Using Servicelog

last modified: 23 April 2007
author: Michael Strosaker <strosake@austin.ibm.com>

Servicelog is a database intended to store log entries relevant to system serviceability, such as:

- indications of serviceable events, including device failures that require the failing device to be replaced.
- informational entries relevant to system service (for example, non-critical failures that may become critical when a threshold is reached).
- indications that repair actions have taken place, such as part replacement (via PCI hotplug, for example) or other service procedures (such as migration of a partition to another physical system).
- notifications of the availability of dump data, which service personnel can use to further debug problems.

Servicelog also provides the ability for users to register tools to be notified when events matching certain criteria are logged. These notification tools will be automatically executed when an appropriate event is logged, and servicelog will provide information about the new event to the newly-executed notification tool.

Access to the servicelog database is controlled via an API; applications should link with libservicelog in order to create new entries in the log or to query the log for existing data. Command-line utilities are provided to perform many common functions; these commands are documented in the appropriate sections in this document.

The database itself resides in `/etc/servicelog`. Deleting the contents of that entire directory will empty out the database in the most brute-force fashion; see the “Servicelog Management” section below for a less invasive method for cleaning out the database (for use on test systems, for example).

Log Entry Format

Each new log entry is assigned a unique number by servicelog when it is logged. This number is referred to as the “servicelog ID” in this document. This ID number, along with other basic information concerning the log entry, is stored in the header data of the entry. This header is an `sl_header` struct, which is defined in `/usr/include/libservicelog.h` (along with other structs and #defines used in servicelog):

```
struct sl_header {
        struct sl_header *next;
        uint32_t        db_key;         /**< db entry key */
        uint32_t        event_type;     /**< event type */
        uint32_t        version;        /**< version of the event_type */
        uint32_t        event_length;   /**< total event length */
        time_t          time_event;     /**< timestamp of event occurence */
        time_t          time_log;       /**< timestamp of event logging */
        uint32_t        severity;       /**< int field of event severity */

        uint32_t        repair_action:1;
        uint32_t        serviceable_event:1;
        uint32_t        event_repaired:1;
        uint32_t        /* reserved */ :29;
};
```

The `event_type` is one of the `SL_TYPE_*` #defines, and indicates who is logging the event: `SL_TYPE_OS` indicates that the event refers to the operating system, for example, while `SL_TYPE_PPC64_RTAS` indicates that it is an event reported by RTAS on a ppc64 platform. The type indicates the format of the data that follows the header; for example, a header with a type of SL_TYPE_PPC64 is followed immediately (i.e. contiguously in memory) by an `sl_ppc64_rtas` struct.

The only exception is if the `repair_action` flag is set; that flag indicates that this log entry represents a service action taken to repair a problem that was reported earlier, and the following data is an `sl_repair` struct (to provide details about the repair action). When a repair action is logged, servicelog will automatically go back to events that were logged earlier and determine if the parts replaced (or the procedures performed) in the newly logged repair action fixes any earlier serviceable events. If so, the `event_repaired` flag in the earlier serviceable events will be set to indicate that no further repair action needs to be taken to fix that problem.

Viewing the Contents of Servicelog

The fastest and most common way of viewing the contents of the servicelog database is via the `/usr/bin/servicelog` command. The servicelog command requires either the `--id` or `--type` option. For example, to display the event with the unique servicelog ID of 20:

```
# servicelog --id=20
PPC64 Platform Event:
Servicelog ID:      20
Event Timestamp:    Thu Apr 12 13:37:02 2007
Log Timestamp:      Thu Apr 12 13:37:08 2007
Severity:           4 (WARNING)
Version:            2
Serviceable Event:  Yes
Event Repaired:     No
```

By default, only the basic, header information will be printed. To view more verbose details about the event, use the `-v` (verbose) or `-vv` (very verbose) options:

```
 # servicelog --id=20 -v
PPC64 Platform Event:
Servicelog ID:      20
Event Timestamp:    Thu Apr 12 13:37:02 2007
```

```
Log Timestamp:      Thu Apr 12 13:37:08 2007
Severity:           4 (WARNING)
Version:            2
Serviceable Event:  Yes
Event Repaired:     No
Reference Code:     B125E500
Action Flags:       a800
Event Type:         224 - Platform Event
Kernel ID:          1000
Platform ID:        50929493
Creator ID:         E - Service Processor
Subsystem ID:       25 - Memory subsystem including external cache
RTAS Severity:      41 - Unrecoverable Error, bypassed with degraded
performance
Event Subtype:      00 - Not applicable
Machine Type/Model: 9118-575
Machine Serial:     0SQIH47

Extended Reference Codes:
2: 030000f0  3: 28f00110  4: c13920ff  5: c1000000
6: 00811630  7: 00000001  8: 00d6000d  9: 00000000

Description:
Memory subsystem including external cache Informational (non-error) Event.
Refer to the system service documentation for more information.

<< Callout 1 >>
Priority            M
Type                16
Repair Event Key:   0
Procedure Id:       n/a
Location:           U787D.001.0481682-P2
FRU:                80P4180
Serial:             YH3016129997
CCIN:               260D

==== Raw RTAS Event Begin ========================================
0x0000:  064400e0 0000050c de008e00 00000000     [.D.............]
0x0010:  00000000 49424d00 50480030 0100dd00     [....IBM.PH.0....]
0x0020:  20041210 21370500 20041210 21370806     [ ...!7.. ...!7..]
...
```

The type flag may be used when the specific event ID is not known. For example, to view all ppc64_rtas events:

```
# servicelog --type=ppc64_rtas -v | less
```

The word “all” may also be used as a type; so, to view all events:

```
# servicelog --type=all -v | less
```

A few other options are provided in order to narrow the query results when the `--type` option is used:

`--start_time=<time>`
: Do not include any events that occurred before `<time>` in the results. The time must be specified in seconds since Epoch.

`--end_time=<time>`
: Do not include any events that occurred after `<time>` in the results. The time must be

specified in seconds since Epoch.

`--repair_action={yes|no|all}`
: Indicates whether to include repair actions in the results. `yes` indicates that only repair actions should be included; `no` indicates that everything except repair actions should be included; `all` indicates that both repair actions and non-repair actions should be included (default).

`--serviceable={yes|no|all}`
: Indicates whether serviceable events should be included in the results. `yes` indicates that only serviceable events should be included; `no` indicates that only non-serviceable events (i.e. informational events) should be included; `all` indicates that both serviceable events and informational events should be included (default).

`--event_repaired={yes|no|all}`
: Indicates whether repaired events should be included in the results. `yes` indicates that only repaired events should be included; `no` indicates that only non-repaired events should be included; `all` indicates that both repaired and non-repaired events should be included (default).

`--severity=<sev>`
: Indicates that only events with a severity of `<sev>` or greater should be displayed.

Applications may also link with libservicelog and call `servicelog_open()` followed by `servicelog_query()` and/or `servicelog_get_event()` to retrieve events from servicelog programmatically. The application should call `servicelog_close()` when it is finished using servicelog. All of these calls are defined in `/usr/include/libservicelog.h`.

Registering Notification Tools

Servicelog provides a tool called `/usr/bin/servicelog_notify` to register, modify, remove, or query notification tools. One of the `--add`, `--modify`, `--remove`, or `--query` flags is required to indicate which action is to be taken.

When adding a new notification tool, the `--command="<cmd>"` option must be used to indicate the path/filename of the new executable to be run. Other options can be included to indicate which events should trigger the notification tool to be run:

`--type="<type>"`
: Indicates which event types should trigger this tool to be started. `"all"` is a valid type; multiple types should be specified by multiple occurrences of the `--type` option.

`--severity="<sev>"`
: Indicates the minimum severity; anything less severe will not cause the tool to be started.

`--repair_action={yes|no|all}`
: Indicates whether to start the notification tool when repair actions are logged. `yes` indicates that only repair actions should trigger the tool; `no` indicates that everything except repair actions should trigger the tool; `all` indicates that both repair actions and non-repair actions should trigger the tool (default).

`--serviceable={yes|no|all}`
: Indicates whether to start the notification tool when serviceable events are logged. `yes` indicates that only serviceable events should trigger the tool; `no` indicates that only non-serviceable events (i.e. informational events) should trigger the tool; `all` indicates that both serviceable events and informational events should trigger the tool (default).

`--method={num_stdin|num_arg|text_stdin|pairs_stdin}`
: Indicates how the notification tool should be passed information about the newly logged event.

The argument to the `--method` option should be one of the following values:

`pairs_stdin`
: This will cause servicelog to pass all of the field data in the newly-logged entry to stdin of the notification tool as a long string of "`<field>: <value>`" pairs, each separated by a newline. Note that the Description field may contain newline characters; those characters will be replaced with '`|`' (vertical bar) characters to ensure that each parameter/value pair fits on a single line. Refer for Appendix A for details on how to retrieve samples of this data.

`num_stdin`
: This will cause the servicelog ID of the new entry to be passed to the notification tool via stdin. The notification tool will be responsible for looking up event details via libservicelog calls if it requires more details.

`num_arg`
: This will cause the servicelog ID of the new entry to be passed to the notification tool at the end of the command line (as the last command-line parameter). The notification tool will be responsible for looking up event details via libservicelog calls if it requires more details.

`text_stdin`
: This will cause the verbose (human-readable) event details to be passed to stdin of the notification tool. This text isn't particularly suitable for machine processing, but it could be used verbatim in an e-mail message sent by the notification tool, for example, or it could be written to a human-readable log file.

Refer to Appendix B for details on how to automatically register a notification tool when an RPM is installed.

The `--modify` option can be used to modify the current registration of a tool. The `--id` flag must be specified to indicate the servicelog ID of the tool to be modified. Everything about the tool will remain unchanged except the options that are modified on the `servicelog_notify --modify` command line; for example, if the `--command` flag is not specified, the currently-registered path/filename of the notification tool will not be modified by `servicelog_notify`.

A registered notification tool may be removed using the `--remove flag`; the tool to be removed can be specified either by servicelog ID (using the `--id` option) or by a portion of the command path/filename (using the `--command` option).

Servicelog can be queried for existing registered notification tools using the `--query` option to `servicelog_notify`. The tool can be specified either by servicelog ID (using the `--id` option) or by a portion of the command path/filename (using the `--command` option). The servicelog_notify tool will have an exit status of 0 if a match is found, or non-zero if it is not; this behavior is useful for scripts that do not wish to re-register a notification tool if one is already registered (see Appendix B for an example of this behavior in a .spec file).

Many users will find that the following command is the easiest way to list the currently-registered notification tools:

```
# servicelog_manage --dump notify
```

Adding Events to Servicelog

New events are added to servicelog by linking a logging application to libservicelog. The logging application should open the servicelog by invoking `servicelog_open()`; the `servicelog` struct passed to that routine will be populated with the appropriate data for the system servicelog. The logging application should then create and populate an appropriate `sl_*` struct for the event to be logged, and then pass that struct along with the `servicelog` struct to `servicelog_log_event()`. When all events have been logged, the servicelog should be closed by invoking `servicelog_close()`. All of the routines in the API are prototyped in `/usr/include/libservicelog.h`.

Most API calls in libservicelog return 0 on success, or a non-zero value on error. The string associated with the most recent error can be retrieved using the `servicelog_error()` routine.

Servicelog Management

The servicelog RPM also installs the `/usr/bin/servicelog_manage` command.

The `--status` option lists the number of entries currently logged in the servicelog database:

```
# servicelog_manage --status
Logged events:                                23
    unrepaired serviceable events:             1
    repaired serviceable events:              11
    informational events:                      2
    repair actions:                            9
Registered notification tools:                 2
```

The `--clean` option can be used to clear out the database of already-repaired serviceable events, as well as serviceable, informational, or repair events older than a specified time. This is useful for cron jobs which may be run daily, weekly, or monthly to automatically trim old events from the database. The default cutoff age is 60 days, but the age can be specified with the `--age` option:

```
# servicelog_manage --clean --age=45
```

All of the registered notification tools can be listed with:

```
# servicelog_manage --dump notify
```

All of the logged events can be viewed (verbosely) with:

```
# servicelog_manage --dump events
```

The `--truncate` option provides the ability to remove all logged events or all registered notification tools from the servicelog database. **NOTE:** the `--truncate` option should only be used on test systems or in otherwise exceptional circumstances.

```
# servicelog_manage --truncate notify
```

```
# servicelog_manage --truncate events
```

Appendix A: Listing Field/Value Pairs for Notification Tools

Create a file called /tmp/test_notify.pl with the following contents:

```
#! /usr/bin/perl

open(OUTFILE, ">> /root/test_notify.out");

while (<STDIN>) {
        print OUTFILE $_;
}
print OUTFILE "\n";

close OUTFILE;
```

Run `chmod +x /tmp/test_notify.pl` to ensure that the tool is executable. Register this new notification tool with the following command:

```
# servicelog_notify --add --type=all --command="/tmp/test_notify.pl" --method=pairs_stdin
```

Now begin to inject log entries into servicelog. Each time a new log entry is created, the field/value pairs will be written to the end of the `/tmp/test_notify.out` file.

The following is a sample of field/value pairs for a serviceable ppc64_rtas event:

```
ServicelogID: 23
EventType: PPC64 Platform Event
Version: 2
RepairAction: 0
Serviceable: 1
Repaired: 0
EventTime: 04/23/2007 14:18:01
LogTime: 04/23/2007 14:18:20
Severity: 4
Refcode: 10009133
AddlWord0: 0x00000040
AddlWord1: 0x1000021a
AddlWord2: 0x0000fd33
AddlWord3: 0x00000000
AddlWord4: 0x20000001
AddlWord5: 0x00000000
AddlWord6: 0x00000000
AddlWord7: 0x00000000
ActionFlags: 0xa000
EventType: 224
KernelID: 1000
PlatformID: 0x820006c3
CreatorID: H
SubsystemID: 0x60
EventSubtype: 0x00
RTASSeverity: 0x40
MachineType: 9123-300
MachineSerial: 0458891
Description: Power/Cooling subsystem Unrecovered Error, general. Refer to the system service
documentation for more information.
Callout: M 192 n/a U787E.001.0458891 SYSBKPL n/a n/a 0
```

Appendix B: Registering Notification Tools During RPM Installation

To register notification tools when an RPM is installed, include a “%post” section in the .spec file. A “%preun” section should also be included to unregister the tool if the RPM is uninstalled. To register a tool at `/etc/foo_notify` to be run with the option “`--bar=arg`”:

```
%post
# Post-install script ------------------------------------------------------

# register foo_notify as a notification tool
/usr/bin/servicelog_notify --query --command="/etc/foo_notify" >/dev/null 2>&1
if [[ $? == 1 ]]; then
    /usr/bin/servicelog_notify --add --command="/etc/foo_notify --bar=arg" --type=ppc64_rtas --
type=ppc64_encl --repair_action=no --serviceable=all --method=pairs_stdin
fi;

%preun
# Pre-unInstall script -----------------------------------------------

if [ "$1" = "0" ]; then # last uninstall
    /usr/bin/servicelog_notify --remove --command="/etc/foo_notify --bar=arg" >/dev/null 2>&1
fi
```