Summary - Socket Write

Name	socket_write
Worker Type	Application (Testing)
Version	v1.4
Release Date	October 2018
Component Library	ocpi.assets.util_comps
Workers	socket_write.rcc
Tested Platforms	linux-c7-x86_64, linux-x13_3-arm

Functionality

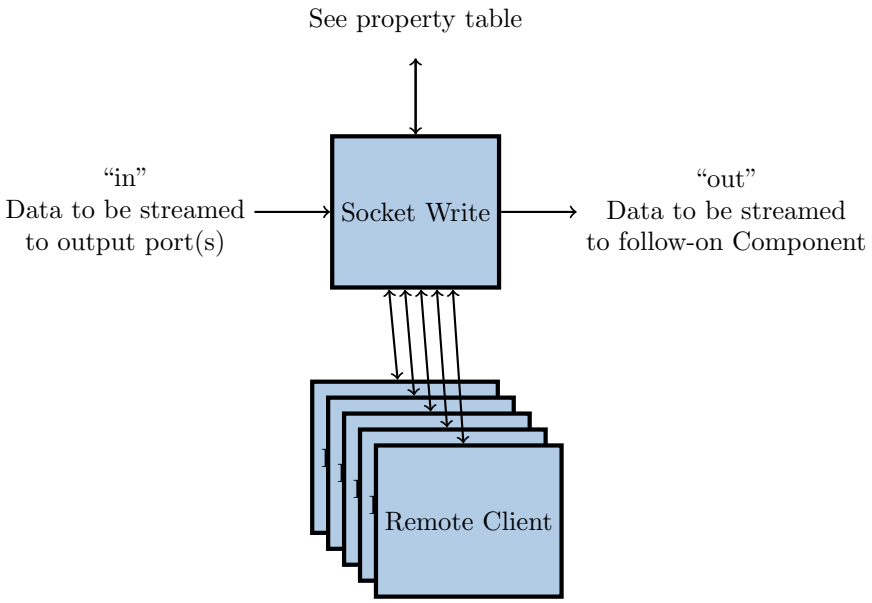
The Socket Write component forwards all incoming data to a TCP port *or* acts as a demultiplexer/router by parsing any protocol and routing different opcodes to various output ports.

The serving/listening TCP ports, along with various ways to determine the Worker's "done" status, are extremely configurable using Properties.

> This Component provides *minimal* error checking and is **not recommended for production use**, but is only intended for prototyping and testing of other Components.

Block Diagrams

Top level



Source Dependencies

socket_write.rcc

- ocpiassets/components/util_comps/socket_write.rcc/socket_write.cc
- ocpiassets/components/util_comps/socket_write.rcc/ext_src/connection.cpp
- ocpiassets/components/util_comps/socket_write.rcc/ext_src/connection.hpp
- ocpiassets/components/util_comps/socket_write.rcc/ext_src/connection_manager.cpp

- ocpiassets/components/util_comps/socket_write.rcc/ext_src/connection_manager.hpp
- ocpiassets/components/util_comps/socket_write.rcc/ext_src/outbound.hpp
- ocpiassets/components/util_comps/socket_write.rcc/ext_src/server.cpp
- ocpiassets/components/util_comps/socket_write.rcc/ext_src/server.hpp
- ocpiassets/components/util_comps/socket_write.rcc/asio/* [1]

socket_write.rcc Compilation

Because OpenCPI maintains backwards compatibility with older compilers, a fully-compliant C++11 environment is *not* required. However, the workaround for non-C++11-compliance is that the ASIO library has dependencies on the Boost[2] library, *e.g.* on CentOS 6, it requires the `boost-devel`, `boost-thread`, and `boost-static` RPMs.

To build this component targeting a non-x86 platform, the vendor must provide the appropriate `boost_system` and `boost_thread` *static* library files. The Worker's build system will attempt to find them using the `locate` command in a subdirectory that has `${OCPI_CROSS_HOST}` within the path.

See the enclosed `README` file for more information, including how to add new platforms.

[1] Externally provided ASIO library for asynchronous IO with C++, with OpenCPI-specific build system
[2] `http://www.boost.org/`

Component Spec Properties

Name	Type	SequenceLength	ArrayDimensions	Accessibility	Valid Range	Default	Usage
`outSocket`[1]	Struct	-	-	Writable, Readable	-	-	TCP socket(s) to use for listening
`outSocket.address`	String	16	-	”	-	`0.0.0.0`	Address/interface to use for port[2], *e.g.* `127.0.0.1`
`outSocket.expectedClients`	UShort	-	-	”	Standard	0	Number of clients required to be connected before `run()` method will proceed.[3]
`outSocket.port`	UShort	-	-	”	1025 - 65535	-	Output port to use if all data should remain combined [4] [5]
`outSocket.ports`	UShort	-	256	”	-	-	A list of port numbers to listen on, with `0` indicating unused [6] [7]
`outSocket .messagesInStream`	Bool	-	256	”	-	false	Write out data in “message” mode with embedded opcode
`current`	Struct	-	-	Volatile	-	-	Current statistics for each opcode
`current.Total`	Struct	-	-	”	-	-	Statistics across *all* opcodes
`current.Total.bytes`	ULongLong	-	-	”	Standard	-	Number of bytes received
`current.Total.messages`	ULongLong	-	-	”	Standard	-	Number of messages received
`current.Opcode`	Struct	-	256	”	-	-	Statistics for *each* opcode
`current.Opcode.*`	Various	-	”	-	-	-	Various[8]
`stopOn`	Struct	-	-	Writable, Readable	-	-	Condition(s) required to have Worker report completion[9]
`stopOn.Total`	Struct	-	-	”	-	-	Stops if any non-zero value is exceeded when counting *all* data received
`stopOn.Total.bytes`	ULongLong	-	-	”	Standard	0	Stop on number of bytes received
`stopOn.Total.messages`	ULongLong	-	-	”	Standard	0	Stop in number of messages received
`stopOn.Opcode`	Struct	-	256	”	-	-	Stops if any non-zero value is exceeded when counting data received using a specific opcode
`stopOn.Opcode.*`	Various	-	-	”	-	-	Various[10]
`stopOn.Any`	Struct	-	-	”	-	-	Stops if any non-zero value is exceeded when counting data received using any single opcode
`stopOn.Any.*`	Various	-	-	”	-	-	Various[10]
`stopOn.ZLM`	UShort	-	-	”	0 - 256	0	Stops if a **Z**ero **L**ength **M**essage is received using a given opcode.[11]

[1]This structure is only read at Component START to configure.
[2]The default listens on all interfaces.
[3]Probably useful only for testing and may incorrectly inhibit data flow.
[4]ICANN reserves up to 49151.
[5]Attempting to use a port that is used by another process will cause a fatal error.
[6]See “Performance and Resource Utilization.”
[7]This Property is only used when `port` is set to `0`.
[8]Internal structure equivalent to `current.Total` and not explicitly shown.
[9]Any matched condition will halt the processing.
[10]Internal structure equivalent to `stopOn.Total` and not explicitly shown.
[11]Default is opcode 0; set to invalid opcode 256 if this feature is *not* desired.

Worker Properties

socket_write.rcc

Control Operations[3] Start, Stop

[3]All TCP connections are terminated in the **Stop** state, while listening ports are opened in the **Start** state. If the Component is **Stop**ped, any clients will be *disconnected* (i.e. not paused) and must reconnect after it is **Start**ed.

Component Ports

Name	Producer	Protocol	Optional	Advanced	Usage
in	false	-	false	numberofopcodes=256	Data to be streamed to sockets(s)
out	true	-	true	numberofopcodes=256	Data pass-through

Worker Interfaces

There are no implementation-specific interfaces for this component.

Performance and Resource Utilization

socket_write.rcc

Processor Type	Processor Frequency	Run Function Time
linux-c6-x86_64 Intel(R) Xeon(R) CPU E5-1607	3.00 GHz	TBD
linux-c7-x86_64 Intel(R) Core(TM) i7-3630QM	2.40 GHz	TBD
linux-13_3-arm ARMv7 Processor rev 0 (v7l)	666 MHz	TBD

Each listening port requires system resources, such as an open file descriptor. When opening more than a handful of ports, the user may need to use `ulimit` to increase the number of open file descriptors. To do this temporarily, the command `ulimit -n 2048` can *sometimes* fix the currently running shell. Consult the documentation for your Operating System to permanently increase the limit for all processes.

TCP connections have a large overhead when compared to other transport processes, such as the OpenCPI internal messaging system. Currently, this component **does not** combine Messages to optimize the outbound connection, *e.g.* taking into account TCP Maximum Segment Size (MSS). It is *highly* recommended that users of this Component use a minimum message size of 4K or combine multiple messages in some way in an upstream Component.

Data buffers are not returned to the framework nor sent out the optional "out" port until all TCP clients have received their data. This may result in throttling or a possible denial-of-service attack if a malicious client connects but never accepts data.

Test and Verification

Usage (local/x86)

After building the component, the user needs to type `make tests RCC_CONTAINERS=1` in the *socket_write.test* directory. Various properties and data flows will be tested to try to cover as many use cases as possible.

If the user would like to execute only one test, `TESTS=test_XX` can be added to the end of the command.

Experimental: Usage (remote/ARM)

Full test environment configuration (*e.g.* NFS mounting, `OCPI_CDK_DIR`, etc.) on the remote GPP is beyond the scope of this document. The test procedures assume that both shells' current working directory is the *socket_write.test* directory (NFS-mounted on remote) and `ocpirun` is in the remote's current `PATH`. NFS **must** be used for the scripts to properly verify the outputs.

In the host shell, the user types `make tests IP=xx.xx.xx.xx`. A command that can be copied and then pasted into the remote shell will be displayed. **This command should be executed in less than a minute to ensure the test system begins listening before the host times out.** The timeout can be changed using the `LISTEN_TIMEOUT` variable. Once the remote shell returns to the bash prompt, pressing "Enter" on the host will begin the verification process.

Single tests can be performed in the same manner as documented above.

Specific Platform Note - Matchstiq-Z1

Some tests have had "Segmentation Faults" or "Alignment Errors" in certain scenarios on the Z1. The problem becomes most evident when there are multiple clients connected, but has been more rarely observed with even a single client. This seems to happen when both USB ports are used to simultaneously transmit a large amount of data, *e.g.* high log-level output to a USB serial console as well as NFS-mounted output files over a USB-to-Ethernet adapter. The default test setup avoids triggering this by limiting output that is fed to the user, but users should be aware of this issue if non-default test scenarios are attempted. If `ssh` is used to have all data routed through the USB-to-Ethernet adapter, this failure mode is avoided.

Detailed Theory of Operation

Each `test_XX` subdirectory has the following files:

- `description` - a one-line description of the test
- `application.xml` - the OAS XML for the test setup
- `portmap` - (optional) list of TCP ports paired to output files
- `localinclude.mk` - (optional) custom `Makefile` rules needed for test
- `golden.md5` - (optional) MD5 checksums of golden/expected output
- `generate.[sh|pl|py]` - (optional) script to generate test data
- `verify.sh` - (optional) script to verify output(s)

Data is sourced with a source component (often `pattern` or `file_read`) within the OAS. If the former, the source data is encapsulated in the OAS. When the latter, a `generate.py` script generates the required data. Most OASs dump the "`current`" property to a file `UUT.current.dump`, which is also confirmed to match expected output. Some tests connect a `file_write_demux` to the `out` port to verify pass-through operation.

If `generate.sh` does not exist, a default one is created that will run `generate.pl` and/or `generate.py` if they exist and are executable. This default script is removed with `make clean`.

At test launch, if a file `portmap` exists, it launches a Python-based utility script `busy_loop_socket.py`, which opens a client on a given port and repeatedly attempts to connect. Each line is a port number followed by a (relative) file path where the data is written upon successful connection.

If `verify.sh` does not exist, a default one is created that will ensure the application did not time out and then run `md5sum` to verify all the checksums listed in `golden.md5`. This default script is also removed upon `make clean`.