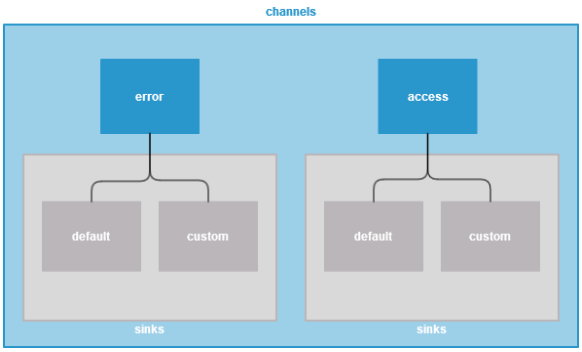
xUML Runtime Logger Configuration

Runtime 2020.7 On startup, the xUML Runtime accepts a file specifying a logger configuration (see `--logging.config.file` on xUML Runtime Command Line Options). The configuration file is a JSON file in which you can specify logging details like which logging data to write where.

Concepts

The logging concept of the xUML Runtime is build around the concepts of channels and sinks.



This file reflects the default logger configuration with some filters added.

On this Page:

Related Pages:

- xUML Service Standard Log
- xUML Service Transaction Log
- Contents of the Transaction Log
- xUML Runtime API

Related Documentation:

- boost Date Time Format Flags
- {fmt} Format String Syntax
- cppreference.com > std:: strftime

```
{
  "channels": {
    "access": {
      "sinks": {
        "default": {
          "type": "daily_rotated_file",
          "params": {
            "name_pattern": "logs/transaction_%Y-%m-%d.log"
          },
          "formatter": {
            "type": "pattern",
            "params": {
              "pattern": "{timestamp:%Y-%m-%d\t%T\t%z}\t{trx_id}\t
{session_id}\t{component}\t{elapsed_ms}\t{trx_status}\t{domain}\t
{trx_entry_type}\t{param1}\t{param2}\t{correlation_id}"
            }
          },
          "filters": [
            {
              "trx_level": ["SERVICE"],
              "domains": ["*", "!PSTATE", "!CONVERSATION"]
            },
            {
              "trx_level": ["IO_INTERNAL"],
              "domains": ["PSTATE", "CONVERSATION"]
            }
          ]
        }
      }
    },
    "error": {
      "sinks": {
        "default": {
          "type": "daily_rotated_file",
          "params": {
            "name_pattern": "logs/bridgeserver_%Y-%m-%d.log"
          },
          "formatter": {
            "type": "pattern",
            "params": {
              "pattern": "[{timestamp:%F %T %z}][{session_id:010}][{level}]
[Internal][{domain}][{code}][{message}]"
            }
          },
          "filters": [
            {
              "level": ["Info"],
              "domains": ["*"]
            }
          ]
        },
            "monitoring": {
          "type": "std_err",
          "formatter": {
            "type": "pattern",
            "params": {
              "pattern": "e2eruntime[{pid}[{timestamp:%F %T UTC}[{level}
[Internal[{domain}[{code}[[{message}]]"
            }
          }
        }
      }
    }
  }
}
```

Element	Description	Allowed Values / Examples	
channel	A channel is an object that describes the data that will be written to a log file. It is identified by a channel name. The following channel names are reserved for internal use of the xUML Runtime: • error • access • channels starting with "xUML" in any casing	`error`	Write service logging data (bridgeserver log).
		`access`	Write transaction logging data.
sink	A channel can contain an arbitrary number of sinks. Sinks define the logging output and how it is written: • log file name and path patterns • log file format • logged content Sink names are not important but you need them to access the logging configurations via the xUML Runtime API. Do not rename the sinks of the access and error channels.		

Configuring Sinks

Sinks define the logging output and how it is written. You can define name and path of the log file, the log file format and filter out data to be logged.

Property	Description	Allowed Values		Example
type	Define whether to write to a file (daily or hourly rotation), or to std_out or std_err.	`std_out`	Write logs to std_out.	`"type": "hourly_rotated_file", "params": { "name_pattern": logs/transaction_%Y-%m-%d-%H.log" }`
		`std_err`	Write logs to std_err.	
		`daily_rotated_file`	Write to a daily rotated log file.	
		`hourly_rotated_file`	Write to an hourly rotated log file.	
params	Define parameters of the sink. At the moment, only accepted parameter is `name_pattern` to define the name of the log file.	`name_pattern`	Defines the relative path and file name pattern of the log file. See Name Patterns for Log Files for available placeholders.	
formatter				
type	Define in which format the log data will be logged to the log file.	`json`	Log the log file data in JSON format.	**Example 1:** `"formatter" : { "type" : "pattern", "params": { "pattern" : "{timesta`

		`pattern`	Log the log file data according to a given pattern. The used pattern is defined in the **params** attribute via attribute **pattern**.	`mp:%Y-%m-%d\t%T\t%z}\t..."`
				`}`
				`}`
				
				Example 2:
				
				`"formatter" : {`
				`"type": "json"`
				`}`

params	If format type is `pattern`, specify the log file pattern.		See Format Variables for Log File Content for more on the available variables.	
legacy_utc	Enable UTC timestamp for monitoring logs. This is a legacy option. Do not use.	true	Use UTC in timestamp.	
		false	Do not use UTC in timestamp.	
filters				

level	Provide an array of service log levels to let pass through the filter.	One of `None`, `Fatal`, `Error`, `Warning`, `Info` and `Debug` (case-sensitive).
		Refer to Bridge Server Log Levels of an xUML Service for more information.
				`"filters" : [`
				`{`
				`"trx_level" : [ "SERVICE" ],`
				`"domains" : [ "*" , "!PSTATE" , "!CONVERSATION" ]`
				`},`
				`{`
				`"trx_level" : [ "IO_INTERNAL" ],`
				`"domains" : [ "PSTATE" , "CONVERSATION" ]`
				`}`
				`]`

trx_level	Provide an array of transaction log levels to let pass through the filter.	One of `NONE`, `CUSTOM`, `SERVICE`, `IO_EXTERNAL` and `IO_INTERNAL` (case-sensitive). Refer to Transaction Log Levels of an xUML Service for more information.		
domains	Provide an array of error domains let pass through the filter. You can invert the filtering by adding a `!` sign in front of a domain name.	`["*"]`	Nothing is filtered.	

		list of domains `["<domain_1>", "<domain_2>, ...]`	Let logs containing one of the listed domains pass through the filter. You can invert the filtering by adding a `!` sign in front of a domain name, so `!domain_1` will not let logs with this domain pass through the filter.	

Name Patterns for Log Files

As a name pattern for log files to use in attribute `name_pattern`, you can use all specifiers listed on boost Date Time Format Flags.

We recommend to use the following subset:

Pattern	Description
`%Y`	year, 4 digits
`%m`	month, 2 digits
`%d`	day of the month, 2 digits
`%H`	hour, 24-hour format, 2 digits
`%M`	minute, 2 digits
`%S`	second, 2 digits

Format Variables for Log File Content

Find below all available format variables for xUML service (bridgeserver) logs and transaction logs.

You can format the output using the `:` sign followed by a format string. Reasonable format strings are:

Pattern	Description	Example Output
Format Patterns For Date/Time Variables		

`%Y`	**Year** as a 4-digit decimal number.	`2020`
`%m`	**Month** as a 2-digit decimal number (range [01,12]).	`07`
`%d`	**Day** of the month as a decimal number (range [01,31]).	`15`
`%F`	Equivalent to `%Y-%m-%d`.	`2020-07-15`
`%H`	**Hour** as a 2-digit decimal number, 24 hour clock (range [00-23])	`16`
`%M`	**Minute** as a 2-digit decimal number (range [00,59])	`34`
`%S`	**Second** as a 2-digit decimal number (range [00,60])	`25`
`%T`	Equivalent to "%H:%M:%S" (the ISO 8601 time format).	`16:34:25`
`%z`	**Offset from UTC** in the ISO 8601 format, or no characters if the time zone information is not available.	`+0200`
Format Patterns for Numeric Variables		
`099`	**Sign-aware zero-padding** for numeric types, where 0 is the indicator to apply the padding and 99 the maximum padding width.	`010` 10-digit number with leading zeros.

If you want to apply more sophisticated formatting, find more options on the {fmt} documentation pages for string formats and cpp documentation pages for date/time formats.

Available Variables for Service Logs

Refer to xUML Service Standard Log for more information on this log.

Field	Type	Description	Example Usage
code	String	error code	`{code}`
domain	String	error domain	`{domain}`
level	String	error level	`{level}`
message	String	error message details	`{message}`
pid	Numeric	process id	`{pid}` `{pid:010}`
session_id	Numeric	session id	`{session_id}` `{session_id:010}`
timestamp	DateTime	timestamp when the log occurred	`{timestamp}` `{timestamp:%F %T %z}`

Available Variables for Transaction Logs

Refer to Contents of the Transaction Log for more information on the listed variables.

Field	Type	Description	Example Usage
component	String	name of the component	`{component}`
correlation_id	String	correlation ID	`{correlation_id}`
domain	String	error domain name	`{domain}`

elapsed_ms	Numeric	milliseconds since session start	`{elapsed_ms}` `{elapsed_ms:010}`
param1	String	parameter 1	`{param1}`
param2	String	parameter 2	`{param2}`
session_id	Numeric	session ID	`{session_id}`
timestamp	DateTime	timestamp when the log occurred	`{timestamp}` `{timestamp:%F %T %z}`
trx_entry_type	String	log type	`{trx_entry_type}`
trx_id	String	transaction ID	`{trx_id}`
trx_status	String	status	`{trx_status}`

Example Patterns for Emulating the Classic Log Format

Log	Example Definition
xUML Service Log (bridgeserver)	`{timestamp:%Y-%m-%d\t%T\t%z}\t{trx_id}\t{session_id}\t{component}\t{elapsed_ms}\t{trx_status}\t{domain}\t{trx_entry_type}\t{param1}\t{param2}\t{correlation_id}`
Transaction Log	`[{timestamp:%F %T %z}][{session_id:010}][{level}][Internal][{domain}][{code}][{message}]`
Monitoring	`e2eruntime[{pid}[{timestamp:%F %T UTC}[{level}[Internal[{domain}[{code}[[{message}]]]`