1.4.2 - Basic SystemTap syntax and control structures

The output of the odds-evens example is slightly different than shown. It should be:

```
odds[0] = 1
odds[1] = 3
odds[3] = 7
odds[4] = 9
evens[4] = 8
evens[2] = 4
evens[1] = 2
evens[0] = 0
```

The last paragraph in this section should simple say "Note that all variable types are inferred, and that all locals and globals are initialized." The part about array elements that are set to 0 or NULL being treated as deleted is no longer true.

1.6 - Safety and Security

The MAXMAPENTRIES should be changed to read as follows:
MAXMAPENTRIES - Maximum number of rows in an array if the array size is not specified explicitly when declared. The default is 2048.

3.1 - Probe definitions

Minor nit... Change "stapprobes(5) manual pages" to "stapprobes(5) man pages".

3.3 - Variables

Last paragraph... Change "regardless of in which script file they are found" to "regardless of the script file in which they are found". Change "such as due to multiple probe handlers" to "such as multiple probe handlers".

3.5 - Embedded C

Mentions "see safety.tex". What does this refer to? I can't find it.

3.6 - Embedded C functions

Insert the following after the sentence that ends "...so this is an advanced technique."

"Be especially careful when dereferencing pointers. Use the kread() macro to dereference any pointers that could potentially be invalid. If you're not sure, error on the side of caution and use kread(). Also note that all SystemTap functions and probes run with interrupts disabled, thus you cannot call functions that might sleep from within embedded C."

4.1 - General systax

3rd paragraph, 2nd sentence - change "event is occurs" to "event occurs".

4.1.2 - Suffixes: .entry, .return

Remove reference to .entry. It's not supported. The absence of a suffix implies .entry.

4.2 - Built-in probe point types (DWARF probes)

2nd bullet - ends with "... as the $return context variable." Add "The entry parameters are also available, though the function may have changed their values."

Paragraph that begins "In the following probe descriptions..." the word "identifies" is misspelled.

Paragraph that begins "Some of the source-level variables..." - 2nd sentence - Change "question mark" to "dollar sign".

4.2.2 - kernel.inline

The example isn't a real example. Change it to:

```
kernel.inline("path_to_nameidata")
kernel.inline("path_to_nameidata@fs/namei.c")
```

4.4 - Timer probes

2nd paragraph, last sentence - Change "run concurrently multiple processors" to "run concurrently on multiple processors".

4th paragraph, change units of time as follows:

- (sec) should be (s or sec)
- (msec) should be (ms or msec)
- (usec) should be (us or usec)
- (nsec) should be (ns or nsec)
- (Hz) should be (hz)

4.5 - Return probes

Insert before the last sentence: "The entry parameters are also accessible in the return probe's context, though their values may have been changed by the function."

4.6.1 - begin

Change last sentence to: "All global variables must be declared prior to this point."

5.6.1 - Binary string operators

Two sections got accidentally merged into one. Heading should be "Binary numeric operators", follow by the line "* / % + - >> ...". Then insert a new heading "Binary string operators" before ". (string contenation)".

5.8.3 - Conditions based on architecture: arch

Add the following sentence to the end of the paragraph: "The currently supported architecture strings are i386, i686, x86_64, ia64, s390x and ppc64."

6.0.6 - - (delete) <name>

Remove this section.

6.0.12 - Use of plus (+) or minus (-) suffix for ascending or descending order

Remove this section

6.0.13 - goto

Remove this section.

6.0.17 - return

Change "value of the function is not reserved" to "value of the function is not returned".

7 - Associative Arrays

Change "a comma-separated list of index expressions" to "a comma-separated list of up to five index expressions".

7.2 - Types of values

Replace everything in this section (including examples) with the following:

Array elements may be set to a number or a string. The type must be consistent throughout the use of the array. The first assignment to the array defines the elements type. Unset array elements may be fetched and return a null value (zero or empty string) as appropriate, but they are not seen by a membership test.

7.3 - Number and types of indexes

Remove this section.

7.4 - Array capacity

Replace all text in this section with the following:

Array sizes can be specified explicitly or allowed to default to the maximum size as defined by MAXMAPENTRIES. See Section 1.6 on page 10 for details on changing MAXMAPENTRIES to suit your needs.

You explicitly specify the size of an array as follows:

```
global ARRAY[<size>]
```

If you leave off [<size>], the array is created to hold MAXMAPENTRIES number of elements.

8.3.1 through 8.3.5

Remove all occurrences of "integer". It's unnecessary.

9.1.4 - printf

There are a several differences between SystemTap's version of printf and C's version. I don't think it's sufficient to say "similar to those of C." Add the following new section after the paragraph that ends "...for type by the translator."

---- NEW SECTION ---

The formatting string can contain tags that are defined as follows:

```
%[flags][width][.precision][length]specifier
```

Where *specifier* is required and defines the type and the interpretation of the value of the corresponding argument:

specifier	**Output**	**Example**
`d` or `i`	Signed decimal	`392`
`o`	Unsigned octal	`610`
`s`	String	`sample`
`u`	Unsigned decimal	`7235`
`x`	Unsigned hexadecimal (lowercase letters)	`7fa`
`X`	Unsigned hexadecimal (uppercase letters)	`7FA`
`p`	Pointer address	`0x0000000000bc614e`
`n`	Writes a binary value that is the total length of the string written by printf. The field width specifies the number of bytes to write. Valid specifications are %n, %1n, %2n and %4n. The default is 2.	See below
`b`	Writes a binary value as text. The field width specifies the number of bytes to write. Valid specifications are %b, %1b, %2b, %4b and %8b. The default width is 4 (32-bits).	See below
`%`	A `%` followed by another `%` character will write `%` to `stdout`.	

The tag can also contain *flags*, *width*, *.precision* and *modifiers* sub-specifiers, which are optional and follow these specifications:

flags	**description**
`-`	Left-justify within the given field width. Right justification is the default (see *width* sub-specifier).
`+`	Precede the result with a plus or minus sign even for positive numbers. By default, only negative numbers are preceded with a `-` sign.
(space)	If no sign is going to be written, a blank space is inserted before the value.
`#`	Used with `o`, `x` or `X` specifiers the value is preceded with `0`, `0x` or `0X` respectively

	for non-zero values.
`0`	Left-pads the number with zeroes instead of spaces, where padding is specified (see *width* sub-specifier).

width	**description**
(number)	Minimum number of characters to be printed. If the value to be printed is shorter than this number, the result is padded with blank spaces. The value is not truncated even if the result is larger.

.precision	**description**
.number	For integer specifiers (`d`, `i`, `o`, `u`, `x`, `X`): *precision* specifies the minimum number of digits to be written. If the value to be written is shorter than this number, the result is padded with leading zeros. The value is not truncated even if the result is longer. A *precision* of `0` means that no character is written for the value `0`. For `s`: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered.. When no *precision* is specified, the default is `1`. If the period is specified without an explicit value for *precision*, `0` is assumed.

--- END NEW SECTION ---

I think the binary write example prints too much. Replace it with this:

```
# stap -e ' probe begin { for (i = 97; i < 110; i++)
> printf("%3d: %1b%1b%1b\n", i, i, i-32, i-64);
> exit()}'
```

This prints:

```
 97: aA!
 98: bB"
 99: cC#
100: dD$
101: eE%
102: fF&
103: gG'
104: hH(
105: iI)
106: jJ*
107: kK+
108: lL,
109: mM-
```

The other examples are fine as is.

9.2 - Task context at the probe point

Append the following sentence: "Note that these may not return correct values when a probe is hit in interrupt context."

9.2.5 - egid

Change "GID" to "group ID".

9.2.6 - euid

Change "UID" to "user ID".

9.2.5 - gid

Change "GID" to "group ID".

9.2.19 - stp_pid

Change "of the stapd daemon" to "of the staprun process".

9.2.22 - task_current

Change "returns the task_struct" to "returns the address of the task_struct". Append the following sentence: "This address can be passed to the various task_*() functions to extract more task-specific data."

9.2.23 - task_egid

Change "GID" to "group ID".

9.2.25 - task_euid

Change "UID" to "user ID".

9.2.26 - task_gid

Change "GID" to "group ID".

9.2.28- task_parent

Change "returns the task_struct" to "returns the address of the task_struct". Append the following sentence: "This address can be passed to the various task_*() functions to extract more task-specific data."

9.2.33 - task_uid

Change "UID" to "user ID".

9.2.35 - uid

Change "UID" to "user ID" and "process" to "task".

9.3.2 - user_string

Change "this function returns <unknown>" to "this function returns the string "<unknown>"".

9.5.1 - qsq_blocked

Change last sentence to:

This function returns the fraction of elapsed time during which one or more requests were on the wait queue.

9.5.2 - qsq_print

Replace last line with the following:
This function prints a line containing the following statistics for the given queue:

- queue name
- average rate of requests per second
- average wait queue length
- average time on the wait queue
- average time to service a request
- percentage of time the wait queue was used
- percentage of time any request was being serviced

9.5.3 - qsq_service_time

Change last sentence to:
This function returns the average time in microseconds required to service a request once it's removed from the wait queue.

9.9.5 - qsq_throughput

Change last sentence to:
This function returns the average number of requests served per microsecond.

9.5.6 - qsq_utilization

Change last sentence to:
This function returns the average time in microseconds that at least one request was being serviced.

9.5.7 - qsq_wait_queue_length

Change "of a wait queue" to "of the wait queue".

9.5.8 - qsq_wait_time

Change last sentence to:
This function returns the average time in microseconds that it took for a request to be serviced (qs_wait() to qs_done()).

9.5.9 - A queue example

After the first sentence, add the following:
It uses the randomize feature of the timer probe to simulate queuing activity.

9.6.2 - probefunc

Change last sentence to:
This function returns the name of the function being probed.

9.6.3 - probemod

Change last sentence to:
This function returns the name of the module containing the probe point.

9.7.1 - ctime

Append to the bottom:
This function does not adjust for timezones. The returned time is always in GMT. Your script must manually adjust epochsecs before passing it to ctime() if you want to print local time.

9.7.4 - thread_indent

Append the following example:

The following example uses thread_indent() to trace the functions called in the drivers/usb/core kernel source. It prints a relative timestamp and the current processes' name and ID, followed by the appropriate indent and the function name. Note that "swapper(0)" indicates the kernel is running in interrupt context and there is no valid current process.

```
probe kernel.function("*@drivers/usb/core/*") {
       printf ("%s -> %s\n", thread_indent(1), probefunc())
}

probe kernel.function("*@drivers/usb/core/*").return {
       printf ("%s <- %s\n", thread_indent(-1), probefunc())
}
```

The output looks like this:

```
    0 swapper(0): -> usb_hcd_irq
    8 swapper(0): <- usb_hcd_irq
    0 swapper(0): -> usb_hcd_irq
   10 swapper(0):  -> usb_hcd_giveback_urb
   16 swapper(0):   -> urb_unlink
   22 swapper(0):   <- urb_unlink
   29 swapper(0):   -> usb_free_urb
   35 swapper(0):   <- usb_free_urb
   39 swapper(0):  <- usb_hcd_giveback_urb
   45 swapper(0): <- usb_hcd_irq
    0 usb-storage(1338): -> usb_submit_urb
    6 usb-storage(1338):  -> usb_hcd_submit_urb
   12 usb-storage(1338):   -> usb_get_urb
   18 usb-storage(1338):   <- usb_get_urb
   25 usb-storage(1338):  <- usb_hcd_submit_urb
   29 usb-storage(1338): <- usb_submit_urb
    0 swapper(0): -> usb_hcd_irq
    7 swapper(0): <- usb_hcd_irq
```

9.8 - String functions

Change "function" to "functions".

9.8.3 - substr

Change "character start" to "character position start". Change "character stop" to "character position stop".

Insert the following new string functions in the appropriate spots:

9.8.? strtol

General syntax:

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

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

9.8.? tokenize

General syntax:

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

This function returns the next token in the given input string. The tokens are assumed to be divided by the first character 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. It returns NULL when no more tokens are available.

10 For further reference

Replace the sentence that starts "From an unpacked source..." with the following:
From an unpacked source tarball, or CVS directory, the examples in `src/examples`, the tapsets in `src/tapset`, and the test scripts in `src/testsuite`.