	43
Arguments	43
General Syntax	43
Description	43
Name	43
Synopsis	44
Arguments	44
General Syntax	44
Description	44
Name	44
Synopsis	44
Arguments	44
General Syntax	44
Description	45
Name	45
Synopsis	45
Arguments	45
General Syntax	45
Description	45
Name	45
Synopsis	45
Arguments	46
General Syntax	46
Description	46
Name	46
Synopsis	46
Arguments	46
General Syntax	47
Description	47
Name	47
Synopsis	47
Arguments	47
General Syntax	47
Description	47
Name	47
Synopsis	48
Arguments	48
General Syntax	48
Description	48
Name	48
Synopsis	48
Arguments	48
General Syntax	49
Description	49
Name	49

Synopsis	49
Arguments	49
Description	49
Name	49
Synopsis	50
Arguments	50
General Syntax	50
Description	50
Please note	50
Name	50
Synopsis	50
Arguments	51
General Syntax	51
Description	51
Name	51
Synopsis	51
Arguments	51
Description	51
Name	51
Synopsis	52
Arguments	52
General Syntax	52
Description	52
Name	52
Synopsis	52
Arguments	52
General Syntax	53
Description	53
Name	53
Synopsis	53
Arguments	53
Description	53
Name	53
Synopsis	54
Arguments	54
Description	54
Name	54
Synopsis	54
Arguments	54
Description	54
Name	55
Synopsis	55
Arguments	55
Description	55
Name	55
Synopsis	55
Arguments	56
General Syntax	56
Description	56
Name	56
Synopsis	56
Arguments	56
Description	56

-----	--
Name	57
Synopsis	57
Arguments	57
General Syntax	57
Description	57
Name	57
Synopsis	57
Arguments	57
General Syntax	58
Description	58
Name	58
Synopsis	58
Arguments	58
General Syntax	58
Description	58
Name	58
Synopsis	59
Arguments	59
General Syntax	59
Description	59
Name	59
Synopsis	59
Arguments	59
Description	59
Note	60
Name	60
Synopsis	60
Arguments	60
Description	60
Note	60
Name	60
Synopsis	61
Arguments	61
Description	61
Note	61
Name	61
Synopsis	61
Arguments	61
Description	62
Note	62
Name	62
Synopsis	62
Arguments	62
General Syntax	62
Description	62
Name	62
Synopsis	63
Arguments	63
General Syntax	63
Description	63
Name	63
Synopsis	63
Arguments	63

Arguments	63
General Syntax	64
Description	64
Name	64
Synopsis	64
Arguments	64
General Syntax	64
Description	64
Name	64
Synopsis	65
Arguments	65
General Syntax	65
Description	65
Name	65
Synopsis	65
Arguments	65
Description	66
Name	66
Synopsis	66
Arguments	66
Description	66
Name	66
Synopsis	66
Arguments	66
General Syntax	67
Description	67
Name	67
Synopsis	67
Arguments	67
General Syntax	67
Description	67
Name	67
Synopsis	68
Arguments	68
General Syntax	68
Description	68
Name	68
Synopsis	68
Arguments	68
General Syntax	69
Description	69
Name	69
Synopsis	69
Arguments	69
General Syntax	69
Description	69
Name	69
Synopsis	70
Arguments	70
General Syntax	70
Description	70
Name	70
Synopsis	70
Arguments	70

Arguments	70
General Syntax	70
Description	71
Name	71
Synopsis	71
Arguments	71
General Syntax	71
Description	71
Name	71
Synopsis	71
Arguments	72
General Syntax	72
Description	72
Name	72
Synopsis	72
Arguments	72
General Syntax	72
Description	73
Name	73
Synopsis	73
Arguments	73
General Syntax	73
Description	73
Chapter 4. Timestamp Functions	74
Name	74
Synopsis	74
Arguments	74
General Syntax	74
Description	74
Name	74
Synopsis	74
Arguments	75
General Syntax	75
Description	75
Name	75
Synopsis	75
Arguments	75
General Syntax	75
Description	75
Name	76
Synopsis	76
Arguments	76
General Syntax	76
Description	76
Name	76
Synopsis	76
Arguments	76
General Syntax	77
Description	77
Chapter 5. Time string utility function	78
Name	78
Synopsis	78

Arguments	78
General Syntax	78
Description	78
Chapter 6. Memory Tapset	80
Name	80
Synopsis	80
Arguments	80
Name	80
Synopsis	80
Values	80
Context	81
Name	81
Synopsis	81
Values	81
Name	81
Synopsis	81
Arguments	82
General Syntax	82
Description	82
Name	82
Synopsis	82
Values	82
Context	82
Description	83
Name	83
Synopsis	83
Values	83
Context	83
Description	83
Name	84
Synopsis	84
Values	84
Context	84
Name	84
Synopsis	84
Values	84
Context	85
Name	85
Synopsis	85
Values	85
Context	85
Name	85
Synopsis	86
Values	86
Context	86
Name	86
Synopsis	86
Values	86
Name	87
Synopsis	87
Values	87
Description	88
Name	88

name	88
Synopsis	88
Values	88
Name	89
Synopsis	89
Values	89
Description	90
Name	90
Synopsis	90
Values	90
Name	90
Synopsis	90
Values	91
Description	91
Name	91
Synopsis	91
Arguments	91
Description	91
Name	92
Synopsis	92
Arguments	92
Description	92
Name	92
Synopsis	92
Arguments	92
Description	92
Name	93
Synopsis	93
Arguments	93
Description	93
Name	93
Synopsis	93
Arguments	93
Description	94
Name	94
Synopsis	94
Arguments	94
Description	94
Name	94
Synopsis	94
Arguments	94
Description	95
Name	95
Synopsis	95
Arguments	95
Description	95
Name	95
Synopsis	95
Arguments	96
Description	96
Name	96
Synopsis	96
Arguments	96
Description	96

Description	96
Name	96
Synopsis	96
Arguments	97
Name	97
Synopsis	97
Arguments	97
Description	97
Name	97
Synopsis	97
Arguments	98
Description	98
Name	98
Synopsis	98
Arguments	98
Description	98
Name	98
Synopsis	99
Arguments	99
Description	99
Chapter 7. Task Time Tapset	100
Name	100
Synopsis	100
Arguments	100
Description	100
Name	100
Synopsis	100
Arguments	100
Description	101
Name	101
Synopsis	101
Arguments	101
Description	101
Name	101
Synopsis	101
Arguments	102
Description	102
Name	102
Synopsis	102
Arguments	102
Name	102
Synopsis	102
Arguments	102
Description	103
Name	103
Synopsis	103
Arguments	103
Description	103
Name	103
Synopsis	103
Arguments	104
Description	104
Name	104

.....	104
Synopsis	104
Arguments	104
Description	104
Chapter 8. IO Scheduler and block IO Tapset	105
Name	105
Synopsis	105
Values	105
Name	105
Synopsis	105
Values	105
Name	106
Synopsis	106
Values	106
Name	107
Synopsis	107
Values	107
Name	107
Synopsis	107
Values	108
Name	108
Synopsis	108
Values	108
Name	109
Synopsis	109
Values	109
Description	110
Name	110
Synopsis	110
Values	110
Description	110
Name	111
Synopsis	111
Values	111
Description	111
Name	111
Synopsis	112
Values	112
Name	112
Synopsis	112
Values	112
Description	113
Name	113
Synopsis	113
Values	113
Description	113
Name	113
Synopsis	113
Values	114
Description	114
Name	114
Synopsis	114
Values	114

Description	114
Context	115
Name	115
Synopsis	115
Values	115
Description	115
Context	115
Name	116
Synopsis	116
Values	116
Description	116
Context	116
Name	116
Synopsis	117
Values	117
Description	117
Context	117
Name	117
Synopsis	117
Values	118
Description	118
Context	118
Chapter 9. SCSI Tapset	119
Name	119
Synopsis	119
Values	119
Name	119
Synopsis	119
Values	120
Name	120
Synopsis	121
Values	121
Name	121
Synopsis	122
Values	122
Name	122
Synopsis	123
Values	123
Name	123
Synopsis	124
Values	124
Chapter 10. TTY Tapset	125
Name	125
Synopsis	125
Values	125
Name	125
Synopsis	126
Values	126
Name	126
Synopsis	126
Values	126
Name	127

name	127
Synopsis	127
Values	127
Name	128
Synopsis	128
Values	128
Name	128
Synopsis	128
Values	128
Name	129
Synopsis	129
Values	129
Name	129
Synopsis	129
Values	130
Name	130
Synopsis	130
Values	130
Name	131
Synopsis	131
Values	131
Name	131
Synopsis	131
Values	131
Chapter 11. Networking Tapset	133
Name	133
Synopsis	133
Values	133
Name	133
Synopsis	133
Values	133
Name	134
Synopsis	134
Values	134
Name	134
Synopsis	134
Values	135
Name	135
Synopsis	135
Values	135
Name	135
Synopsis	135
Values	135
Name	136
Synopsis	136
Values	136
Name	136
Synopsis	136
Values	136
Name	137
Synopsis	137
Values	137
Name	137

Synopsis	137
Values	138
Name	138
Synopsis	138
Values	138
Name	138
Synopsis	138
Values	138
Name	139
Synopsis	139
Values	139
Name	139
Synopsis	139
Values	139
Name	140
Synopsis	140
Values	140
Context	140
Name	140
Synopsis	140
Values	141
Context	141
Name	141
Synopsis	141
Values	141
Context	142
Name	142
Synopsis	142
Values	142
Context	143
Name	143
Synopsis	143
Values	143
Context	143
Name	144
Synopsis	144
Values	144
Context	144
Name	144
Synopsis	144
Values	144
Context	145
Name	145
Synopsis	145
Values	145
Context	145
Name	146
Synopsis	146
Values	146
Name	147
Synopsis	147
Values	147
Context	147

-----	---
Name	147
Synopsis	148
Values	148
Context	148
Name	148
Synopsis	148
Values	148
Context	149
Name	149
Synopsis	149
Values	149
Context	149
Name	149
Synopsis	149
Values	150
Context	150
Name	150
Synopsis	150
Values	150
Context	150
Name	151
Synopsis	151
Arguments	151
Chapter 12. Socket Tapset	152
Name	152
Synopsis	152
Values	152
Context	152
Name	153
Synopsis	153
Values	153
Context	153
Name	154
Synopsis	154
Values	154
Context	154
Description	154
Name	155
Synopsis	155
Values	155
Context	155
Description	156
Name	156
Synopsis	156
Values	156
Context	156
Description	157
Name	157
Synopsis	157
Values	157
Context	158
Description	158

Name	158
Synopsis	158
Values	158
Context	159
Description	159
Name	159
Synopsis	159
Values	159
Context	160
Description	160
Name	160
Synopsis	160
Values	160
Context	161
Description	161
Name	161
Synopsis	161
Values	161
Context	162
Description	162
Name	162
Synopsis	162
Values	162
Context	163
Description	163
Name	163
Synopsis	163
Values	163
Context	164
Description	164
Name	164
Synopsis	164
Values	164
Context	165
Description	165
Name	165
Synopsis	165
Values	165
Context	166
Description	166
Name	166
Synopsis	166
Values	167
Context	167
Description	167
Name	167
Synopsis	167
Values	167
Context	168
Description	168
Name	168
Synopsis	168
Values	169

Context	169
Description	169
Name	169
Synopsis	169
Values	169
Context	170
Description	170
Name	170
Synopsis	170
Arguments	170
Name	170
Synopsis	170
Arguments	171
Name	171
Synopsis	171
Arguments	171
Name	171
Synopsis	171
Arguments	171
Description	172
Name	172
Synopsis	172
Arguments	172
Name	172
Synopsis	172
Arguments	172
Chapter 13. Kernel Process Tapset	173
Name	173
Synopsis	173
Values	173
Context	173
Description	173
Name	173
Synopsis	173
Values	174
Context	174
Description	174
Name	174
Synopsis	174
Values	174
Context	174
Description	175
Name	175
Synopsis	175
Values	175
Context	175
Description	175
Name	175
Synopsis	175
Values	176
Context	176
Description	176

Name	176
Synopsis	176
Values	176
Context	176
Description	177
Chapter 14. Signal Tapset	178
Name	178
Synopsis	178
Values	178
Context	179
Name	179
Synopsis	179
Values	179
Context	179
Description	179
which means that	180
Name	180
Synopsis	180
Values	180
Name	181
Synopsis	181
Values	181
Name	181
Synopsis	181
Values	182
Name	182
Synopsis	182
Values	182
Name	183
Synopsis	183
Values	183
Name	183
Synopsis	183
Values	183
Name	184
Synopsis	184
Values	184
Name	184
Synopsis	184
Values	184
Name	185
Synopsis	185
Values	185
Name	185
Synopsis	185
Values	186
Description	186
Name	186
Synopsis	186
Values	186
Name	187
Synopsis	187
Values	187

values	187
Description	187
Name	187
Synopsis	188
Values	188
Name	188
Synopsis	188
Values	188
Name	189
Synopsis	189
Values	189
Name	189
Synopsis	189
Values	189
Description	190
Name	190
Synopsis	190
Values	190
Name	190
Synopsis	190
Values	191
Name	191
Synopsis	191
Values	191
Name	192
Synopsis	192
Values	192
Name	192
Synopsis	193
Values	193
Name	193
Synopsis	193
Values	193
Name	194
Synopsis	194
Values	194
Name	194
Synopsis	194
Values	194
Chapter 15. Directory-entry (dentry) Tapset	196
Name	196
Synopsis	196
Arguments	196
Description	196
Name	196
Synopsis	196
Arguments	196
Description	197
Name	197
Synopsis	197
Arguments	197
Description	197
Name	197

.....	---
Synopsis	197
Arguments	198
Description	198
Chapter 16. Logging Tapset	199
Name	199
Synopsis	199
Arguments	199
General Syntax	199
Description	199
Name	199
Synopsis	199
Arguments	200
General Syntax	200
Description	200
Name	200
Synopsis	200
Arguments	200
General Syntax	200
Description	201
Name	201
Synopsis	201
Arguments	201
Description	201
Name	201
Synopsis	201
Arguments	201
Description	202
Chapter 17. Random functions Tapset	203
Name	203
Synopsis	203
Arguments	203
Chapter 18. String and data retrieving functions Tapset	204
Name	204
Synopsis	204
Arguments	204
General Syntax	204
Description	204
Name	204
Synopsis	204
Arguments	205
General Syntax	205
Description	205
Name	205
Synopsis	205
Arguments	205
General Syntax	206
Description	206
Name	206
Synopsis	206
Arguments	206
General Syntax	206

General Syntax	205
Description	206
Name	206
Synopsis	207
Arguments	207
Description	207
Name	207
Synopsis	207
Arguments	207
General Syntax	207
Description	208
Name	208
Synopsis	208
Arguments	208
General Syntax	208
Description	208
Name	208
Synopsis	209
Arguments	209
General Syntax	209
Description	209
Name	209
Synopsis	209
Arguments	209
General Syntax	209
Description	210
Name	210
Synopsis	210
Arguments	210
General Syntax	210
Description	210
Name	210
Synopsis	211
Arguments	211
General Syntax	211
Description	211
Name	211
Synopsis	211
Arguments	211
General Syntax	212
Description	212
Name	212
Synopsis	212
Arguments	212
General Syntax	212
Description	212
Name	213
Synopsis	213
Arguments	213
General Syntax	213
Description	213
Name	213
Synopsis	214
Arguments	214

Arguments	214
General Syntax	214
Description	214
Name	214
Synopsis	214
Arguments	214
General Syntax	215
Description	215
Name	215
Synopsis	215
Arguments	215
General Syntax	215
Description	215
Name	216
Synopsis	216
Arguments	216
General Syntax	216
Description	216
Name	216
Synopsis	216
Arguments	217
General Syntax	217
Description	217
Name	217
Synopsis	217
Arguments	217
General Syntax	217
Description	217
Name	218
Synopsis	218
Arguments	218
General Syntax	218
Description	218
Name	218
Synopsis	218
Arguments	219
General Syntax	219
Description	219
Name	219
Synopsis	219
Arguments	219
General Syntax	219
Description	219
Name	220
Synopsis	220
Arguments	220
General Syntax	220
Description	220
Chapter 19. A collection of standard string functions	221
Name	221
Synopsis	221
Arguments	221
General Syntax	221

Description	221
Name	221
Synopsis	221
Arguments	222
General Syntax	222
Description	222
Name	222
Synopsis	222
Arguments	222
General Syntax	223
Description	223
Name	223
Synopsis	223
Arguments	223
General syntax	223
Description	223
Name	224
Synopsis	224
Arguments	224
General Syntax	224
Description	224
Name	224
Synopsis	224
Arguments	224
General Syntax	225
Description	225
Name	225
Synopsis	225
Arguments	225
General Syntax	226
Description	226
Name	226
Synopsis	226
Arguments	226
General Syntax	226
Description	227
Name	227
Synopsis	227
Arguments	227
General Syntax	227
Description	227
Name	227
Synopsis	228
Arguments	228
General Syntax	228
Description	228
Chapter 20. Utility functions for using ansi control chars in logs	229
Name	229
Synopsis	229
Arguments	229
General Syntax	229
Description	229

Name	229
Synopsis	229
Arguments	230
General Syntax	230
Description	230
Name	230
Synopsis	230
Arguments	230
General Syntax	230
Description	231
Name	231
Synopsis	231
Arguments	231
General Syntax	231
Description	231
Name	232
Synopsis	232
Arguments	232
General Syntax	232
Description	232
Name	232
Synopsis	232
Arguments	232
General Syntax	233
Description	233
Name	233
Synopsis	233
Arguments	233
General Syntax	233
Description	233
Name	234
Synopsis	234
Arguments	234
General Syntax	234
Description	234
Name	234
Synopsis	234
Arguments	234
General Syntax	235
Description	235
Name	235
Synopsis	235
Arguments	235
General Syntax	235
Description	235
Name	235
Synopsis	236
Arguments	236
General Syntax	236
Description	236
Appendix A. Revision History	237

Chapter 1. Introduction

SystemTap provides free software (GPL) infrastructure to simplify the gathering of information about the running Linux system. This assists diagnosis of a performance or functional problem. SystemTap eliminates the need for the developer to go through the tedious and disruptive instrument, recompile, install, and reboot sequence that may be otherwise required to collect data.

SystemTap provides a simple command line interface and scripting language for writing instrumentation for a live, running kernel. This instrumentation uses probe points and functions provided in the *tapset* library.

Tapsets are scripts that encapsulate knowledge about a kernel subsystem into pre-written probes and functions that can be used by other scripts. Tapsets are analogous to libraries for C programs. They hide the underlying details of a kernel area while exposing the key information needed to manage and monitor that aspect of the kernel. They are typically developed by kernel subject matter experts.

A tapset exposes the high-level data and state transitions of a subsystem. For the most part, good tapset developers assume that SystemTap users know little to nothing about the kernel subsystem's low-level details. As such, tapset developers write tapsets that help ordinary SystemTap users write meaningful and useful SystemTap scripts.

[Report a bug](#)

1.1. Documentation Goals

This guide aims to document SystemTap's most useful and common tapset entries; it also contains guidelines on proper tapset development and documentation. The tapset definitions contained in this guide are extracted automatically from properly-formatted comments in the code of each tapset file. As such, any revisions to the definitions in this guide should be applied directly to their respective tapset file.

[Report a bug](#)

Chapter 2. Tapset Development Guidelines

This chapter describes guidelines on proper tapset documentation. It also contains information on how to properly document tapsets, to ensure they are properly defined in this guide.

[Report a bug](#)

2.1. Writing Good Tapsets

The first step to writing good tapsets is to create a simple model of your subject area. For example, a model of the process subsystem might include the following:

Key Data

- ✧ process ID
- ✧ parent process ID
- ✧ process group ID

State Transitions

- ✧ forked
- ✧ exec'd
- ✧ running
- ✧ stopped
- ✧ terminated



Note

Both lists are examples, and are not meant to represent a complete list.

Use your subsystem expertise to find probe points (function entries and exits) that expose the elements of the model, then define probe aliases for those points. Be aware that some state transitions can occur in more than one place. In those cases, an alias can place a probe in multiple locations.

For example, process execs can occur in either the `do_execve()` or the `compat_do_execve()` functions. The following alias inserts probes at the beginning of those functions:

```
probe kprocess.exec = kernel.function("do_execve"),
kernel.function("compat_do_execve")
{probe body}
```

Try to place probes on stable interfaces (i.e., functions that are unlikely to change at the interface level) whenever possible. This will make the tapset less likely to break due to kernel changes. Where kernel version or architecture dependencies are unavoidable, use preprocessor conditionals (see the **stap(1)** man page for details).

Fill in the probe bodies with the key data available at the probe points. Function entry probes can access the entry parameters specified to the function, while exit probes can access the entry parameters and the return value. Convert the data into meaningful forms where appropriate (e.g., bytes to kilobytes, state values to strings, etc).

You may need to use auxiliary functions to access or convert some of the data. Auxiliary functions often use embedded C to do things that cannot be done in the SystemTap language, like access structure fields in some contexts, follow linked lists, etc. You can use auxiliary functions defined in other tapsets or write your own.

In the following example, `copy_process()` returns a pointer to the `task_struct` for the new process. Note that the process ID of the new process is retrieved by calling `task_pid()` and passing it the `task_struct` pointer. In this case, the auxiliary function is an embedded C function defined in `task.stp`.

```
probe kprocess.create = kernel.function("copy_process").return
{
    task = $return
    new_pid = task_pid(task)
}
```

It is not advisable to write probes for every function. Most SystemTap users will not need or understand them. Keep your tapsets simple and high-level.

[Report a bug](#)

2.2. Elements of a Tapset

The following sections describe the most important aspects of writing a tapset. Most of the content herein is suitable for developers who wish to contribute to SystemTap's upstream library of tapsets.

[Report a bug](#)

2.2.1. Tapset Files

Tapset files are stored in `src/tapset/` of the SystemTap GIT directory. Most tapset files are kept at that level. If you have code that only works with a specific architecture or kernel version, you may choose to put your tapset in the appropriate subdirectory.

Installed tapsets are located in `/usr/share/systemtap/tapset/` or `/usr/local/share/systemtap/tapset`.

Personal tapsets can be stored anywhere. However, to ensure that SystemTap can use them, use `-I tapset_directory` to specify their location when invoking `stap`.

[Report a bug](#)

2.2.2. Namespace

Probe alias names should take the form `tapset_name.probe_name`. For example, the probe for sending a signal could be named `signal.send`.

Global symbol names (probes, functions, and variables) should be unique across all tapsets. This helps avoid namespace collisions in scripts that use multiple tapsets. To ensure this, use tapset-specific prefixes in your global symbols.

Internal symbol names should be prefixed with an underscore (_).

[Report a bug](#)

2.2.3. Comments and Documentation

All probes and functions should include comment blocks that describe their purpose, the data they provide, and the context in which they run (e.g. interrupt, process, etc). Use comments in areas where your intent may not be clear from reading the code.

Note that specially-formatted comments are automatically extracted from most tapsets and included in this guide. This helps ensure that tapset contributors can write their tapset *and* document it in the same place. The specified format for documenting tapsets is as follows:

```
/**
 * probe tapset.name - Short summary of what the tapset does.
 * @argument: Explanation of argument.
 * @argument2: Explanation of argument2. Probes can have multiple
arguments.
 *
 * Context:
 * A brief explanation of the tapset context.
 * Note that the context should only be 1 paragraph short.
 *
 * Text that will appear under "Description."
 *
 * A new paragraph that will also appear under the heading
"Description".
 *
 * Header:
 * A paragraph that will appear under the heading "Header".
 **/
```

For example:

```
/**
 * probe vm.write_shared_copy- Page copy for shared page write.
 * @address: The address of the shared write.
 * @zero: Boolean indicating whether it is a zero page
 *         (can do a clear instead of a copy).
 *
 * Context:
 * The process attempting the write.
 *
 * Fires when a write to a shared page requires a page copy. This is
 * always preceded by a vm.shared_write.
 **/
```

To override the automatically-generated **Synopsis** content, use:

```
* Synopsis:
* New Synopsis string
*
```

For example:


```
/**
 * probe signal.handle - Fires when the signal handler is invoked
 * @sig: The signal number that invoked the signal handler
 *
 * Synopsis:
 * <programlisting>static int handle_signal(unsigned long sig, siginfo_t
*info, struct k_sigaction *ka,
 * sigset_t *oldset, struct pt_regs * regs)</programlisting>
 */
```

It is recommended that you use the **<programlisting>** tag in this instance, since overriding the **Synopsis** content of an entry does not automatically form the necessary tags.

For the purposes of improving the DocBook XML output of your comments, you can also use the following XML tags in your comments:

- ✧ **command**
- ✧ **emphasis**
- ✧ **programlisting**
- ✧ **remark** (tagged strings will appear in Publican beta builds of the document)

[Report a bug](#)

Chapter 3. Context Functions

The context functions provide additional information about where an event occurred. These functions can provide information such as a backtrace to where the event occurred and the current register values for the processor.

Name

`function::print_regs` — Print a register dump.

Synopsis

```
function print_regs()
```

Arguments

None

General Syntax

`print_regs`

Description

This function prints a register dump.

Name

`function::execname` — Returns the execname of a target process (or group of processes).

Synopsis

```
function execname:string()
```

Arguments

None

General Syntax

execname:**string**

Description

Returns the execname of a target process (or group of processes).

Name

function::pid — Returns the ID of a target process.

Synopsis

```
function pid:long()
```

Arguments

None

General Syntax

pid:**long**

Description

This function returns the ID of a target process.

Name

function::tid — Returns the thread ID of a target process.

Synopsis

```
function tid:long()
```

Arguments

None

General Syntax

tid:**long**

Description

This function returns the thread ID of the target process.

Name

function::ppid — Returns the process ID of a target process's parent process.

Synopsis

```
function ppid:long()
```

Arguments

None

General Syntax

ppid:**long**

Description

This function return the process ID of the target process's parent process.

Name

function::pgrp — Returns the process group ID of the current process.

Synopsis

```
function pgrp:long()
```

Argument s

None

General Syntax

pgrp:**long**

Description

This function returns the process group ID of the current process.

Name

function::sid — Returns the session ID of the current process.

Synopsis

```
function sid:long()
```

Arguments

None

General Syntax

sid:**long**

Description

The session ID of a process is the process group ID of the session leader. Session ID is stored in the `signal_struct` since Kernel 2.6.0.

Name

`function::pexecname` — Returns the execname of a target process's parent process.

Synopsis

```
function pexecname:string()
```

Arguments

None

General Syntax

pexecname:**string**

Description

This function returns the execname of a target process's parent process.

Name

function::gid — Returns the group ID of a target process.

Synopsis

```
function gid:long()
```

Arguments

None

General Syntax

gid:long

Description

This function returns the group ID of a target process.

Name

function::egid — Returns the effective gid of a target process.

Synopsis

```
function egid:long()
```

Arguments

None

General Syntax

egid:**long**

Description

This function returns the effective gid of a target process

Name

function::uid — Returns the user ID of a target process.

Synopsis

```
function uid:long()
```

Arguments

None

General Syntax

uid:**long**

Description

This function returns the user ID of the target process.

Name

`function::euid` — Return the effective uid of a target process.

Synopsis

```
function euid:long()
```

Arguments

None

General Syntax

`euid:long`

Description

Returns the effective user ID of the target process.

Name

`function::is_myproc` — Determines if the current probe point has occurred in the user's own process.

Synopsis

```
function is_myproc:long()
```

Arguments

None

General Syntax

is_myproc:**long**

Description

This function returns 1 if the current probe point has occurred in the user's own process.

Name

function::cpu — Returns the current cpu number.

Synopsis

```
function cpu:long()
```

Argument s

None

General Syntax

cpu:**long**

Description

This function returns the current cpu number.

Name

function::pp — Returns the active probe point.

Synopsis

```
function pp:string()
```

Arguments

None

General Syntax

`pp:string`

Description

This function returns the fully-resolved probe point associated with a currently running probe handler, including alias and wild-card expansion effects. Context: The current probe point.

Name

`function::registers_valid` — Determines validity of **register** and **u_register** in current context.

Synopsis

```
function registers_valid:long()
```

Arguments

None

General Syntax

`registers_valid:long`

Description

This function returns 1 if **register** and **u_register** can be used in the current context, or 0 otherwise. For example, **registers_valid** returns 0 when called from a begin or end probe.

Name

function::user_mode — Determines if probe point occurs in user-mode.

Synopsis

```
function user_mode:long()
```

Arguments

None

General Syntax

user_mode:**long**

Return 1 if the probe point occurred in user-mode.

Name

function::is_return — Whether the current probe context is a return probe.

Synopsis

```
function is_return:long()
```

Arguments

None

General Syntax

is_return:**long**

Description

Returns 1 if the current probe context is a return probe, returns 0 otherwise.

Name

function::target — Return the process ID of the target process.

Synopsis

```
function target:long()
```

Arguments

None

General Syntax

target:**long**

Description

This function returns the process ID of the target process. This is useful in conjunction with the -x PID or -c CMD command-line options to stap. An example of its use is to create scripts that filter on a specific process.

Name

function::module_name — The module name of the current script.

Synopsis

```
function module_name:string()
```

Arguments

None

General Syntax

module_name:**string**

Description

This function returns the name of the stap module. Either generated randomly (stap_[0-9a-f]+_[0-9a-f]+) or set by stap -m <module_name>.

Name

function::stp_pid — The process id of the stapio process.

Synopsis

```
function stp_pid:long()
```

Arguments

None

General Syntax

stp_pid:**long**

Description

This function returns the process id of the stapio process that launched this script. There could be other SystemTap scripts and stapio processes running on the system.

Name

function::stack_size — Return the size of the kernel stack.

Synopsis

```
function stack_size:long()
```

Argument s

None

General Syntax

stack_size:**long**

Description

This function returns the size of the kernel stack.

Name

function::stack_used — Returns the amount of kernel stack used.

Synopsis

```
function stack_used:long()
```

Arguments

None

General Syntax

stack_used:**long**

Description

This function determines how many bytes are currently used in the kernel stack.

Name

function::stack_unused — Returns the amount of kernel stack currently available.

Synopsis

```
function stack_unused:long()
```

Arguments

None

General Syntax

stack_unused:**long**

Description

This function determines how many bytes are currently available in the kernel stack.

Name

function::uaddr — User space address of current running task. EXPERIMENTAL.

Synopsis

```
function uaddr:long()
```

Arguments

None

General Syntax

uaddr:**long**

Description

Returns the address in userspace that the current task was at when the probe occurred. When the current running task isn't a user space thread, or the address cannot be found, zero is returned. Can be used to see where the current task is combined with **usymname** or **syndata**. Often the task will be in the VDSO where it entered the kernel. FIXME - need VDSO tracking support #10080.

Name

function::cmdline_args — Fetch command line arguments from current process

Synopsis

```
function cmdline_args:string(n:long,m:long,delim:string)
```

Arguments

n

First argument to get (zero is the command itself)

m

Last argument to get (or minus one for all arguments after *n*)

delim

String to use to delimit arguments when more than one.

General Syntax

```
cmdline_arg:string(n:long, m:long, delim:string)
```

Description

Returns arguments from the current process starting with argument number *n*, up to argument *m*. If there are less than *n* arguments, or the arguments cannot be retrieved from the current process, the empty string is returned. If *m* is smaller than *n* then all arguments starting from argument *n* are returned. Argument zero is traditionally the command itself.

Name

`function::cmdline_arg` — Fetch a command line argument.

Synopsis

```
function cmdline_arg:string(n:long)
```

Arguments

n

Argument to get (zero is the command itself)

General Syntax

```
cmdline_arg:string(n:long)
```

Description

Returns argument the requested argument from the current process or the empty string when there are not that many arguments or there is a problem retrieving the argument. Argument zero is traditionally the command itself.

Name

function::cmdline_str — Fetch all command line arguments from current process

Synopsis

```
function cmdline_str:string()
```

Arguments

None

General Syntax

```
cmdline_str:string
```

Description

Returns all arguments from the current process delimited by spaces. Returns the empty string when the arguments cannot be retrieved.

Name

function::env_var — Fetch environment variable from current process

Synopsis

```
function env_var:string(name:string)
```

Arguments

name

Name of the environment variable to fetch

General Syntax

```
env_var:string(name:string)
```

Description

Returns the contents of the specified environment value for the current process. If the variable isn't set an empty string is returned.

Name

function::print_stack — Print out kernel stack from string.

Synopsis

```
function print_stack(stk:string)
```

Arguments

stk

String with list of hexadecimal addresses.

General Syntax

```
print_stack(stk:string)
```

Description

This function performs a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to **backtrace**.

Print one line per address, including the address, the name of the function containing the address, and an estimate of its position within that function. Return nothing.

Name

function::sprint_stack — Return stack for kernel addresses from string. EXPERIMENTAL!

Synopsis

```
function sprint_stack:string(stk:string)
```

Arguments

stk

String with list of hexadecimal (kernel) addresses.

Description

Perform a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to **backtrace**.

Returns a simple backtrace from the given hex string. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use `print_stack`.

Name

`function::probefunc` — Return the probe point's function name, if known.

Synopsis

```
function probefunc:string()
```

Arguments

None

General Syntax

`probefunc:string`

Description

This function returns the name of the function being probed. It will do this based on the probe point string as returned by `pp`.

Please note

this function is deprecated, please use `symname` and/or `usymname`. This function might return a function name based on the current address if the probe point context couldn't be parsed.

Name

`function::probemod` — Return the probe point's kernel module name.

Synopsis

```
function probemod:string()
```

Arguments

None

General Syntax

probemod:**string**

Description

This function returns the name of the kernel module containing the probe point, if known.

Name

function::modname — Return the kernel module name loaded at the address.

Synopsis

```
function modname:string(addr:long)
```

Arguments

addr

The address.

Description

Returns the module name associated with the given address if known. If not known it will return the string “<unknown>”. If the address was not in a kernel module, but in the kernel itself, then the string “kernel” will be returned.

Name

function::symname — Return the kernel symbol associated with the given address.

Synopsis

```
function symname:string(addr:long)
```

Arguments

addr

The address to translate.

General Syntax

```
symname:string(addr:long)
```

Description

Returns the (function) symbol name associated with the given address if known. If not known it will return the hex string representation of *addr*.

Name

`function::symdata` — Return the kernel symbol and module offset for the address.

Synopsis

```
function symdata:string(addr:long)
```

Arguments

addr

The address to translate.

General Syntax

```
symdata:string(addr:long)
```

Description

Returns the (function) symbol name associated with the given address if known, the offset from the start and size of the symbol, plus module name (between brackets). If symbol is unknown, but module is known, the offset inside the module, plus the size of the module is added. If any element is not known it will be omitted and if the symbol name is unknown it will return the hex string for the given address.

Name

function::usymname — Return the symbol of an address in the current task. EXPERIMENTAL!

Synopsis

```
function usymname:string(addr:long)
```

Arguments

addr

The address to translate.

Description

Returns the (function) symbol name associated with the given address if known. If not known it will return the hex string representation of `addr`.

Name

function::usymdata — Return the symbol and module offset of an address. EXPERIMENTAL!

Synopsis

```
function usymdata:string(addr:long)
```

Arguments

addr

The address to translate.

Description

Returns the (function) symbol name associated with the given address in the current task if known, the offset from the start and the size of the symbol, plus the module name (between brackets). If symbol is unknown, but module is known, the offset inside the module, plus the size of the module is added. If any element is not known it will be omitted and if the symbol name is unknown it will return the hex string for the given address.

Name

function::print_ustack — Print out stack for the current task from string. EXPERIMENTAL!

Synopsis

```
function print_ustack(stk:string)
```

Arguments

stk

String with list of hexadecimal addresses for the current task.

Description

Perform a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to **ubacktrace** for the current task.

Print one line per address, including the address, the name of the function containing the address, and an estimate of its position within that function. Return nothing.

Name

`function::sprint_ustack` — Return stack for the current task from string. EXPERIMENTAL!

Synopsis

```
function sprint_ustack:string(stk:string)
```

Arguments

stk

String with list of hexadecimal addresses for the current task.

Description

Perform a symbolic lookup of the addresses in the given string, which is assumed to be the result of a prior call to **ubacktrace** for the current task.

Returns a simple backtrace from the given hex string. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to MAXSTRINGLEN, to print fuller and richer stacks use `print_ustack`.

Name

`function::print_backtrace` — Print stack back trace

Synopsis

```
function print_backtrace()
```

Arguments

None

General Syntax

`print_backtrace`

Description

This function is equivalent to `print_stack(backtrace)`, except that deeper stack nesting may be supported. The function does not return a value.

Name

`function::sprint_backtrace` — Return stack back trace as string. EXPERIMENTAL!

Synopsis

```
function sprint_backtrace:string()
```

Arguments

None

Description

Returns a simple (kernel) backtrace. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to `MAXSTRINGLEN`, to print fuller and richer stacks use `print_backtrace`. Equivalent to `sprint_stack(backtrace)`, but more efficient (no need to translate between hex strings and final backtrace string).

Name

name

function::backtrace — Hex backtrace of current stack

Synopsis

```
function backtrace:string()
```

Arguments

None

General Syntax

backtrace:**string**

Description

This function returns a string of hex addresses that are a backtrace of the stack. Output may be truncated as per maximum string length (MAXSTRINGLEN).

Name

function::task_backtrace — Hex backtrace of an arbitrary task

Synopsis

```
function task_backtrace:string(task:long)
```

Arguments

task

pointer to task_struct

General Syntax

task_backtrace:string(task:long)

Description

This function returns a string of hex addresses that are a backtrace of the stack of a particular task. Output may be truncated as per maximum string length.

Name

function::caller — Return name and address of calling function

Synopsis

```
function caller:string()
```

Arguments

None

General Syntax

caller:**string**

Description

This function returns the address and name of the calling function. This is equivalent to calling: `sprintf("s 0xx", symname(caller_addr, caller_addr))` Works only for return probes at this time.

Name

`function::caller_addr` — Return caller address

Synopsis

```
function caller_addr:long()
```

Arguments

None

General Syntax

`caller_addr:long`

Description

This function returns the address of the calling function. Works only for return probes at this time.

Name

`function::print_ubacktrace` — Print stack back trace for current task. EXPERIMENTAL!

Synopsis

```
function print_ubacktrace()
```

Arguments

None

Description

Equivalent to `print_ustack(ubacktrace)`, except that deeper stack nesting may be supported. Returns nothing.

Note

To get (full) backtraces for user space applications and shared libraries not mentioned in the current script run `stap` with `-d /path/to/exe-or-so` and/or add `--ldd` to load all needed unwind data.

Name

`function::sprint_ubacktrace` — Return stack back trace for current task as string. EXPERIMENTAL!

Synopsis

```
function sprint_ubacktrace:string()
```

Arguments

None

Description

Returns a simple backtrace for the current task. One line per address. Includes the symbol name (or hex address if symbol couldn't be resolved) and module name (if found). Includes the offset from the start of the function if found, otherwise the offset will be added to the module (if found, between brackets). Returns the backtrace as string (each line terminated by a newline character). Note that the returned stack will be truncated to `MAXSTRINGLEN`, to print fuller and richer stacks use `print_ubacktrace`. Equivalent to `sprint_ustack(ubacktrace)`, but more efficient (no need to translate between hex strings and final backtrace string).

Note

To get (full) backtraces for user space applications and shared libraries not mentioned in the current script run `stap` with `-d /path/to/exe-or-so` and/or add `--ldd` to load all needed unwind data.

Name

function::print_ubacktrace_brief — Print stack back trace for current task. EXPERIMENTAL!

Synopsis

```
function print_ubacktrace_brief()
```

Arguments

None

Description

Equivalent to **print_ubacktrace**, but output for each symbol is shorter (just name and offset, or just the hex address of no symbol could be found).

Note

To get (full) backtraces for user space applications and shared libraries not mentioned in the current script run stap with `-d /path/to/exe-or-so` and/or add `--ltd` to load all needed unwind data.

Name

function::ubacktrace — Hex backtrace of current task stack. EXPERIMENTAL!

Synopsis

```
function ubacktrace:string()
```

Arguments

None

Description

Return a string of hex addresses that are a backtrace of the stack of the current task. Output may be truncated as per maximum string length. Returns empty string when current probe point cannot determine user backtrace.

Note

To get (full) backtraces for user space applications and shared libraries not mentioned in the current script run stap with `-d /path/to/exe-or-so` and/or add `--ldd` to load all needed unwind data.

Name

`function::task_current` — The current `task_struct` of the current task.

Synopsis

```
function task_current:long()
```

Arguments

None

General Syntax

`task_current:long`

Description

This function returns the `task_struct` representing the current process. This address can be passed to the various `task_*`() functions to extract more task-specific data.

Name

`function::task_parent` — The `task_struct` of the parent task.

Synopsis

```
function task_parent:long(task:long)
```

Arguments

task

task_struct pointer.

General Syntax

```
task_parent:long(task:long)
```

Description

This function returns the parent task_struct of the given task. This address can be passed to the various task_*() functions to extract more task-specific data.

Name

function::task_state — The state of the task.

Synopsis

```
function task_state:long(task:long)
```

Arguments

task

task_struct pointer.

General Syntax

```
task_state:long(task:long)
```

Description

Return the state of the given task, one of: TASK_RUNNING (0), TASK_INTERRUPTIBLE (1), TASK_UNINTERRUPTIBLE (2), TASK_STOPPED (4), TASK_TRACED (8), EXIT_ZOMBIE (16), EXIT_DEAD (32).

Name

function::task_execname — The name of the task.

Synopsis

```
function task_execname:string(task:long)
```

Arguments

task

task_struct pointer.

General Syntax

```
task_execname:string(task:long)
```

Description

Return the name of the given task.

Name

function::task_pid — The process identifier of the task.

Synopsis

```
function task_pid:long(task:long)
```

Argument s

task

task_struct pointer.

General Syntax

```
task_pid:long (task:long)
```

Description

This function returns the process id of the given task.

Name

function::pid2task — The task_struct of the given process identifier.

Synopsis

```
function pid2task:long(pid:long)
```

Argument s

pid

Process identifier.

Description

Return the task struct of the given process id.

Name

function::pid2execname — The name of the given process identifier.

Synopsis

```
function pid2execname:string(pid:long)
```

Arguments

pid

Process identifier.

Description

Return the name of the given process id.

Name

function::task_tid — The thread identifier of the task.

Synopsis

```
function task_tid:long(task:long)
```

Arguments

task

task struct pointer.

```
task_struct pointer.
```

General Syntax

```
task_tid:long(task:long)
```

Description

This function returns the thread id of the given task.

Name

function::task_gid — The group identifier of the task.

Synopsis

```
function task_gid:long(task:long)
```

Argument s

task

task_struct pointer.

General Syntax

```
task_gid:long(task:long)
```

Description

This function returns the group id of the given task.

Name

function::task_egid — The effective group identifier of the task.

Synopsis

```
function task_egid:long(task:long)
```

Arguments

task

task_struct pointer.

General Syntax

```
task_egid:long(task:long)
```

Description

This function returns the effective group id of the given task.

Name

function::task_uid — The user identifier of the task.

Synopsis

```
function task_uid:long(task:long)
```

Arguments

task

task_struct pointer.

General Syntax

```
task_uid:long(task:long)
```

Description

This function returns the user id of the given task.

Name

function::task_uid — The effective user identifier of the task.

Synopsis

```
function task_uid:long(task:long)
```

Argument s

task

task_struct pointer.

General Syntax

```
task_uid:long(task:long)
```

Description

This function returns the effective user id of the given task.

Name

function::task_prio — The priority value of the task.

Synopsis

```
function task_prio:long(task:long)
```

Argument s

task

task_struct pointer.

General Syntax

```
task_prio:long(task:long)
```

Description

This function returns the priority value of the given task.

Name

function::task_nice — The nice value of the task.

Synopsis

```
function task_nice:long(task:long)
```

Argument s

task

task_struct pointer.

General Syntax

```
task_nice:long(task:long)
```

Description

This function returns the nice value of the given task.

Name

function::task_cpu — The scheduled cpu of the task.

Synopsis

```
function task_cpu:long(task:long)
```

Arguments

task

task_struct pointer.

General Syntax

```
task_cpu:long(task:long)
```

Description

This function returns the scheduled cpu for the given task.

Name

function::task_open_file_handles — The number of open files of the task.

Synopsis

```
function task_open_file_handles:long(task:long)
```

Arguments

task

task_struct pointer.

General Syntax

```
task_open_file_handles:long(task:long)
```

Description

This function returns the number of open file handlers for the given task.

Name

function::task_max_file_handles — The max number of open files for the task.

Synopsis

```
function task_max_file_handles:long(task:long)
```

Arguments

task

task_struct pointer.

General Syntax

```
task_max_file_handles:long(task:long)
```

Description

This function returns the maximum number of file handlers for the given task.

Name

`function::pn` — Returns the active probe name.

Synopsis

```
function pn:string()
```

Arguments

None

General Syntax

`pn:string`

Description

This function returns the script-level probe point associated with a currently running probe handler, including wild-card expansion effects. Context: The current probe point.

[Report a bug](#)

Chapter 4. Timestamp Functions

Each timestamp function returns a value to indicate when a function is executed. These returned values can then be used to indicate when an event occurred, provide an ordering for events, or compute the amount of time elapsed between two time stamps.

Name

function::get_cycles — Processor cycle count.

Synopsis

```
function get_cycles:long()
```

Arguments

None

General Syntax

get_cycles:**long**

Description

This function returns the processor cycle counter value if available, else it returns zero. The cycle counter is free running and unsynchronized on each processor. Thus, the order of events cannot be determined by comparing the results of the get_cycles function on different processors.

Name

function::gettimeofday_ns — Number of nanoseconds since UNIX epoch.

Synopsis

```
function gettimeofday_ns:long()
```

Arguments

None

General Syntax

gettimeofday_ns:**long**

Description

This function returns the number of nanoseconds since the UNIX epoch.

Name

function::gettimeofday_us — Number of microseconds since UNIX epoch.

Synopsis

```
function gettimeofday_us:long()
```

Arguments

None

General Syntax

gettimeofday_us:**long**

Description

This function returns the number of microseconds since the UNIX epoch.

Name

function::gettimeofday_ms — Number of milliseconds since UNIX epoch.

Synopsis

```
function gettimeofday_ms:long()
```

Arguments

None

General Syntax

gettimeofday_ms:**long**

Description

This function returns the number of milliseconds since the UNIX epoch.

Name

function::gettimeofday_s — Number of seconds since UNIX epoch.

Synopsis

```
function gettimeofday_s:long()
```

Arguments

None

General Syntax

`gettimeofday_s:long`

Description

This function returns the number of seconds since the UNIX epoch.

[Report a bug](#)

Chapter 5. Time string utility function

Utility function to turn seconds since the epoch (as returned by the timestamp function `gettimeofday_s()`) into a human readable date/time string.

Name

`function::ctime` — Convert seconds since epoch into human readable date/time string.

Synopsis

```
function ctime:string(epochsecs:long)
```

Arguments

epochsecs

Number of seconds since epoch (as returned by `gettimeofday_s`).

General Syntax

```
ctime:string(epochsecs:long)
```

Description

Takes an argument of seconds since the epoch as returned by `gettimeofday_s`. Returns a string of the form

```
“Wed Jun 30 21:49:08 1993”
```

The string will always be exactly 24 characters. If the time would be unreasonable far in the past (before what can be represented with a 32 bit offset in seconds from the epoch) the returned string will be “a long, long time ago...”. If the time would be unreasonable far in the future the returned string will be “far, far in the future...” (both these strings are also 24 characters wide).

Note that the epoch (zero) corresponds to

```
“Thu Jan 1 00:00:00 1970”
```

The earliest full date given by `ctime`, corresponding to `epochsecs -2147483648` is “Fri Dec 13 20:45:52 1901”. The latest full date given by `ctime`, corresponding to `epochsecs 2147483647` is “Tue Jan 19 03:14:07 2038”.

The abbreviations for the days of the week are Sun, Mon, Tue, Wed, Thu, Fri, and Sat. The abbreviations for the months are Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, and Dec.

Note that the real C library `ctime` function puts a newline ('\n') character at the end of the string that this function does not. Also note that since the kernel has no concept of timezones, the returned time is always in GMT.

[Report a bug](#)

Chapter 6. Memory Tapset

This family of probe points is used to probe memory-related events or query the memory usage of the current process. It contains the following probe points:

Name

function::vm_fault_contains — Test return value for page fault reason

Synopsis

```
function vm_fault_contains:long(value:long, test:long)
```

Arguments

value

The `fault_type` returned by `vm.page_fault.return`

test

The type of fault to test for (`VM_FAULT_OOM` or similar)

Name

probe::vm.pagefault — Records that a page fault occurred.

Synopsis

```
vm.pagefault
```

Values

write_access

Indicates whether this was a write or read access; 1 indicates a write, while 0 indicates a read.

name

Name of the probe point

address

The address of the faulting memory access; i.e. the address that caused the page fault.

Context

The process which triggered the fault

Name

probe::vm.pagefault.return — Indicates what type of fault occurred.

Synopsis

```
vm.pagefault.return
```

Values

name

Name of the probe point

fault_type

Returns either 0 (VM_FAULT_OOM) for out of memory faults, 2 (VM_FAULT_MINOR) for minor faults, 3 (VM_FAULT_MAJOR) for major faults, or 1 (VM_FAULT_SIGBUS) if the fault was neither OOM, minor fault, nor major fault.

Name

function::addr_to_node — Returns which node a given address belongs to within a NUMA system.

Synopsis

```
function addr_to_node:long(addr:long)
```

Arguments

addr

The address of the faulting memory access.

General Syntax

```
addr_to_node:long(addr:long)
```

Description

This function accepts an address, and returns the node that the given address belongs to in a NUMA system.

Name

probe::vm.write_shared — Attempts at writing to a shared page.

Synopsis

```
vm.write_shared
```

Values

name

Name of the probe point

address

The address of the shared write.

Context

The context is the process attempting the write.

Description

Fires when a process attempts to write to a shared page. If a copy is necessary, this will be followed by a `vm.write_shared_copy`.

Name

`probe::vm.write_shared_copy` — Page copy for shared page write.

Synopsis

```
vm.write_shared_copy
```

Values

name

Name of the probe point

zero

Boolean indicating whether it is a zero page (can do a clear instead of a copy).

address

The address of the shared write.

Context

The process attempting the write.

Description

Fires when a write to a shared page requires a page copy. This is always preceded by a `vm.shared_write`.

Name

probe::vm.mmap — Fires when an mmap is requested.

Synopsis

```
vm.mmap
```

Values

length

The length of the memory segment

name

Name of the probe point

address

The requested address

Context

The process calling mmap.

Name

probe::vm.munmap — Fires when an munmap is requested.

Synopsis

```
vm.munmap
```

Values

length

The length of the memory segment

name

Name of the probe point

address

The requested address

Context

The process calling munmap.

Name

probe::vm.brk — Fires when a brk is requested (i.e. the heap will be resized).

Synopsis

vm.brk

Values

length

The length of the memory segment

name

Name of the probe point

address

The requested address

Context

The process calling brk.

Name

name

probe::vm.oom_kill — Fires when a thread is selected for termination by the OOM killer.

Synopsis

```
vm.oom_kill
```

Values

name

Name of the probe point

task

The task being killed

Context

The process that tried to consume excessive memory, and thus triggered the OOM.

Name

probe::vm.kmalloc — Fires when kmalloc is requested.

Synopsis

```
vm.kmalloc
```

Values

ptr

Pointer to the kmemory allocated

caller_function

Name of the caller function.

call_site

Address of the kmemory function.

gfp_flag_name

type of kmemory to allocate (in String format)

name

Name of the probe point

bytes_req

Requested Bytes

bytes_alloc

Allocated Bytes

gfp_flags

type of kmemory to allocate

Name

probe::vm.kmem_cache_alloc — Fires when \

Synopsis

```
vm.kmem_cache_alloc
```

Values

ptr

Pointer to the kmemory allocated

caller_function

Name of the caller function.

call_site

Address of the function calling this kmemory function.

gfp_flag_name

Type of kmemory to allocate(in string format)

name

Name of the probe point

bytes_req

Requested Bytes

bytes_alloc

Allocated Bytes

gfp_flags

type of kmemory to allocate

Description

kmem_cache_alloc is requested.

Name

probe::vm.kmalloc_node — Fires when kmalloc_node is requested.

Synopsis

```
vm.kmalloc_node
```

Values

ptr

Pointer to the kmemory allocated

caller_function

Name of the caller function.

call_site

Address of the function calling this kmemory function.

gfp_flag_name

Type of kmemory to allocate(in string format)

name

Name of the probe point

bytes_req

Requested Bytes

bytes_alloc

Allocated Bytes

gfp_flags

type of kmemory to allocate

Name

probe::vm.kmem_cache_alloc_node — Fires when \

Synopsis

vm.kmem_cache_alloc_node

Values***ptr***

Pointer to the kmemory allocated

caller_function

Name of the caller function.

call_site

Address of the function calling this kmemory function.

gfp_flag_name

Type of kmemory to allocate(in string format)

name

Name of the probe point

bytes_req

Requested Bytes

bytes_alloc

Allocated Bytes

gfp_flags

type of kmemory to allocate

Description

kmem_cache_alloc_node is requested.

Name

probe::vm.kfree — Fires when kfree is requested.

Synopsis

```
vm.kfree
```

Values

ptr

Pointer to the kmemory allocated which is returned by kmalloc

caller_function

Name of the caller function.

call_site

Address of the function calling this kmemory function.

name

Name of the probe point

Name

probe::vm.kmem_cache_free — Fires when \

Synopsis

```
vm.kmem_cache_free
```

Values

ptr

Pointer to the kmemory allocated which is returned by `kmem_cache`

caller_function

Name of the caller function.

call_site

Address of the function calling this kmemory function.

name

Name of the probe point

Description

`kmem_cache_free` is requested.

Name

`function::proc_mem_size` — Total program virtual memory size in pages

Synopsis

```
function proc_mem_size:long()
```

Arguments

None

Description

Returns the total virtual memory size in pages of the current process, or zero when there is no current process or the number of pages couldn't be retrieved.

Name

function::proc_mem_size_pid — Total program virtual memory size in pages

Synopsis

```
function proc_mem_size_pid:long(pid:long)
```

Arguments

pid

The pid of process to examine

Description

Returns the total virtual memory size in pages of the given process, or zero when that process doesn't exist or the number of pages couldn't be retrieved.

Name

function::proc_mem_rss — Program resident set size in pages

Synopsis

```
function proc_mem_rss:long()
```

Arguments

None

Description

Returns the resident set size in pages of the current process, or zero when there is no current process or the number of pages couldn't be retrieved.

Name

function::proc_mem_rss_pid — Program resident set size in pages

Synopsis

```
function proc_mem_rss_pid:long(pid:long)
```

Argument s

pid

The pid of process to examine

Description

Returns the resident set size in pages of the given process, or zero when the process doesn't exist or the number of pages couldn't be retrieved.

Name

function::proc_mem_shr — Program shared pages (from shared mappings)

Synopsis

```
function proc_mem_shr:long()
```

Argument s

None

Description

Returns the shared pages (from shared mappings) of the current process, or zero when there is no current process or the number of pages couldn't be retrieved.

Name

function::proc_mem_shr_pid — Program shared pages (from shared mappings)

Synopsis

```
function proc_mem_shr_pid:long(pid:long)
```

Arguments

pid

The pid of process to examine

Description

Returns the shared pages (from shared mappings) of the given process, or zero when the process doesn't exist or the number of pages couldn't be retrieved.

Name

function::proc_mem_txt — Program text (code) size in pages

Synopsis

```
function proc_mem_txt:long()
```

Arguments

None

Description

Returns the current process text (code) size in pages, or zero when there is no current process or the number of pages couldn't be retrieved.

Name

function::proc_mem_txt_pid — Program text (code) size in pages

Synopsis

```
function proc_mem_txt_pid:long(pid:long)
```

Arguments

pid

The pid of process to examine

Description

Returns the given process text (code) size in pages, or zero when the process doesn't exist or the number of pages couldn't be retrieved.

Name

function::proc_mem_data — Program data size (data + stack) in pages

Synopsis

```
function proc_mem_data:long()
```

Arguments

None

Description

Returns the current process data size (data + stack) in pages, or zero when there is no current process or the number of pages couldn't be retrieved.

Name

function::proc_mem_data_pid — Program data size (data + stack) in pages

Synopsis

```
function proc_mem_data_pid:long(pid:long)
```

Arguments

pid

The pid of process to examine

Description

Returns the given process data size (data + stack) in pages, or zero when the process doesn't exist or the number of pages couldn't be retrieved.

Name

function::mem_page_size — Number of bytes in a page for this architecture

Synopsis

```
function mem_page_size:long()
```

Arguments

None

Name

function::bytes_to_string — Human readable string for given bytes

Synopsis

```
function bytes_to_string:string(bytes:long)
```

Arguments

bytes

Number of bytes to translate.

Description

Returns a string representing the number of bytes (up to 1024 bytes), the number of kilobytes (when less than 1024K) postfixed by 'K', the number of megabytes (when less than 1024M) postfixed by 'M' or the number of gigabytes postfixed by 'G'. If representing K, M or G, and the number is amount is less than 100, it includes a '.' plus the remainder. The returned string will be 5 characters wide (padding with whitespace at the front) unless negative or representing more than 9999G bytes.

Name

function::pages_to_string — Turns pages into a human readable string

Synopsis

```
function pages_to_string:string(pages:long)
```

Arguments

pages

Number of pages to translate.

Description

Multiplies *pages* by **page_size** to get the number of bytes and returns the result of **bytes_to_string**.

Name

function::proc_mem_string — Human readable string of current proc memory usage

Synopsis

```
function proc_mem_string:string()
```

Arguments

None

Description

Returns a human readable string showing the size, rss, shr, txt and data of the memory used by the current process. For example “size: 301m, rss: 11m, shr: 8m, txt: 52k, data: 2248k”.

Name

function::proc_mem_string_pid — Human readable string of process memory usage

Synopsis

```
function proc_mem_string_pid:string(pid:long)
```

Arguments

pid

The pid of process to examine

Description

Returns a human readable string showing the size, rss, shr, txt and data of the memory used by the given process. For example “size: 301m, rss: 11m, shr: 8m, txt: 52k, data: 2248k”.

[Report a bug](#)

Chapter 7. Task Time Tapset

This tapset defines utility functions to query time related properties of the current tasks, translate those in milliseconds and human readable strings.

Name

function::task_untime — User time of the current task

Synopsis

```
function task_untime:long()
```

Arguments

None

Description

Returns the user time of the current task in cputime. Does not include any time used by other tasks in this process, nor does it include any time of the children of this task.

Name

function::task_untime_tid — User time of the given task

Synopsis

```
function task_untime_tid:long(tid:long)
```

Arguments

tid

Thread id of the given task

thread id of the given task

Description

Returns the user time of the given task in `cputime`, or zero if the task doesn't exist. Does not include any time used by other tasks in this process, nor does it include any time of the children of this task.

Name

`function::task_stime` — System time of the current task

Synopsis

```
function task_stime:long()
```

Argument s

None

Description

Returns the system time of the current task in `cputime`. Does not include any time used by other tasks in this process, nor does it include any time of the children of this task.

Name

`function::task_stime_tid` — System time of the given task

Synopsis

```
function task_stime_tid:long(tid:long)
```

Arguments

tid

Thread id of the given task

Description

Returns the system time of the given task in cputime, or zero if the task doesn't exist. Does not include any time used by other tasks in this process, nor does it include any time of the children of this task.

Name

function::cputime_to_msecs — Translates the given cputime into milliseconds

Synopsis

```
function cputime_to_msecs:long(cputime:long)
```

Arguments

cputime

Time to convert to milliseconds.

Name

function::msecs_to_string — Human readable string for given milliseconds

Synopsis

```
function msecs_to_string:string(msecs:long)
```

Arguments

msecs

Number of milliseconds to translate.

Description

Returns a string representing the number of milliseconds as a human readable string consisting of “XmY.ZZZs”, where X is the number of minutes, Y is the number of seconds and ZZZ is the number of milliseconds.

Name

function::cputime_to_string — Human readable string for given cputime

Synopsis

```
function cputime_to_string:string(cputime:long)
```

Arguments***cputime***

Time to translate.

Description

Equivalent to calling: msec_to_string (cputime_to_msecs (cputime)).

Name

function::task_time_string — Human readable string of task time usage

Synopsis

```
function task_time_string:string()
```

Arguments

None

Description

Returns a human readable string showing the user and system time the current task has used up to now. For example “usr: 0m12.908s, sys: 1m6.851s”.

Name

function::task_time_string_tid — Human readable string of task time usage

Synopsis

```
function task_time_string_tid:string(tid:long)
```

Arguments

tid

Thread id of the given task

Description

Returns a human readable string showing the user and system time the given task has used up to now. For example “usr: 0m12.908s, sys: 1m6.851s”.

[Report a bug](#)

Chapter 8. IO Scheduler and block IO Tapset

This family of probe points is used to probe block IO layer and IO scheduler activities. It contains the following probe points:

Name

probe::ioscheduler.elv_next_request — Fires when a request is retrieved from the request queue

Synopsis

```
ioscheduler.elv_next_request
```

Values

name

Name of the probe point

elevator_name

The type of I/O elevator currently enabled

Name

probe::ioscheduler.elv_next_request.return — Fires when a request retrieval issues a return signal

Synopsis

```
ioscheduler.elv_next_request.return
```

Values

disk_major

Disk major number of the request

rq

Address of the request

name

Name of the probe point

disk_minor

Disk minor number of the request

rq_flags

Request flags

Name

probe::ioscheduler.elv_completed_request — Fires when a request is completed

Synopsis

```
ioscheduler.elv_completed_request
```

Values

disk_major

Disk major number of the request

rq

Address of the request

name

Name of the probe point

elevator_name

The type of I/O elevator currently enabled

disk_minor

Disk minor number of the request

rq_flags

Request flags

Name

probe::ioscheduler.elv_add_request.kp — kprobe based probe to indicate that a request was added to the request queue

Synopsis

```
ioscheduler.elv_add_request.kp
```

Values

disk_major

Disk major number of the request

rq

Address of the request

q

pointer to request queue

name

Name of the probe point

elevator_name

The type of I/O elevator currently enabled

disk_minor

Disk minor number of the request

rq_flags

Request flags

Name

probe::ioscheduler.elv_add_request.tp — tracepoint based probe to indicate a request is added to the request queue.

Synopsis

```
ioscheduler.elv_add_request.tp
```

Values

disk_major

Disk major no of request.

rq

Address of request.

q

Pointer to request queue.

name

Name of the probe point

elevator_name

The type of I/O elevator currently enabled.

disk_minor

Disk minor number of request.

rq_flags

Request flags.

Name

probe::ioscheduler.elv_add_request — probe to indicate request is added to the request queue.

Synopsis

```
ioscheduler.elv_add_request
```

Values

disk_major

Disk major no of request.

rq

Address of request.

q

Pointer to request queue.

elevator_name

The type of I/O elevator currently enabled.

disk_minor

Disk minor number of request.

rq_flags

Request flags.

Name

probe::ioscheduler_trace.elv_completed_request — Fires when a request is

Synopsis

`ioscheduler_trace.elv_completed_request`

Values

disk_major

Disk major no of request.

rq

Address of request.

name

Name of the probe point

elevator_name

The type of I/O elevator currently enabled.

disk_minor

Disk minor number of request.

rq_flags

Request flags.

Description

completed.

Name

probe::ioscheduler_trace.elv_issue_request — Fires when a request is

Synopsis

```
ioscheduler_trace.elv_issue_request
```

Values

disk_major

Disk major no of request.

rq

Address of request.

name

Name of the probe point

elevator_name

The type of I/O elevator currently enabled.

disk_minor

Disk minor number of request.

rq_flags

Request flags.

Description

scheduled.

Name

probe::ioscheduler_trace.elv_requeue_request — Fires when a request is

Synopsis

```
ioscheduler_trace.elv_requeue_request
```

Values

disk_major

Disk major no of request.

rq

Address of request.

name

Name of the probe point

elevator_name

The type of I/O elevator currently enabled.

disk_minor

Disk minor number of request.

rq_flags

Request flags.

Description

put back on the queue, when the hardware cannot accept more requests.

Name

probe::ioscheduler_trace.elv_abort_request — Fires when a request is aborted.

Synopsis

```
ioscheduler_trace.elv_abort_request
```

Values

disk_major

Disk major no of request.

rq

Address of request.

name

Name of the probe point

elevator_name

The type of I/O elevator currently enabled.

disk_minor

Disk minor number of request.

rq_flags

Request flags.

Name

probe::ioscheduler_trace.plugin — Fires when a request queue is plugged;

Synopsis

```
ioscheduler_trace.plugin
```

Values

name

Name of the probe point

rq_queue

request queue

Description

ie, requests in the queue cannot be serviced by block driver.

Name

probe::ioscheduler_trace.unplug_io — Fires when a request queue is unplugged;

Synopsis

```
ioscheduler_trace.unplug_io
```

Values

name

Name of the probe point

rq_queue

request queue

Description

Either, when number of pending requests in the queue exceeds threshold or, upon expiration of timer that was activated when queue was plugged.

Name

probe::ioscheduler_trace.unplug_timer — Fires when unplug timer associated

Synopsis

```
ioscheduler_trace.unplug_timer
```

Values

name

Name of the probe point

rq_queue

request queue

Description

with a request queue expires.

Name

probe::ioblock.request — Fires whenever making a generic block I/O request.

Synopsis

```
ioblock.request
```

Values

None

Description

name - name of the probe point **devname** - block device name **ino** - i-node number of the mapped file **sector** - beginning sector for the entire bio **flags** - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-of-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

rw - binary trace for read/write request ***vcnt*** - bio vector count which represents number of array element (page, offset, length) which make up this I/O request ***idx*** - offset into the bio vector array ***phys_segments*** - number of segments in this bio after physical address coalescing is performed ***hw_segments*** - number of segments after physical and DMA remapping hardware coalescing is performed ***size*** - total size in bytes ***bdev*** - target block device ***bdev_contains*** - points to the device object which contains the partition (when bio structure represents a partition) ***p_start_sect*** - points to the start sector of the partition structure of the device

Context

The process makes block I/O request

Name

probe::ioblock.end — Fires whenever a block I/O transfer is complete.

Synopsis

```
ioblock.end
```

Values

None

Description

name - name of the probe point ***devname*** - block device name ***ino*** - i-node number of the mapped file ***bytes_done*** - number of bytes transferred ***sector*** - beginning sector for the entire bio ***flags*** - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-of-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported ***error*** - 0 on success ***rw*** - binary trace for read/write request ***vcnt*** - bio vector count which represents number of array element (page, offset, length) which makes up this I/O request ***idx*** - offset into the bio vector array ***phys_segments*** - number of segments in this bio after physical address coalescing is performed. ***hw_segments*** - number of segments after physical and DMA remapping hardware coalescing is performed ***size*** - total size in bytes

Context

The process signals the transfer is done.

Name

probe::ioblock_trace.bounce — Fires whenever a buffer bounce is needed for at least one page of a block IO request.

Synopsis

```
ioblock_trace.bounce
```

Values

None

Description

name - name of the probe point **q** - request queue on which this bio was queued. **devname** - device for which a buffer bounce was needed. **ino** - i-node number of the mapped file **bytes_done** - number of bytes transferred **sector** - beginning sector for the entire bio **flags** - see below
BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-out-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported **rw** - binary trace for read/write request **vcnt** - bio vector count which represents number of array element (page, offset, length) which makes up this I/O request **idx** - offset into the bio vector array **phys_segments** - number of segments in this bio after physical address coalescing is performed. **size** - total size in bytes **bdev** - target block device **bdev_contains** - points to the device object which contains the partition (when bio structure represents a partition) **p_start_sect** - points to the start sector of the partition structure of the device

Context

The process creating a block IO request.

Name

probe::ioblock_trace.request — Fires just as a generic block I/O request is created for a bio.

Synopsis

```
ioblock_trace.request
```

Values

None

Description

name - name of the probe point **q** - request queue on which this bio was queued. **devname** - block device name **ino** - i-node number of the mapped file **bytes_done** - number of bytes transferred **sector** - beginning sector for the entire bio **flags** - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-of-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

rw - binary trace for read/write request **vcnt** - bio vector count which represents number of array element (page, offset, length) which make up this I/O request **idx** - offset into the bio vector array **phys_segments** - number of segments in this bio after physical address coalescing is performed. **size** - total size in bytes **bdev** - target block device **bdev_contains** - points to the device object which contains the partition (when bio structure represents a partition) **p_start_sect** - points to the start sector of the partition structure of the device

Context

The process makes block I/O request

Name

probe::ioblock_trace.end — Fires whenever a block I/O transfer is complete.

Synopsis

```
ioblock_trace.end
```

Values

None

Description

name - name of the probe point **q** - request queue on which this bio was queued. **devname** - block device name **ino** - i-node number of the mapped file **bytes_done** - number of bytes transferred **sector** - beginning sector for the entire bio **flags** - see below BIO_UPTODATE 0 ok after I/O completion BIO_RW_BLOCK 1 RW_AHEAD set, and read/write would block BIO_EOF 2 out-of-bounds error BIO_SEG_VALID 3 nr_hw_seg valid BIO_CLONED 4 doesn't own data BIO_BOUNCED 5 bio is a bounce bio BIO_USER_MAPPED 6 contains user pages BIO_EOPNOTSUPP 7 not supported

rw - binary trace for read/write request **vcnt** - bio vector count which represents number of array element (page, offset, length) which makes up this I/O request **idx** - offset into the bio vector array **phys_segments** - number of segments in this bio after physical address coalescing is performed. **size** - total size in bytes **bdev** - target block device **bdev_contains** - points to the device object which contains the partition (when bio structure represents a partition) **p_start_sect** - points to the start sector of the partition structure of the device

Context

The process signals the transfer is done.

[Report a bug](#)

Chapter 9. SCSI Tapset

This family of probe points is used to probe SCSI activities. It contains the following probe points:

Name

probe::scsi.ioentry — Prepares a SCSI mid-layer request

Synopsis

```
scsi.ioentry
```

Values

disk_major

The major number of the disk (-1 if no information)

device_state_str

The current state of the device, as a string

device_state

The current state of the device

req_addr

The current struct request pointer, as a number

disk_minor

The minor number of the disk (-1 if no information)

Name

probe::scsi.iodispatching — SCSI mid-layer dispatched low-level SCSI command

Synopsis

```
scsi.iodispatching
```

Values

device_state_str

The current state of the device, as a string

dev_id

The scsi device id

channel

The channel number

data_direction

The data_direction specifies whether this command is from/to the device 0 (DMA_BIDIRECTIONAL), 1 (DMA_TO_DEVICE), 2 (DMA_FROM_DEVICE), 3 (DMA_NONE)

lun

The lun number

request_bufflen

The request buffer length

host_no

The host number

device_state

The current state of the device

data_direction_str

Data direction, as a string

req_addr

The current struct request pointer, as a number

request_buffer

The request buffer address

Name

probe::scsi.iodone — SCSI command completed by low level driver and enqueued into the done queue.

Synopsis

```
scsi.iodone
```

Values

device_state_str

The current state of the device, as a string

dev_id

The scsi device id

channel

The channel number

data_direction

The data_direction specifies whether this command is from/to the device.

lun

The lun number

host_no

The host number

data_direction_str

Data direction, as a string

device_state

The current state of the device

scsi_timer_pending

1 if a timer is pending on this request

req_addr

The current struct request pointer, as a number

Name

probe::scsi.iocompleted — SCSI mid-layer running the completion processing for block device I/O requests

Synopsis

```
scsi.iocompleted
```

Values

device_state_str

The current state of the device, as a string

dev_id

The scsi device id

channel

The channel number

data_direction

The data_direction specifies whether this command is from/to the device

lun

The lun number

host_no

The host number

data_direction_str

Data direction, as a string

device_state

The current state of the device

req_addr

The current struct request pointer, as a number

goodbytes

The bytes completed

Name

probe::scsi.ioexecute — Create mid-layer SCSI request and wait for the result

Synopsis

```
scsi.ioexecute
```

Values

retries

Number of times to retry request

device_state_str

The current state of the device, as a string

dev_id

The scsi device id

channel

The channel number

data_direction

The data_direction specifies whether this command is from/to the device.

lun

The lun number

timeout

Request timeout in seconds

request_bufflen

The data buffer length

host_no

The host number

data_direction_str

Data direction, as a string

device_state

The current state of the device

request_buffer

The data buffer address

Name

name

probe::scsi.set_state — Order SCSI device state change

Synopsis

```
scsi.set_state
```

Values

state_str

The new state of the device, as a string

dev_id

The scsi device id

channel

The channel number

state

The new state of the device

old_state_str

The current state of the device, as a string

lun

The lun number

old_state

The current state of the device

host_no

The host number

[Report a bug](#)

Chapter 10. TTY Tapset

This family of probe points is used to probe TTY (Teletype) activities. It contains the following probe points:

Name

probe::tty.open — Called when a tty is opened

Synopsis

```
tty.open
```

Values

inode_state

the inode state

file_name

the file name

file_mode

the file mode

file_flags

the file flags

inode_number

the inode number

inode_flags

the inode flags

Name

probe::tty.release — Called when the tty is closed

Synopsis

```
tty.release
```

Values

inode_state

the inode state

file_name

the file name

file_mode

the file mode

file_flags

the file flags

inode_number

the inode number

inode_flags

the inode flags

Name

probe::tty.resize — Called when a terminal resize happens

Synopsis

```
tty.resize
```

Values

new_ypixel

the new ypixel value

old_col

the old col value

old_xpixel

the old xpixel

old_ypixel

the old ypixel

name

the tty name

old_row

the old row value

new_row

the new row value

new_xpixel

the new xpixel value

new_col

the new col value

Name

probe::tty.ioctl — called when a ioctl is request to the tty

Synopsis

```
tty.ioctl
```

Values

cmd

the ioctl command

arg

the ioctl argument

name

the file name

the file name

Name

probe::tty.init — Called when a tty is being initialized

Synopsis

```
tty.init
```

Values

driver_name

the driver name

name

the driver .dev_name name

module

the module name

Name

probe::tty.register — Called when a tty device is registered

Synopsis

```
tty.register
```

Values

driver_name

the driver name

name

the driver `.dev_name` name

index

the tty index requested

module

the module name

Name

probe::tty.unregister — Called when a tty device is being unregistered

Synopsis

tty.unregister

Values

driver_name

the driver name

name

the driver `.dev_name` name

index

the tty index requested

module

the module name

Name

probe::tty.poll — Called when a tty device is being polled

Synopsis

`tty.poll`

Values

file_name

the tty file name

wait_key

the wait queue key

Name

probe::tty.receive — called when a tty receives a message

Synopsis

`tty.receive`

Values

driver_name

the driver name

count

The amount of characters received

name

the name of the module file

fp

The flag buffer

cp

the buffer that was received

index

The tty Index

id

the tty id

Name

probe::tty.write — write to the tty line

Synopsis

```
tty.write
```

Values

driver_name

the driver name

buffer

the buffer that will be written

file_name

the file name lreated to the tty

nr

The amount of characters

Name

probe::tty.read — called when a tty line will be read

Synopsis

```
tty.read
```

Values

driver_name

the driver name

buffer

the buffer that will receive the characters

file_name

the file name created to the tty

nr

The amount of characters to be read

[Report a bug](#)

Chapter 11. Networking Tapset

This family of probe points is used to probe the activities of the network device and protocol layers.

Name

probe::netdev.receive — Data received from network device.

Synopsis

```
netdev.receive
```

Values

protocol

Protocol of received packet.

dev_name

The name of the device. e.g: eth0, ath1.

length

The length of the receiving buffer.

Name

probe::netdev.transmit — Network device transmitting buffer

Synopsis

```
netdev.transmit
```

Values

protocol

The protocol of this packet(defined in include/linux/if_ether.h)

The protocol of this packet (defined in `include/linux/if_ether.h`).

dev_name

The name of the device. e.g: eth0, ath1.

length

The length of the transmit buffer.

truesize

The size of the data to be transmitted.

Name

probe::netdev.change_mtu — Called when the netdev MTU is changed

Synopsis

```
netdev.change_mtu
```

Values

dev_name

The device that will have the MTU changed

new_mtu

The new MTU

old_mtu

The current MTU

Name

probe::netdev.open — Called when the device is opened

Synopsis

```
netdev.open
```

Values

dev_name

The device that is going to be opened

Name

probe::netdev.close — Called when the device is closed

Synopsis

```
netdev.close
```

Values

dev_name

The device that is going to be closed

Name

probe::netdev.hard_transmit — Called when the devices is going to TX (hard)

Synopsis

```
netdev.hard_transmit
```

Values

protocol

The protocol used in the transmission

dev_name

The device scheduled to transmit

length

The length of the transmit buffer.

truesize

The size of the data to be transmitted.

Name

probe::netdev.rx — Called when the device is going to receive a packet

Synopsis

```
netdev.rx
```

Values

protocol

The packet protocol

dev_name

The device received the packet

Name

probe::netdev.change_rx_flag — Called when the device RX flag will be changed

Synopsis

```
netdev.change_rx_flag
```

Values

dev_name

The device that will be changed

flags

The new flags

Name

probe::netdev.set_promiscuity — Called when the device enters/leaves promiscuity

Synopsis

```
netdev.set_promiscuity
```

Values***dev_name***

The device that is entering/leaving promiscuity mode

enable

If the device is entering promiscuity mode

inc

Count the number of promiscuity openers

disable

If the device is leaving promiscuity mode

Name

probe::netdev.ioctl — Called when the device suffers an IOCTL

Synopsis

```
netdev.ioctl
```

Values

cmd

The IOCTL request

arg

The IOCTL argument (usually the netdev interface)

Name

probe::netdev.register — Called when the device is registered

Synopsis

```
netdev.register
```

Values

dev_name

The device that is going to be registered

Name

probe::netdev.unregister — Called when the device is being unregistered

Synopsis

```
netdev.unregister
```

Values

dev_name

The device that is going to be unregistered

Name

probe::netdev.get_stats — Called when someone asks the device statistics

Synopsis

```
netdev.get_stats
```

Values

dev_name

The device that is going to provide the statistics

Name

probe::netdev.change_mac — Called when the netdev_name has the MAC changed

Synopsis

```
netdev.change_mac
```

Values

dev_name

The device that will have the MTU changed

new_mac

The new MAC address

mac_len

The MAC length

old_mac

The current MAC address

Name

probe::tcp.sendmsg — Sending a tcp message

Synopsis

```
tcp.sendmsg
```

Values

name

Name of this probe

size

Number of bytes to send

sock

Network socket

Context

The process which sends a tcp message

Name

probe::tcp.sendmsg.return — Sending TCP message is done

Synopsis

```
tcp.sendmsg.return
```


Values

name

Name of this probe

size

Number of bytes sent or error code if an error occurred.

Context

The process which sends a tcp message

Name

probe::tcp.recvmsg — Receiving TCP message

Synopsis

```
tcp.recvmsg
```

Values

saddr

A string representing the source IP address

daddr

A string representing the destination IP address

name

Name of this probe

sport

TCP source port

dport

TCP destination port

size

Number of bytes to be received

sock

Network socket

Context

The process which receives a tcp message

Name

probe::tcp.recvmsg.return — Receiving TCP message complete

Synopsis

```
tcp.recvmsg.return
```

Values

saddr

A string representing the source IP address

daddr

A string representing the destination IP address

name

Name of this probe

sport

TCP source port

dport

TCP destination port

size

Number of bytes received or error code if an error occurred.

Context

The process which receives a tcp message

Name

probe::tcp.disconnect — TCP socket disconnection

Synopsis

```
tcp.disconnect
```

Values

saddr

A string representing the source IP address

daddr

A string representing the destination IP address

flags

TCP flags (e.g. FIN, etc)

name

Name of this probe

sport

TCP source port

dport

TCP destination port

sock

Network socket

Context

The process which disconnects tcp

Name

probe::tcp.disconnect.return — TCP socket disconnection complete

Synopsis

```
tcp.disconnect.return
```

Values

ret

Error code (0: no error)

name

Name of this probe

Context

The process which disconnects tcp

Name

probe::tcp.setsockopt — Call to **setsockopt**

Synopsis

```
tcp.setsockopt
```

Values

optstr

Resolves optname to a human-readable format

level

The level at which the socket options will be manipulated

optlen

Used to access values for **setsockopt**

name

Name of this probe

optname

TCP socket options (e.g. TCP_NODELAY, TCP_MAXSEG, etc)

sock

Network socket

Context

The process which calls **setsockopt**

Name

probe::tcp.setsockopt.return — Return from **setsockopt**

Synopsis

```
tcp.setsockopt.return
```

Values

ret

Error code (0: no error)

name

Name of this probe

Context

The process which calls `setsockopt`

Name

`probe::tcp.receive` — Called when a TCP packet is received

Synopsis

```
tcp.receive
```

Values

urg

TCP URG flag

protocol

Packet protocol from driver

psh

TCP PSH flag

name

Name of the probe point

rst

TCP RST flag

dport

TCP destination port

saddr

A string representing the source IP address

daddr

A string representing the destination IP address

ack

TCP ACK flag

fin

TCP FIN flag

syn

TCP SYN flag

sport

TCP source port

iphdrIP header address

Name

probe::udp.sendmsg — Fires whenever a process sends a UDP message

Synopsis

`udp.sendmsg`

Values

name

The name of this probe

size

Number of bytes sent by the process

sock

Network socket used by the process

Context

The process which sent a UDP message

Name

probe::udp.sendmsg.return — Fires whenever an attempt to send a UDP message is completed

Synopsis

```
udp.sendmsg.return
```

Values

name

The name of this probe

size

Number of bytes sent by the process

Context

The process which sent a UDP message

Name

probe::udp.recvmsg — Fires whenever a UDP message is received

Synopsis

```
udp.recvmsg
```

Values

name

The name of this probe

size

Number of bytes received by the process

sock

Network socket used by the process

Context

The process which received a UDP message

Name

probe::udp.recvmsg.return — Fires whenever an attempt to receive a UDP message received is completed

Synopsis

```
udp.recvmsg.return
```

Values

name

The name of this probe

size

Number of bytes received by the process

Context

The process which received a UDP message

Name

probe::udp.disconnect — Fires when a process requests for a UDP disconnection

Synopsis

```
udp.disconnect
```

Values

flags

Flags (e.g. FIN, etc)

name

The name of this probe

sock

Network socket used by the process

Context

The process which requests a UDP disconnection

Name

probe::udp.disconnect.return — UDP has been disconnected successfully

Synopsis

```
udp.disconnect.return
```

Values

ret

Error code (0: no error)

name

The name of this probe

Context

The process which requested a UDP disconnection

Name

function::ip_ntop — returns a string representation from an integer IP number

Synopsis

```
function ip_ntop:string(addr:long)
```

Arguments

addr

the ip represented as an integer

[Report a bug](#)

Chapter 12. Socket Tapset

This family of probe points is used to probe socket activities. It contains the following probe points:

Name

probe::socket.send — Message sent on a socket.

Synopsis

```
socket . send
```

Values

success

Was send successful? (1 = yes, 0 = no)

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Size of message sent (in bytes) or error code if success = 0

type

Socket type value

family

Protocol family value

Context

The message sender

Name

probe::socket.receive — Message received on a socket.

Synopsis

```
socket.receive
```

Values

success

Was send successful? (1 = yes, 0 = no)

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Size of message received (in bytes) or error code if success = 0

type

Socket type value

family

Protocol family value

Context

The message receiver

Name

probe::socket.sendmsg — Message is currently being sent on a socket.

Synopsis

```
socket . sendmsg
```

Values

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Message size in bytes

type

Socket type value

family

Protocol family value

Context

The message sender

Description

Fires at the beginning of sending a message on a socket via the `sock_sendmsg` function

Name

`probe::socket.sendmsg.return` — Return from `socket.sendmsg`.

Synopsis

```
socket.sendmsg.return
```

Values

success

Was send successful? (1 = yes, 0 = no)

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Size of message sent (in bytes) or error code if success = 0

type

Socket type value

family

Protocol family value

Context

The message sender.

Description

Fires at the conclusion of sending a message on a socket via the `sock_sendmsg` function

Name

probe::socket.recvmsg — Message being received on socket

Synopsis

```
socket.recvmsg
```

Values

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Message size in bytes

type

Socket type value

family

Protocol family value

Context

The message receiver.

Description

Fires at the beginning of receiving a message on a socket via the `sock_recvmsg` function

Name

probe::socket.recvmsg.return — Return from Message being received on socket

Synopsis

```
socket.recvmsg.return
```

Values

success

Was receive successful? (1 = yes, 0 = no)

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Size of message received (in bytes) or error code if success = 0

type

Socket type value

family

Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of receiving a message on a socket via the `sock_recvmsg` function.

Name

probe::socket.aio_write — Message send via `sock_aio_write`

Synopsis

```
socket.aio_write
```

Values

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Message size in bytes

type

Socket type value

family

Protocol family value

Context

The message sender

Description

Fires at the beginning of sending a message on a socket via the `sock_aio_write` function

Name

`probe::socket.aio_write.return` — Conclusion of message send via `sock_aio_write`

Synopsis

```
socket.aio_write.return
```

Values

success

Was receive successful? (1 = yes, 0 = no)

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Size of message received (in bytes) or error code if success = 0

type

Socket type value

family

Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of sending a message on a socket via the **sock_aio_write** function

Name

probe::socket.aio_read — Receiving message via **sock_aio_read**

Synopsis

```
socket.aio_read
```

Values

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Message size in bytes

type

Socket type value

family

Protocol family value

Context

The message sender

Description

Fires at the beginning of receiving a message on a socket via the **sock_aio_read** function

Name

probe::socket.aio_read.return — Conclusion of message received via **sock_aio_read**

Synopsis

```
socket.aio_read.return
```

Values

success

Was receive successful? (1 = yes, 0 = no)

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Size of message received (in bytes) or error code if success = 0

type

Socket type value

family

Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of receiving a message on a socket via the **sock_ain_read** function

Name

probe::socket.writev — Message sent via **socket_writev**

Synopsis

```
socket.writev
```

Values

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Message size in bytes

type

Socket type value

family

Protocol family value

Context

The message sender

Description

Fires at the beginning of sending a message on a socket via the `socket_writ ev` function

Name

probe::socket.writev.return — Conclusion of message sent via `socket_writ ev`

Synopsis

```
socket.writev.return
```

Values

success

Was send successful? (1 = yes, 0 = no)

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Size of message sent (in bytes) or error code if success = 0

type

Socket type value

family

Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of sending a message on a socket via the **sock_writ**ev function

Name

probe::socket.readv — Receiving a message via **sock_read**v

Synopsis

```
socket.readv
```

Values

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Message size in bytes

type

Socket type value

family

Protocol family value

Context

The message sender

Description

Fires at the beginning of receiving a message on a socket via the **sock_readv** function

Name

probe::socket.readv.return — Conclusion of receiving a message via **sock_readv**

Synopsis

```
socket.readv.return
```

Values

success

Was receive successful? (1 = yes, 0 = no)

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

size

Size of message received (in bytes) or error code if success = 0

type

Socket type value

family

Protocol family value

Context

The message receiver.

Description

Fires at the conclusion of receiving a message on a socket via the **sock_readv** function

Name

probe::socket.create — Creation of a socket

Synopsis

```
socket.create
```

Values

protocol

Protocol value

name

Name of this probe

requester

Requested by user process or the kernel (1 = kernel, 0 = user)

type

Socket type value

family

Protocol family value

Context

The requester (see requester variable)

Description

Fires at the beginning of creating a socket.

Name

probe::socket.create.return — Return from Creation of a socket

Synopsis

```
socket.create.return
```

Values

success

Was socket creation successful? (1 = yes, 0 = no)

protocol

Protocol value

err

Error code if success == 0

name

Name of this probe

requester

Requested by user process or the kernel (1 = kernel, 0 = user)

type

Socket type value

family

Protocol family value

Context

The requester (user process or kernel)

Description

Fires at the conclusion of creating a socket.

Name

probe::socket.close — Close a socket

Synopsis

```
socket.close
```

Values

protocol

Protocol value

flags

Socket flags value

name

Name of this probe

state

Socket state value

type

Socket type value

family

Protocol family value

Context

The requester (user process or kernel)

Description

Fires at the beginning of closing a socket.

Name

probe::socket.close.return — Return from closing a socket

Synopsis

```
socket.close.return
```

Values

name

Name of this probe

Context

The requester (user process or kernel)

Description

Fires at the conclusion of closing a socket.

Name

function::sock_prot_num2str — Given a protocol number, return a string representation.

Synopsis

```
function sock_prot_num2str:string(proto:long)
```

Arguments***proto***

The protocol number.

Name

function::sock_prot_str2num — Given a protocol name (string), return the corresponding protocol number.

Synopsis

```
function sock_prot_str2num:long(proto:string)
```

Arguments

proto

The protocol name.

Name

function::sock_fam_num2str — Given a protocol family number, return a string representation.

Synopsis

```
function sock_fam_num2str:string(family:long)
```

Arguments

family

The family number.

Name

function::sock_fam_str2num — Given a protocol family name (string), return the corresponding

Synopsis

```
function sock_fam_str2num:long(family:string)
```

Arguments

family

The family name.

Description

protocol family number.

Name

function::sock_state_num2str — Given a socket state number, return a string representation.

Synopsis

```
function sock_state_num2str:string(state:long)
```

Arguments

state

The state number.

Name

function::sock_state_str2num — Given a socket state string, return the corresponding state number.

Synopsis

```
function sock_state_str2num:long(state:string)
```

Arguments

state

The state name.

[Report a bug](#)

Chapter 13. Kernel Process Tapset

This family of probe points is used to probe process-related activities. It contains the following probe points:

Name

probe::kprocess.create — Fires whenever a new process is successfully created

Synopsis

```
kprocess.create
```

Values

new_pid

The PID of the newly created process

Context

Parent of the created process.

Description

Fires whenever a new process is successfully created, either as a result of fork (or one of its syscall variants), or a new kernel thread.

Name

probe::kprocess.start — Starting new process

Synopsis

```
kprocess.start
```

Values

None

Context

Newly created process.

Description

Fires immediately before a new process begins execution.

Name

probe::kprocess.exec — Attempt to exec to a new program

Synopsis

```
kprocess.exec
```

Values

filename

The path to the new executable

Context

The caller of exec.

Description

Fires whenever a process attempts to exec to a new program.

Name

probe::kprocess.exec_complete — Return from exec to a new program

Synopsis

```
kprocess.exec_complete
```

Values

success

A boolean indicating whether the exec was successful

errno

The error number resulting from the exec

Context

On success, the context of the new executable. On failure, remains in the context of the caller.

Description

Fires at the completion of an exec call.

Name

probe::kprocess.exit — Exit from process

Synopsis

`kprocess.exit`

Values

code

The exit code of the process

Context

The process which is terminating.

Description

Fires when a process terminates. This will always be followed by a `kprocess.release`, though the latter may be delayed if the process waits in a zombie state.

Name

probe::kprocess.release — Process released

Synopsis

`kprocess.release`

Values

pid

PID of the process being released

task

A task handle to the process being released

Context

The context of the parent, if it wanted notification of this process' termination, else the context of the process itself.

Description

Fires when a process is released from the kernel. This always follows a `kprocess.exit`, though it may be delayed somewhat if the process waits in a zombie state.

[Report a bug](#)

Chapter 14. Signal Tapset

This family of probe points is used to probe signal activities. It contains the following probe points:

Name

probe::signal.send — Signal being sent to a process

Synopsis

```
signal.send
```

Values

send2queue

Indicates whether the signal is sent to an existing sigqueue

name

The name of the function used to send out the signal

task

A task handle to the signal recipient

sinfo

The address of siginfo struct

si_code

Indicates the signal type

sig_name

A string representation of the signal

sig

The number of the signal

shared

Indicates whether the signal is shared by the thread group

sig_pid

The PID of the process receiving the signal

pid_name

The name of the signal recipient

Context

The signal's sender.

Name

probe::signal.send.return — Signal being sent to a process completed

Synopsis

```
signal.send.return
```

Values

retstr

The return value to either `__group_send_sig_info`, `specific_send_sig_info`, or `send_sigqueue`

send2queue

Indicates whether the sent signal was sent to an existing sigqueue

name

The name of the function used to send out the signal

shared

Indicates whether the sent signal is shared by the thread group.

Context

The signal's sender. (correct?)

Description

Possible `__group_send_sig_info` and `specific_send_sig_info` return values are as follows;

0 -- The signal is successfully sent to a process,

which means that

(1) the signal was ignored by the receiving process, (2) this is a non-RT signal and the system already has one queued, and (3) the signal was successfully added to the sigqueue of the receiving process.

-EAGAIN -- The sigqueue of the receiving process is overflowing, the signal was RT, and the signal was sent by a user using something other than `kill`.

Possible `send_group_sigqueue` and `send_sigqueue` return values are as follows;

0 -- The signal was either successfully added into the sigqueue of the receiving process, or a `SI_TIMER` entry is already queued (in which case, the overrun count will be simply incremented).

1 -- The signal was ignored by the receiving process.

-1 -- (`send_sigqueue` only) The task was marked exiting, allowing * `posix_timer_event` to redirect it to the group leader.

Name

`probe::signal.checkperm` — Check being performed on a sent signal

Synopsis

```
signal.checkperm
```

Values

name

Name of the probe point

task

A task handle to the signal recipient

sinfo

The address of the `siginfo` structure

si_code

Indicates the signal type

sig_name

A string representation of the signal

sig

The number of the signal

pid_name

Name of the process receiving the signal

sig_pid

The PID of the process receiving the signal

Name

probe::signal.checkperm.return — Check performed on a sent signal completed

Synopsis

```
signal.checkperm.return
```

Values

retstr

Return value as a string

name

Name of the probe point

Name

probe::signal.wakeup — Sleeping process being wakened for signal

Synopsis

`signal.wakeup`

Values

resume

Indicates whether to wake up a task in a STOPPED or TRACED state

state_mask

A string representation indicating the mask of task states to wake. Possible values are TASK_INTERRUPTIBLE, TASK_STOPPED, TASK_TRACED, and TASK_INTERRUPTIBLE.

pid_name

Name of the process to wake

sig_pid

The PID of the process to wake

Name

probe::signal.check_ignored — Checking to see signal is ignored

Synopsis

`signal.check_ignored`

Values

sig_name

A string representation of the signal

sig

The number of the signal

pid_name

Name of the process receiving the signal

sig_pid

The PID of the process receiving the signal

Name

probe::signal.check_ignored.return — Check to see signal is ignored completed

Synopsis

```
signal.check_ignored.return
```

Values

retstr

Return value as a string

name

Name of the probe point

Name

probe::signal.force_segv — Forcing send of SIGSEGV

Synopsis

```
signal.force_segv
```

Values

name

Name of the probe point

sig_name

A string representation of the signal

sig

The number of the signal

pid_name

Name of the process receiving the signal

sig_pid

The PID of the process receiving the signal

Name

probe::signal.force_segv.return — Forcing send of SIGSEGV complete

Synopsis

```
signal.force_segv.return
```

Values

retstr

Return value as a string

name

Name of the probe point

Name

probe::signal.syskill — Sending kill signal to a process

Synopsis

```
signal.syskill
```

Values

name

Name of the probe point

sig_name

A string representation of the signal

sig

The specific signal sent to the process

pid_name

The name of the signal recipient

sig_pid

The PID of the process receiving the signal

Name

probe::signal.syskill.return — Sending kill signal completed

Synopsis

```
signal.syskill.return
```

Values

None

Name

probe::signal.sys_tkill — Sending a kill signal to a thread

Synopsis

```
signal.sys_tkill
```

Values

name

Name of the probe point

sig_name

A string representation of the signal

sig

The specific signal sent to the process

pid_name

The name of the signal recipient

sig_pid

The PID of the process receiving the kill signal

Description

The `tkill` call is analogous to `kill(2)`, except that it also allows a process within a specific thread group to be targeted. Such processes are targeted through their unique thread IDs (TID).

Name

`probe::signal.systkill.return` — Sending kill signal to a thread completed

Synopsis

```
signal.systkill.return
```

Values

retstr

The return value to either `__group_send_sig_info`,

name

Name of the probe point

Name

probe::signal.sys_tgkill — Sending kill signal to a thread group

Synopsis

```
signal.sys_tgkill
```

Values

name

Name of the probe point

sig_name

A string representation of the signal

sig

The specific kill signal sent to the process

tgid

The thread group ID of the thread receiving the kill signal

pid_name

The name of the signal recipient

sig_pid

The PID of the thread receiving the kill signal

Description

The tgkill call is similar to tkill, except that it also allows the caller to specify the thread group ID of the thread to be signalled. This protects against TID reuse.

Name

probe::signal.sys_tgkill.return — Sending kill signal to a thread group completed

Synopsis

```
signal.sys_tgkill.return
```

Values

retstr

The return value to either `__group_send_sig_info`,

name

Name of the probe point

Name

probe::signal.send_sig_queue — Queuing a signal to a process

Synopsis

```
signal.send_sig_queue
```

Values

sigqueue_addr

The address of the signal queue

name

Name of the probe point

sig_name

A string representation of the signal

sig

The queued signal

pid_name

Name of the process to which the signal is queued

sig_pid

The PID of the process to which the signal is queued

Name

probe::signal.send_sig_queue.return — Queuing a signal to a process completed

Synopsis

```
signal.send_sig_queue.return
```

Values

retstr

Return value as a string

name

Name of the probe point

Name

probe::signal.pending — Examining pending signal

Synopsis

```
signal.pending
```

Values

name

Name of the probe point

sigset_size

The size of the user-space signal set

sigset_add

The address of the user-space signal set (`sigset_t`)

Description

This probe is used to examine a set of signals pending for delivery to a specific thread. This normally occurs when the `do_sigpending` kernel function is executed.

Name

`probe::signal.pending.return` — Examination of pending signal completed

Synopsis

```
signal.pending.return
```

Values

retstr

Return value as a string

name

Name of the probe point

Name

`probe::signal.handle` — Signal handler being invoked

Synopsis

```
signal.handle
```

Values

regs

The address of the kernel-mode stack area

sig_code

The `si_code` value of the `siginfo` signal

name

Name of the probe point

sig_mode

Indicates whether the signal was a user-mode or kernel-mode signal

sinfo

The address of the `siginfo` table

sig_name

A string representation of the signal

oldset_addr

The address of the bitmask array of blocked signals

sig

The signal number that invoked the signal handler

ka_addr

The address of the `k_sigaction` table associated with the signal

Name

`probe::signal.handle.return` — Signal handler invocation completed

Synopsis

```
signal.handle.return
```

Values

retstr

Return value as a string

return value as a string

name

Name of the probe point

Name

probe::signal.do_action — Examining or changing a signal action

Synopsis

```
signal.do_action
```

Values

sa_mask

The new mask of the signal

name

Name of the probe point

sig_name

A string representation of the signal

oldsigact_addr

The address of the old sigaction struct associated with the signal

sig

The signal to be examined/changed

sa_handler

The new handler of the signal

sigact_addr

The address of the new sigaction struct associated with the signal

Name

probe::signal.do_action.return — Examining or changing a signal action completed

Synopsis

```
signal.do_action.return
```

Values

retstr

Return value as a string

name

Name of the probe point

Name

probe::signal.procmask — Examining or changing blocked signals

Synopsis

```
signal.procmask
```

Values

how

Indicates how to change the blocked signals; possible values are SIG_BLOCK=0 (for blocking signals), SIG_UNBLOCK=1 (for unblocking signals), and SIG_SETMASK=2 for setting the signal mask.

name

Name of the probe point

oldsigset_addr

The old address of the signal set (sigset_t)

sigset

The actual value to be set for sigset_t (correct?)

sigset_addr

The address of the signal set (`sigset_t`) to be implemented

Name

`probe::signal.procmask.return` — Examining or changing blocked signals completed

Synopsis

```
signal.procmask.return
```

Values***retstr***

Return value as a string

name

Name of the probe point

Name

`probe::signal.flush` — Flushing all pending signals for a task

Synopsis

```
signal.flush
```

Values***name***

Name of the probe point

task

The task handler of the process performing the flush

pid_name

The name of the process associated with the task performing the flush

sig_pid

The PID of the process associated with the task performing the flush

[Report a bug](#)

Chapter 15. Directory-entry (dentry) Tapset

This family of functions is used to map kernel VFS directory entry pointers to file or full path names.

Name

function::d_name — get the dirent name

Synopsis

```
function d_name:string(dentry:long)
```

Arguments

dentry

Pointer to dentry.

Description

Returns the dirent name (path basename).

Name

function::reverse_path_walk — get the full dirent path

Synopsis

```
function reverse_path_walk:string(dentry:long)
```

Arguments

dentry

Pointer to dentry.

Description

Returns the path name (partial path to mount point).

Name

function::task_dentry_path — get the full dentry path

Synopsis

```
function task_dentry_path:string(task:long,dentry:long,vfsmnt:long)
```

Arguments

task

task_struct pointer.

dentry

dirent pointer.

vfsmnt

vfsmnt pointer.

Description

Returns the full dirent name (full path to the root), like the kernel d_path function.

Name

function::d_path — get the full nameidata path

Synopsis

```
function d_path:string(nd:long)
```

Arguments

nd

Pointer to nameidata.

Description

Returns the full dirent name (full path to the root), like the kernel `d_path` function.

[Report a bug](#)

Chapter 16. Logging Tapset

This family of functions is used to send simple message strings to various destinations.

Name

`function::log` — Send a line to the common trace buffer.

Synopsis

```
function log(msg:string)
```

Arguments

msg

The formatted message string.

General Syntax

```
log(msg:string)
```

Description

This function logs data. `log` sends the message immediately to `staprun` and to the bulk transport (relays) if it is being used. If the last character given is not a newline, then one is added. This function is not as efficient as `printf` and should be used only for urgent messages.

Name

`function::warn` — Send a line to the warning stream.

Synopsis

```
function warn(msg:string)
```

Argument s

msg

The formatted message string.

General Syntax

```
warn (msg:string)
```

Description

This function sends a warning message immediately to staprun. It is also sent over the bulk transport (relays) if it is being used. If the last character is not a newline, the one is added.

Name

function::exit — Start shutting down probing script.

Synopsis

```
function exit()
```

Argument s

None

General Syntax

```
exit
```

Description

This only enqueues a request to start shutting down the script. New probes will not fire (except “end” probes), but all currently running ones may complete their work.

Name

function::error — Send an error message.

Synopsis

```
function error(msg:string)
```

Arguments

msg

The formatted message string.

Description

An implicit end-of-line is added. staprun prepends the string “ERROR:”. Sending an error message aborts the currently running probe. Depending on the MAXERRORS parameter, it may trigger an **exit**.

Name

function::ftrace — Send a message to the ftrace ring-buffer.

Synopsis

```
function ftrace(msg:string)
```

Arguments

msg

The formatted message string.

Description

If the `ftrace` ring-buffer is configured & available, see `/debugfs/tracing/trace` for the message. Otherwise, the message may be quietly dropped. An implicit end-of-line is added.

[Report a bug](#)

Chapter 17. Random functions Tapset

These functions deal with random number generation.

Name

`function::randint` — Return a random number between $[0,n)$

Synopsis

```
function randint:long(n:long)
```

Arguments

n

Number past upper limit of range, not larger than $2^{*}20$.

[Report a bug](#)

Chapter 18. String and data retrieving functions Tapset

Functions to retrieve strings and other primitive types from the kernel or a user space programs based on addresses. All strings are of a maximum length given by MAXSTRINGLEN.

Name

function::kernel_string — Retrieves string from kernel memory.

Synopsis

```
function kernel_string:string(addr:long)
```

Arguments

addr

The kernel address to retrieve the string from.

General Syntax

```
kernel_string:string(addr:long)
```

Description

This function returns the null terminated C string from a given kernel memory address. Reports an error on string copy fault.

Name

function::kernel_string2 — Retrieves string from kernel memory with alternative error string.

Synopsis


```
function kernel_string2:string(addr:long, err_msg:string)
```

Arguments

addr

The kernel address to retrieve the string from.

err_msg

The error message to return when data isn't available.

General Syntax

```
kernel_string2:string(addr:long, err_msg:string)
```

Description

This function returns the null terminated C string from a given kernel memory address. Reports the given error message on string copy fault.

Name

function::kernel_string_n — Retrieves string of given length from kernel memory.

Synopsis

```
function kernel_string_n:string(addr:long, n:long)
```

Arguments

addr

The kernel address to retrieve the string from.

n

The maximum length of the string (if not null terminated).

General Syntax

```
kernel_string_n:string(addr:long, n:long)
```

Description

Returns the C string of a maximum given length from a given kernel memory address. Reports an error on string copy fault.

Name

function::kernel_long — Retrieves a long value stored in kernel memory.

Synopsis

```
function kernel_long:long(addr:long)
```

Argument s

addr

The kernel address to retrieve the long from.

General Syntax

```
kernel_long:long(addr:long)
```

Description

Returns the long value from a given kernel memory address. Reports an error when reading from the given address fails.

Name

function::kernel_int — Retrieves an int value stored in kernel memory.

Synopsis

```
function kernel_int:long(addr:long)
```

Argument s

addr

The kernel address to retrieve the int from.

Description

Returns the int value from a given kernel memory address. Reports an error when reading from the given address fails.

Name

function::kernel_short — Retrieves a short value stored in kernel memory.

Synopsis

```
function kernel_short:long(addr:long)
```

Argument s

addr

The kernel address to retrieve the short from.

General Syntax

```
kernel_short:long(addr:long)
```

Description

Returns the short value from a given kernel memory address. Reports an error when reading from the given address fails.

Name

`function::kernel_char` — Retrieves a char value stored in kernel memory.

Synopsis

```
function kernel_char:long(addr:long)
```

Arguments

addr

The kernel address to retrieve the char from.

General Syntax

```
kernel_char:long(addr:long)
```

Description

Returns the char value from a given kernel memory address. Reports an error when reading from the given address fails.

Name

`function::kernel_pointer` — Retrieves a pointer value stored in kernel memory.

Synopsis

```
function kernel_pointer:long(addr:long)
```

Arguments

addr

The kernel address to retrieve the pointer from.

General Syntax

```
kernel_pointer:long(addr:long)
```

Description

Returns the pointer value from a given kernel memory address. Reports an error when reading from the given address fails.

Name

`function::user_string` — Retrieves string from user space.

Synopsis

```
function user_string:string(addr:long)
```

Arguments

addr

The user space address to retrieve the string from.

General Syntax

user_string:string(addr:long)

Description

Returns the null terminated C string from a given user space memory address. Reports “<unknown>” on the rare cases when userspace data is not accessible.

Name

function::user_string2 — Retrieves string from user space with alternative error string.

Synopsis

```
function user_string2:string(addr:long, err_msg:string)
```

Arguments

addr

The user space address to retrieve the string from.

err_msg

The error message to return when data isn't available.

General Syntax

```
user_string2:string(addr:long, err_msg:string)
```

Description

Returns the null terminated C string from a given user space memory address. Reports the given error message on the rare cases when userspace data is not accessible.

Name

function::user_string_warn — Retrieves string from user space.

Synopsis

```
function user_string_warn:string(addr:long)
```

Arguments

addr

The user space address to retrieve the string from.

General Syntax

```
user_string_warn:string(addr:long)
```

Description

Returns the null terminated C string from a given user space memory address. Reports “<unknown>” on the rare cases when userspace data is not accessible and warns (but does not abort) about the failure.

Name

function::user_string_quoted — Retrieves and quotes string from user space.

Synopsis

```
function user_string_quoted:string(addr:long)
```

Arguments

addr

The user space address to retrieve the string from.

General Syntax

```
user_string_quoted:string(addr:long)
```

Description

Returns the null terminated C string from a given user space memory address where any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string. Reports “NULL” for address zero. Returns “<unknown>” on the rare cases when userspace data is not accessible at the given address.

Name

function::user_string_n — Retrieves string of given length from user space.

Synopsis

```
function user_string_n:string(addr:long, n:long)
```

Arguments

addr

The user space address to retrieve the string from.

n

The maximum length of the string (if not null terminated).

General Syntax

```
user_string_n:string(addr:long, n:long)
```

Description

Returns the C string of a maximum given length from a given user space address. Returns “<unknown>” on the rare cases when userspace data is not accessible at the given address.

Name

function::user_string_n2 — Retrieves string of given length from user space.

Synopsis

```
function user_string_n2:string(addr:long, n:long, err_msg:string)
```

Arguments

addr

The user space address to retrieve the string from.

n

The maximum length of the string (if not null terminated).

err_msg

The error message to return when data isn't available.

General Syntax

```
user_string_n2:string(addr:long, n:long, err_msg:string)
```

Description

Returns the C string of a maximum given length from a given user space address. Returns the given error message string on the rare cases when userspace data is not accessible at the given address.

Name

function::user_string_n_warn — Retrieves string from user space.

Synopsis

```
function user_string_n_warn:string(addr:long,n:long)
```

Arguments

addr

The user space address to retrieve the string from.

n

The maximum length of the string (if not null terminated).

General Syntax

```
user_string_n_warn:string(addr:long, n:long)
```

Description

Returns up to *n* characters of a C string from a given user space memory address. Reports “<unknown>” on the rare cases when userspace data is not accessible and warns (but does not abort) about the failure.

Name

`function::user_string_n_quoted` — Retrieves and quotes string from user space.

Synopsis

```
function user_string_n_quoted:string(addr:long,n:long)
```

Arguments

addr

The user space address to retrieve the string from.

n

The maximum length of the string (if not null terminated).

General Syntax

```
user_string_n_quoted:string(addr:long, n:long)
```

Description

Returns up to *n* characters of a C string from the given user space memory address where any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string. Reports “NULL” for address zero. Returns “<unknown>” on the rare cases when userspace data is not accessible at the given address.

Name

function::user_short — Retrieves a short value stored in user space.

Synopsis

```
function user_short:long(addr:long)
```

Arguments

addr

The user space address to retrieve the short from.

General Syntax

```
user_short:long(addr:long)
```

Description

Returns the short value from a given user space address. Returns zero when user space data is not accessible.

Name

function::user_short_warn — Retrieves a short value stored in user space.

Synopsis

```
function user_short_warn:long(addr:long)
```

Arguments

addr

The user space address to retrieve the short from.

General Syntax

```
user_short_warn:long(addr:long)
```

Description

Returns the short value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure.

Name

function::user_int — Retrieves an int value stored in user space.

Synopsis

```
function user_int:long(addr:long)
```

Arguments

addr

The user space address to retrieve the int from.

General Syntax

```
user_int:long(addr:long)
```

Description

Returns the int value from a given user space address. Returns zero when user space data is not accessible.

Name

function::user_int_warn — Retrieves an int value stored in user space.

Synopsis

```
function user_int_warn:long(addr:long)
```

Arguments

addr

The user space address to retrieve the int from.

General Syntax

```
user_ing_warn:long(addr:long)
```

Description

Returns the int value from a given user space address. Returns zero when user space and warns (but

does not abort) about the failure.

Name

function::user_long — Retrieves a long value stored in user space.

Synopsis

```
function user_long:long(addr:long)
```

Arguments

addr

The user space address to retrieve the long from.

General Syntax

```
user_long:long(addr:long)
```

Description

Returns the long value from a given user space address. Returns zero when user space data is not accessible. Note that the size of the long depends on the architecture of the current user space task (for those architectures that support both 64/32 bit compat tasks).

Name

function::user_long_warn — Retrieves a long value stored in user space.

Synopsis

```
function user_long_warn:long(addr:long)
```

Arguments

addr

The user space address to retrieve the long from.

General Syntax

```
user_long_warn:long(addr:long)
```

Description

Returns the long value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure. Note that the size of the long depends on the architecture of the current user space task (for those architectures that support both 64/32 bit compat tasks).

Name

function::user_char — Retrieves a char value stored in user space.

Synopsis

```
function user_char:long(addr:long)
```

Arguments

addr

The user space address to retrieve the char from.

General Syntax

```
user_char:long(addr:long)
```

Description

Returns the char value from a given user space address. Returns zero when user space data is not accessible.

Name

`function::user_char_warn` — Retrieves a char value stored in user space.

Synopsis

```
function user_char_warn:long(addr:long)
```

Arguments

addr

The user space address to retrieve the char from.

General Syntax

```
user_char_warn:long(addr:long)
```

Description

Returns the char value from a given user space address. Returns zero when user space and warns (but does not abort) about the failure.

[Report a bug](#)

Chapter 19. A collection of standard string functions

Functions to get the length, a substring, getting at individual characters, string searching, escaping, tokenizing, and converting strings to longs.

Name

`function::strlen` — Returns the length of a string.

Synopsis

```
function strlen:long(s:string)
```

Arguments

s

the string

General Syntax

```
strlen: long (str:string)
```

Description

This function returns the length of the string, which can be zero up to MAXSTRINGLEN.

Name

`function::substr` — Returns a substring.

Synopsis

```
function substr:string(str:string,start:long,length:long)
```

Arguments

str

The string to take a substring from

start

Starting position. 0 = start of the string.

length

Length of string to return.

General Syntax

substr:string (str:string, start:long, stop:long)

Description

Returns the substring of the up to the given length starting at the given start position and ending at given stop position.

Name

function::stringat — Returns the char at a given position in the string.

Synopsis

```
function stringat:long(str:string, pos:long)
```

Arguments

str

The string to fetch the character from.

pos

The position to get the character from. 0 = start of the string.

General Syntax

stringat:long(srt:string, pos:long)

Description

This function returns the character at a given position in the string or zero if the string doesn't have as many characters.

Name

function::isinstr — Returns whether a string is a substring of another string.

Synopsis

```
function isinstr:long(s1:string, s2:string)
```

Argument s

s1

String to search in.

s2

Substring to find.

General syntax

isinstr:long (s1:string, s2:string)

Description

This function returns 1 if string *s1* contains *s2*, otherwise zero.

Name

function::text_str — Escape any non-printable chars in a string.

Synopsis

```
function text_str:string(input:string)
```

Arguments

input

The string to escape.

General Syntax

text_str:string (input:string)

Description

This function accepts a string argument, and any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string.

Name

function::text_strn — Escape any non-printable chars in a string.

Synopsis

```
function text_strn:string(input:string, len:long, quoted:long)
```

Arguments

input

The string to escape.

len

Maximum length of string to return. 0 means MAXSTRINGLEN.

quoted

Put double quotes around the string. If input string is truncated it will have “...” after the second quote.

General Syntax

text_strn:string (input:string, len:long, quoted:long)

Description

This function accepts a string of designated length, and any ASCII characters that are not printable are replaced by the corresponding escape sequence in the returned string.

Name

function::tokenize — Return the next non-empty token in a string.

Synopsis

```
function tokenize:string(input:string, delim:string)
```

Arguments

input

String to tokenize. If NULL, returns the next non-empty token in the string passed in the previous call to **tokenize**.

delim

Token delimiter. Set of characters that delimit the tokens.

General Syntax

tokenize:string (input:string, delim:string)

Description

This function returns the next non-empty token in the given input string, where the tokens are delimited by characters in the delim string. If the input string is non-NULL, it returns the first token. If the input string is NULL, it returns the next token in the string passed in the previous call to tokenize. If no delimiter is found, the entire remaining input string is returned. It returns NULL when no more tokens are available.

Name

function::str_replace — str_replace Replaces all instances of a substring with another.

Synopsis

```
function  
str_replace:string(prnt_str:string, srch_str:string, rplc_str:string)
```

Arguments

prnt_str

The string to search and replace in.

srch_str

The substring which is used to search in prnt_str string.

rplc_str

The substring which is used to replace srch_str.

General Syntax

str_replace:string(prnt_str:string, srch_str:string, rplc_str:string)

Description

This function returns the given string with substrings replaced.

Name

function::strtol — strtol - Convert a string to a long.

Synopsis

```
function strtol:long(str:string,base:long)
```

Arguments

str

String to convert.

base

The base to use

General Syntax

```
strtol:long (str:string, base:long)
```

Description

This function converts the string representation of a number to an integer. The base parameter indicates the number base to assume for the string (eg. 16 for hex, 8 for octal, 2 for binary).

Name

function::isdigit — Checks for a digit.

Synopsis

```
function isdigit:long(str:string)
```

Arguments

str

String to check.

General Syntax

```
isdigit:long(str:string)
```

Description

Checks for a digit (0 through 9) as the first character of a string. Returns non-zero if true, and a zero if false.

[Report a bug](#)

Chapter 20. Utility functions for using ansi control chars in logs

Utility functions for logging using ansi control characters. This lets you manipulate the cursor position and character color output and attributes of log messages.

Name

`function::ansi_clear_screen` — Move cursor to top left and clear screen.

Synopsis

```
function ansi_clear_screen()
```

Arguments

None

General Syntax

`ansi_clear_screen`

Description

Sends ansi code for moving cursor to top left and then the ansi code for clearing the screen from the cursor position to the end.

Name

`function::ansi_set_color` — Set the ansi Select Graphic Rendition mode.

Synopsis

```
function ansi_set_color(fg:long)
```

Argument s

fg

Foreground color to set.

General Syntax

```
ansi_set_color(fh:long)
```

Description

Sends ansi code for Select Graphic Rendition mode for the given foreground color. Black (30), Blue (34), Green (32), Cyan (36), Red (31), Purple (35), Brown (33), Light Gray (37).

Name

function::ansi_set_color2 — Set the ansi Select Graphic Rendition mode.

Synopsis

```
function ansi_set_color2(fg:long,bg:long)
```

Argument s

fg

Foreground color to set.

bg

Background color to set.

General Syntax

```
ansi_set_color2(fg:long, bg:long)
```

Description

Sends ansi code for Select Graphic Rendition mode for the given foreground color, Black (30), Blue (34), Green (32), Cyan (36), Red (31), Purple (35), Brown (33), Light Gray (37) and the given background color, Black (40), Red (41), Green (42), Yellow (43), Blue (44), Magenta (45), Cyan (46), White (47).

Name

function::ansi_set_color3 — Set the ansi Select Graphic Rendition mode.

Synopsis

```
function ansi_set_color3(fg:long, bg:long, attr:long)
```

Arguments

fg

Foreground color to set.

bg

Background color to set.

attr

Color attribute to set.

General Syntax

```
ansi_set_color3(fg:long, bg:long, attr:long)
```

Description

Sends ansi code for Select Graphic Rendition mode for the given foreground color, Black (30), Blue (34), Green (32), Cyan (36), Red (31), Purple (35), Brown (33), Light Gray (37), the given background color, Black (40), Red (41), Green (42), Yellow (43), Blue (44), Magenta (45), Cyan (46), White (47) and the color attribute All attributes off (0), Intensity Bold (1), Underline Single (4), Blink

Slow (5), Blink Rapid (6), Image Negative (7).

Name

function::ansi_reset_color — Resets Select Graphic Rendition mode.

Synopsis

```
function ansi_reset_color()
```

Arguments

None

General Syntax

ansi_reset_color

Description

Sends ansi code to reset foreground, background and color attribute to default values.

Name

function::ansi_new_line — Move cursor to new line.

Synopsis

```
function ansi_new_line()
```

Arguments

None

General Syntax

`ansi_new_line`

Description

Sends ansi code new line.

Name

`function::ansi_cursor_move` — Move cursor to new coordinates.

Synopsis

```
function ansi_cursor_move(x:long, y:long)
```

Argument s

x

Row to move the cursor to.

y

Colomn to move the cursor to.

General Syntax

`ansi_curos_move(x:long, y:long)`

Description

Sends ansi code for positioning the cursor at row x and column y. Coordinates start at one, (1,1) is the top-left corner.

Name

function::ansi_cursor_hide — Hides the cursor.

Synopsis

```
function ansi_cursor_hide()
```

Arguments

None

General Syntax

`ansi_cusor_hide`

Description

Sends ansi code for hiding the cursor.

Name

function::ansi_cursor_save — Saves the cursor position.

Synopsis

```
function ansi_cursor_save()
```

Arguments

None

General Syntax

`ansi_cursor_save`

Description

Sends ansi code for saving the current cursor position.

Name

`function::ansi_cursor_restore` — Restores a previously saved cursor position.

Synopsis

```
function ansi_cursor_restore()
```

Argument s

None

General Syntax

`ansi_cursor_restore`

Description

Sends ansi code for restoring the current cursor position previously saved with `ansi_cursor_save`.

Name

`function::ansi_cursor_show` — Shows the cursor.

Synopsis

```
function ansi_cursor_show()
```

Arguments

None

General Syntax

```
ansi_cursor_show
```

Description

Sends ansi code for showing the cursor.

[Report a bug](#)

Appendix A. Revision History

Revision 1-2	Thu Nov 11 2015	Robert Kratky
Release for Red Hat Enterprise Linux 7.2		