U.S. DEPARTMENT OF ENERGY

PNNL-22588

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

The State-of-Health Server

DES-0006

Revision 4

Charlie Hubbard

March 2013

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by **Battelle** *Since 1965*

The State-of-Health Server

Charlie Hubbard

DES-0006
Revision 4
March 2013

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under
Contract DE-AC05-76RL01830

Printed in the United States of America

Contents

1 Introduction

The state-of-health (SoH) server is a standard system server that provides SoH data logging services for the control system. Like all system servers, the SoH server is derived from the `BaseServer` class (see *cliserv.h* and *cliserv.cpp* in the code base), and it inherits the inter-process communication framework, standard server messages, etc., that `BaseServer` provides. For that reason, to achieve a detailed understanding of the SoH server and its client interface library, the reader should first be thoroughly familiar with the `BaseServer` and `BaseClient` classes, which are described in detail in design document ***DES-0005, The Client/Server Architecture***.

In the code, the implementation for the SoH server can be found in the file *sohServer.cpp*. The client interface library for the server is defined and implemented in *sohClientLib.h* and *sohClientLib.cpp*. A simple text-based menu client for the server also exists, and it is implemented in *sohMenu.cpp*.

The primary documentation for the SoH server source code is the Doxygen-generated HTML documentation associated with each of the above source files. That documentation set is automatically built based on the source code itself. It provides the most detailed, up-to-date descriptions of the code. The document you are reading now is supplemental, and is intended to provide deeper background for the server, and its client API. If contradictions between this document and the Doxygen-generated documentation are found, the Doxygen-generated documentation should be considered correct.

2 Basic Operation

All servers derived from the `BaseServer` class (which is to say *all* system servers), have the ability to return state-of-health information in a standard way (although by default, servers return empty SoH data records and must be overridden on a case-by-case basis to provide useful SoH data — see ***DES-0005, The Client/Server Architecture*** for complete details).

Because this is an attribute of the generic `BaseServer` class, the generic `BaseClient` class has the ability to query the state-of-health information offered by *any* server derived from `BaseServer`. The SoH server takes advantage of this fact by instantiating one `BaseClient` client interface for each server in the system from which the state-of-health server expects to receive SoH data[1]. The client interface references are stored in an STL map keyed on server name.

At some user-definable interval (typically 60 seconds, although this is client-settable), the SoH server wakes up, iterates through the map, and requests the latest SoH data for all servers for which the map contains a corresponding client interface. The SoH data for all

[1]The names of the actual servers that will be queried for SoH data are specified in the SoH server's configuration file

queried servers is brought together and written as one record in the current state-of-health log file. This process continues until the server is shut down or a client instructs the server to stop logging data.

3 File Channels

The SoH logger server supports the concept of *file channels*. On startup, the server allocates one or more of these file channels (the exact number to allocate is specified on the command line). Individual file channels are referred to by number, beginning with zero and increasing sequentially.

The way this works is simple. When new SoH data is acquired from the other servers on the system, copies of that data are written to every open file channel. When clients open a new log file or close one that is currently logging, they also specify the ID of the file channel the file is attached to. However, both the `Start()` and `Stop()` client API methods default the file channel ID to zero, so on systems where only a single file channel is needed, clients do not have to understand or think about the file channels, or even realize that the server supports them.

At first glance, file channels appear to do nothing but create a lot of redundant files. After all, when new data is available, that same data is written to all open file channels. To what end?

The need for file channels springs from several past and present projects in which a sample is processed and then moved into a nuclear counting cell, where it it sits for a very long time being analyzed. These systems typically support multiple nuclear counting cells. As soon as one counting cell is loaded with sample, it is possible to start processing another sample with the intent of loading it into a different counting cell. The result is that at any given time, multiple samples can be in various states of staggered but overlapping processing or analysis. Because data is typically reported on a per-sample basis, it is convenient to have a state-of-health record that covers a particular sample from initial processing to final analysis. File channels handle this situation. When a new sample begins processing, a new SoH log is opened for it on an unused file channel. When final analysis of that sample is complete, the associated file channel can be closed. Other samples that are in other states of processing/analysis continue to log SoH data on their associated file channels.

Of course, not all projects will have a situation like described above in which multiple, simultaneous (but staggered) SoH logging is desirable. Initially, support for file channels was thought to be highly project-specific, and unneeded in what is supposed to be a general-purpose SoH logger. In fact, the initial implementation of file channels was a modification of an earlier, single file SoH logger that was performed specifically to support a staggered sample project similar to the one described above. However, because support for file channels has been implemented in such a way as to be invisible to clients that do not require it, it was decided to fold the capability into the general purpose server.

4 The Server Configuration File

As mentioned previously, the server maintains an internal STL map that holds a set of client interfaces to servers. This map is keyed on server name. The names of the servers that the SoH server should query come from a text-based configuration file.

The format of the configuration file is simple. Blank lines and lines beginning with a pound sign (#) are ignored. All other lines are relevant. The remaining lines are expected to contain the names of the servers to be queried (one name per line). The server names must be the same names the servers use when they register themselves with the server name resolver (see design document ***DES-0004*, *The Server Name Resolver*** for details). In most of the code base, these names are specified by symbolic constants. These constants are defined in the file *servers.h*. Refer to that file to determine the actual text names the various servers go by.

The server reads through its configuration file on startup, creates one `BaseClient` client interface for every server listed, and inserts the interfaces into its server map. An example configuration file appears below.

```
###########################################################################
#
# This file is simply a list of the servers from which the SoH server will
# request SoH information.  The names in the list must be the names the
# respective servers register with the name resolver when they start.
#
###########################################################################

anadig
dps
pid
```

5 SoH Data File Format

In this section we describe the format of the SoH log files the server writes.

The server's log files are text format files, organized into three sections. The first section, consisting of exactly one line, is the header. This is immediately followed by a sensor information record, also consisting of exactly one line. All lines following the information record are data records. Individual records take up exactly one line, and a new data record is written each time the server writes a new set of SoH data. All lines in the SoH file are encoded as Data Serialization Protocol strings, which makes them easy to parse using the `MessageParser` class[2].

[2]For a complete description of the Data Serialization Protocol and the `MessageParser` class, see design document ***DES-0002*, *Data Serialization Protocol***.

5.1 The Header Line

Every SoH log file begins with a header line that describes something about the system that created the file. The header line format is shown below. As can be seen, this is a DSP format string, therefore it is most easily parsed with the `MessageParser` utility class.

```
format
(header,<version>,(<projectID>,<systemID>,<location>))
```

- `header` — This is the string literal "header." The sole purpose of this field is to identify the line as the header line, which may be useful to some parsers. Mostly this field is fluff.
- `<version>` — This field contains the file format version. As of this writing, version 2 is the most current. No changes are anticipated, but, if changes are made, this field provides a simple way for a viewer program to recognize different versions and parse them accordingly.
- `<projectID>` — This field identifies the *type* of system that generated the file (i.e., a DAS system, a MRVL system, etc.).
- `<systemID>` — This field identifies the *specific* system that generated the file (this could be something like a site identifier, USX01 say).
- `<location>` — This identifies the system's location. During development, this is typically a room number.

The last three items in the above list are populated from environment variables that are assumed to already be defined on the control computer at the time a new log file is started. The `<projectID>` field corresponds to the `LOG_PROJECT_ID` environment variable, `<systemID>` corresponds to `LOG_SYSTEM_ID`, and `<location>` corresponds to `LOG_LOCATION`. If one or more of these environment variables is not defined when the log file is started, the string "Unknown" is used instead.

5.2 The Info Line

The line immediately following the header line is called the *info* line. It contains information about each of the sensors that report values to the SoH file. The info line format is shown below. As can be seen, this is a DSP format string, therefore it is most easily parsed with the `MessageParser` utility class.

```
format
(info,(<name>,<unit>,<analog>),(<name>,<unit>,<analog>),...,(<name>,<unit>,<analog>))
```

The first field in the info line is always the string literal "info." This identifies the line as the information line. Following the "info" field are a series of one or more DSP sub-records that each contain information about one sensor that reports data to the file. There is one of these sub-records for every reporting sensor.

- `<name>` — This is the name of the specific sensor being described. These names are short, human-readable names (a pressure sensor named "PS101" for example) unique across a project.
- `<unit>` — In the case of analog sensors, this field contains the name of the engineering units the sensor reports in (for example, "ml/min", "PSI", "C", etc.). For digital sensors (such as valves, limit switches, etc.), the value of this field is unimportant, but typically contains something like "none," "bool," "digital," etc..
- `<analog>` — If this field is "1," the sensor is an analog sensor (so the `<unit>` field is meaningful). If the field is "0," the sensor is a digital sensor. The primary reason for this field is so viewer programs can optionally plot digital sensor values (valve states and so on) on a secondary Y axis so that their display is unaffected by vertical zooming of the primary Y axis.

5.3 Data Lines

All subsequent lines in the file are *data* lines. The format for a data line appears below. As can be seen, these are DSP format strings, so they are most easily parsed using the `MessageParser` utility class.

format
```
(data,<timestamp>,<value>,<value>,...,<value>)
```

The first field in the data line is always the string literal "data." This identifies the line as a data line.

Following the "data" field is a 10-digit timestamp that indicates the real-world time at which this entry was written to the file. This field is a standard POSIX `time_t` timestamp. It corresponds to the number of seconds that have elapsed since midnight, January 1, 1970 GMT. All POSIX compliant operating systems (Linux, QNX, etc.) have library functions that can convert this value to various human-readable forms.

Finally, there appears a series of values. These correspond to the values of the corresponding sensors in the "info" record, and they appear in the same order, and at the same index, as their "info" record counterparts.

The decision to provide only values in the "data" records, as opposed to sensor-name/value pairs,[3] has certain ramifications. Most importantly, it enormously reduces the total size of the file (perhaps by a factor of three or more, depending on the size of the sensor names and the types of data they return). But it also introduces a weakness. If one or more of the

[3]Version 1 of the file format did use name/value pairs in the "data" records, but that was reduced to just values in version 2 as a space-saving measure.

servers that are providing SoH data go off line, the SoH server will no longer receive SoH data from them. Placeholders for the missing sensors still need to be added to the output “data” record, otherwise the reported values will no longer appear in the same indices, and there will be no way to relate the values to a specific sensor in the “info” line. However, the SoH server does handle this properly. Missing sensors are replaced with empty fields (that is zero-byte fields) still separated by commas. An example may make this more clear.

Consider the following “data” record as the aggregate SoH data from three servers: A, B and C. Server A provides the values 1, 2, and 3, server B provides the values 10, 20, and 30, and Server C provides the values 100, 200, and 300.

```
(data,1320854963,1,2,3,10,20,30,100,200,300)
```

Now assume that Server B crashes or otherwise goes offline. During the period that Server B is unavailable to provide SoH data, the data records will look like so…

```
(data,1320864873,1,2,3,,,,100,200,300)
```

With empty fields used to ensure the values that are reported from working servers stay in the same columns they were in to begin with. Later, if server B restarts, SoH data from Server B will again appear in its proper place.

6 Server Start-Up

The server is started from the command line using the following syntax…

```
sohServer configFile interval [numChannels]
```

where…

- `configFile` is the path/filename of the server’s configuration file (remember, this is the file that contains the names of the servers the SoH server will query for new SoH data).
- `interval` is the startup logging interval specified in seconds. This specifies the rate at which new “data” records will be written to the output SoH data files. For example, a value of 60 means that a new “data” record will be written to each open file channel every 60 seconds. The value of the `interval` parameter can be no smaller than 4. If a value less than 4 is entered, 4 will be the actual value used. The logging interval can also be modified at run-time via the server’s client interface.

- `numChannels` The final parameter is optional. If present, it represents the number of file channels the server should make available for use. If this parameter is left off, only a single file channel will be supported.

If the server is started with no command line parameters specified, it exits with the following usage message.

```
This is the SoH logger server.  It uses the server-side SoH mechanism
whereby servers from which the SoH data is obtained, decide themselves
which values to report as SoH and which not to report.  They also
maintain and provide full parameter descriptions of each parameter they
are willing to report (name, type, and units).

This server implements a 'file channel' concept. The server supports
some number (specified on the command line) of 'file channels'.  Each
file channel can be separately opened (associated with an independent
file on disk) and closed.  SoH data records are written to all file
channels that are open when the data is queried.  This obviously leads
to a lot of duplication of data.  The intent here is to support projects
which perform staggered, overlapping runs.  That is, they can start a
new run before the current run completes (this is common in projects
that use multiple nuclear detectors for example).  With the file channel
concept, complete, contiguous SoH files can be maintained for each run,
eliminating the need to piece together complete SoH coverage from a
series of smaller SoH files.

Multiple file channels does not however make the server more difficult
to use for projects in which a single SoH file open at any given time is
sufficient.  If no 'max file channels' value is given on the command
line, then the server will only support one.  Also, the client interface
to this server (see sohClientLib.h/.cpp) defaults to using the first
file channel (channel 0) unless explicitly specified otherwise by the
user, so simple projects that use this server never need know that
multiple file channels are even an option.

Note that this server DOES NOT immediately begin logging upon startup.
Clients must explicitly issue SOH_START commands to begin logging.

Once every user-defined interval, this server queries the various
servers for SoH information and writes the returned information to all
open file channels.

Usage: sohServer configFile interval [numChannels]

       configFile - The path/name of the server's configuration file

       interval - The initial logging interval specified in seconds.

       numChannels - The maximum number of file channels the server will
                     support (i.e. the maximum number of simultaneously
                     recording SoH files that can exist).
```

7 The Test Menu Program

For development and testing purposes, a small text-mode menu client application called *sohMenu* (see *sohMenu.cpp*), has been developed for the SoH server. The program uses the

`SOHClient` class as its interface to the server, and it also maintains a client interface to the system event logger, so it can record when it starts up and when it shuts down.

The menu program is meant to be started from within a terminal window. It requires no command line arguments. When the program runs, it presents the user with the following text menu.

```
General Server Items:
 -1 - ping server
 -2 - get server statistics
 -3 - get server message response interval histogram
-99 - shutdown server

Server Specific Items:
 1 - Get current logging interval
 2 - Set a new logging interval
 3 - Start logging
 4 - Stop logging
 5 - Stop logging on ALL channels
 6 - Force a record
 7 - Get file names

0 - Exit Program

Enter Selection >
```

The first three menu items correspond to standard server messages that all servers can support[4]. The six options under the "Server Specific Items" heading correspond to the six server-specific messages implemented by the SoH server. Users choose the number of the message they want to send, and they are prompted for additional parameters as needed.

The menu program is a full client, supporting every client request message the SoH server is able to process. It allows testers to send each of those messages to the server and view the server's responses. It is intended primarily as a development, testing and debugging tool; however, experience has shown that it is also useful as a bare-bones user interface to the SoH server running on real-world systems as well.

[4]See design document ***DES-0005*, *The Client/Server Architecture*** for more information on these and other standard server messages.