Summary - Advanced Pattern

Name	advanced_pattern
Worker Type	Application
Version	v1.3
Release Date	February 2018
Component Library	ocpi.assets.util_comps
Workers	advanced_pattern.rcc
Tested Platforms	linux-c7-x86_64, linux-x13_3-arm

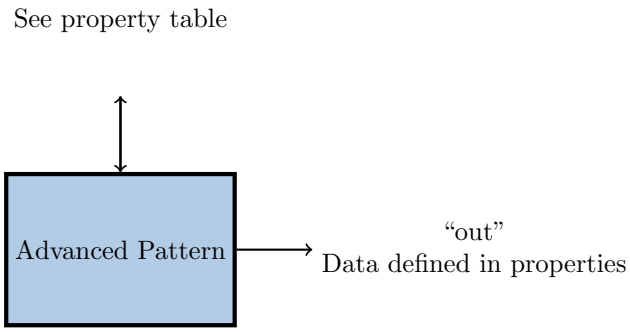
Functionality

The Advanced Pattern Component provides predefined data to assist in the testing of other Components.

The data can be arranged in messages of up to 2048 bytes at a time with each block having a specific opcode. By default, 32 of these messages are available, but that configuration is exposed as a build-time parameter with a default configuration building additional Workers allowing for 64, 128, and 256 messages.

Block Diagrams

Top level



Source Dependencies

advanced_pattern.rcc

- ocpiassets/components/util_comps/advanced_pattern.rcc/advanced_pattern.cc

Component Spec Properties

Name	Type	SequenceLength	ArrayDimensions	Accessibility	Valid Range	Default	Usage
`maxPatternLength`	ULong	-	-	Parameter	Standard	32	Maximum "**Pattern**" sequence length to allow [1]
`Pattern`	Struct	`maxPatternLength`	-	Initial, Readable	-	-	Message to send
`Pattern.Opcode`	UChar	-	-	"	Standard	0	Opcode metadata to send with this message's data
`Pattern.Bytes`	UChar	2048	-	"	Standard	0	Data to send
`LoopCount`	ULongLong	-	-	Initial, Readable	Standard	1	How many times to repeat the "**Pattern**" sequence[2]
`ZLM`	UShort	-	-	Initial, Readable	0 ... 256	0	Opcode for a **Z**ero **L**ength **M**essage with when finished.[3]
`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	Various[4]

[1] Each **Pattern** entry requires about 2K of RAM.
[2] 0 will continue as long as Worker is running.
[3] Default is opcode 0; set to invalid opcode 256 if this feature is *not* desired.
[4] Internal structure equivalent to `current.Total` and not explicitly shown.

Worker Properties

There are no implementation-specific properties for this component.

Component Ports

Name	Producer	Protocol	Optional	Advanced	Usage
out	true	-	-	numberofopcodes=256	Data defined in properties

Worker Interfaces

There are no implementation-specific interfaces for this component.

Performance and Resource Utilization

advanced_pattern.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-x13_3-arm ARMv7 Processor rev 0 (v7l)	666 MHz	TBD

Test and Verification

Usage (local/x86)

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

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 *advanced_pattern.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 internally generated and then written to disk using the `file_write_demux` component. Some OASs have the pattern information embedded within the XML itself, while others use the `valueFile` option to import a file named `UUT.Pattern.input`. Most OASs dump the "`current`" property from the tested component as well as the file writer to files named `UUT.current.dump` and `fwout.current.dump` respectively, which are 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`.