Summary - File Write Demux

Name	file_write_demux
Worker Type	Application
Version	v1.5
Release Date	4/2019
Component Library	ocpi.assets.util_comps
Workers	file_write_demux.rcc
Tested Platforms	centos7, xilinx13_3 (limited)

Functionality

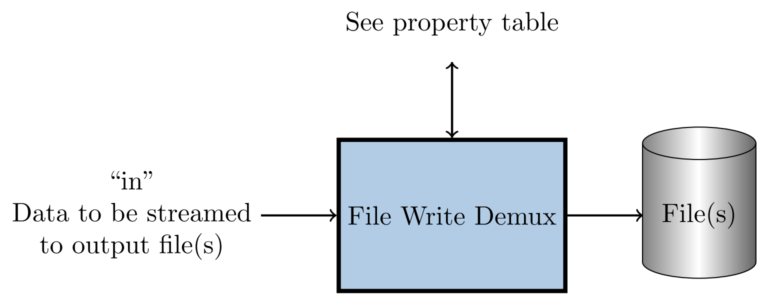
The File Writer Demux component acts as a demultiplexer/router by parsing any protocol and routing different opcodes to various output files.

The names of the files being written, 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

file_write_demux.rcc

- <assets>/components/util_comps/file_write_demux.rcc/file_write_demux.cc

Component Spec Properties

Name	Type	SequenceLength	ArrayDimensions	Accessibility	Valid Range	Default	Usage
`outFile`	Struct	-	-	Writable, Readable[1]	-	-	File name(s) to write to
`outFile.prefix`	String	1024	-	”	-	*None*	File prefix[2]
`outFile.digits`	UChar	-	-	”	1 - 3	1	Width for opcode number output padding
`outFile.suffix`	String	1024	-	”	-	.bin	File suffix[2]
`outFile.messagesInFile`	Bool	-	256	”	-	false	Write file 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[3]
`stopOn`	Struct	-	-	Writable, Readable[1]	-	-	Condition(s) required to have Worker report completion[4]
`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[5]
`stopOn.Any`	Struct	-	-	”	-	-	Stops if any non-zero value is exceeded when counting data received using any single opcode
`stopOn.Any.*`	Various	-	-	”	-	-	Various[5]
`stopOn.ZLM`	UShort	-	-	”	0 - 256	0	Stops if a **Z**ero **L**ength **M**essage is received using a given opcode.[6]

[1] “Readable” is deprecated and superfluous here. It will be removed in a future release.
[2] The output filename will use `strftime` substitutions to format the string if any `%` is found within it.
[3] Internal structure equivalent to `current.Total` and not explicitly shown.
[4] *Any* matched condition will halt the processing.
[5] Internal structure equivalent to `stopOn.Total` and not explicitly shown.
[6] Default is opcode 0; set to invalid opcode 256 if this feature is *not* desired.

Worker Properties

file_write_demux.rcc

Control Operations: Stop

Component Ports

Name	Producer	Protocol	Optional	Advanced	Usage
in	false	-	false	numberofopcodes=256	Data to be streamed to output file(s)

Worker Interfaces

There are no implementation-specific interfaces for this component.

Test and Verification

Usage (local/x86)

After building the component, the user needs to type `make tests RCC_CONTAINERS=1` in the *file_write_demux.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 *file_write_demux.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. 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.

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
- `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 the `pattern` component 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.

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`.

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