Logging (not tracing)

Logging from C

Gerrit repo: https://gerrit.o-ran-sc.org/r/admin/repos/com/log

RIC provides a C library for logging: mdclog. The library implements thread-local MDC management and formatting of the logs into the correct JSON format.

Logging from Go

The native Golang version of RIC logging library is in Gerrit: https://gerrit.o-ran-sc.org/r/admin/repos/com/golog

The Go version of the library works like the C version, except that the MDC values are not thread but logger instance specific. The mdclog C library can also be used from Go, but note that the thread-local MDCs do not work with Go goroutines. An example of how to use the mdclog C library from Go is shown below.

```
package main

/*
#cgo CFLAGS: -I/usr/local/include
#cgo LDFLAGS: -lmdclog
#
#include <mdclog/mdclog.h>
void my_log_write(mdclog_severity_t severity, const char *msg) {

  mdclog_write(severity, "%s", msg);
}
*/
import "C"
import "fmt"

func mylog(severity C.mdclog_severity_t, msg string) {

   C.my_log_write(severity, C.CString(msg))
}

func main() {

   C.mdclog_mdc_add(C.CString("Weather"), C.CString("Sunny"))
   C.mdclog_mdc_add(C.CString("Temperature"), C.CString("15.5"))
   mylog(C.MDCLOG_INFO, fmt.Sprintf("Weather report log"))
}
```

Logging from Python

Gerrit repo: https://gerrit.o-ran-sc.org/r/admin/repos/com/pylog You can also install the python library using pip: `python3 -m pip install mdclogpy`

RIC provides a Python library for logging: mdclogpy. The Python version of the library works like the Golang version, i.e. the MDC values are logger instance specific.

MDC (Mapped Diagnostic Context)

MDC is a list of key-value pairs the RIC components can define to be included in their logs. It is currently rarely used in RIC.

MDC key	MDC value (type, range, list of values, format, ...)	Description	Components using the MDC	Comments

Weather	Possible values: *sunny*, *cloudy*, *rainy*, *snowy*	Weather type included in every log made by the Weather Manager.	Weather Manager xApp	This is an example.
Temperature	Value range from -100.0 to 100.0 with one decimal. Example: *15.5*	Temperature (Celsius degrees) included in every log made by the Weather Manager.	Weather Manager xApp	This is an example.
time	yyyy-MM-dd HH:mm:ss.SSS	Human readable timestamp. Note that the timezone can be derived from comparing this value against the ts millisecond timestamp in the same log entry.	E2 Manager	Example: {"crit":"INFO","ts":1560266556006,"id":"E2Manager","msg":"#rmrCgoApi.Init - RMR router has been initiated","mdc":{"time":"2019-06-11 15:22:36.006"}}

Severity Levels

The following severity levels are used in RIC:

- DEBUG - information useful for debugging
- INFO - informational message related to normal operation
- WARNING - indication of a potential error
- ERROR - error condition

Fields in the log entry

- ts – Timestamp, number of milliseconds since Unix Epoch (i.e. 1970-01-01 00:00:00 +0000 (UTC)), set by the logging library
- crit – Severity level of the log, given by the application process: DEBUG, INFO, WARNING, ERROR
- id – the name of the process, set by the logging library
- msg – log message given by the application process
- mdc – a list of key value pairs, both strings, unique key names, given by the application process

The logging library writes the logs to stdout. Each log entry is one line. All line feed characters, as well as non-printable characters, are replaced with a space by the logging library. In addition, characters having a special meaning in the output format are escaped by the logging library. The logging library supports only JSON format, only. Having named fields is flexible from post-processing point of view and, as we might have new requirements from the applications in later releases, allows us to easily add, remove or modify the fields to the logs in the future, if needed. If other formats are needed, we can add them as configurable options in the future versions. The format of each field (e.g. timestamp) can also be made configurable in the future versions of the logging library, if needed. No separate “markers”, which are defined in the ONAP specification (https://wiki.onap.org/pages/viewpage.action?pageId=28378955#ONAPApplicationLoggingSpecificationv1.2(Casablanca)), are supported. Reasoning is that this is a tracing concept and needs to be covered by a tracing solution such as OpenTracing. It will cover the functionality of having specific ENTRY, EXIT, INVOKE etc marked logs. In the following examples, the process has logged an INFO log with message “This is an example log” and with two key-value pairs "key1"="value1" and "key2"=" value with a space". The one-line log has been divided into several lines for readability.

JSON

```
{
    "ts": 1550045469,
    "crit": "INFO",
    "id": "applicationABC",
    "msg": "This is an example log",
    "mdc":
    {
        "key1": "value1",
        "key2": "value with a space"
    }
}
```

CSV (proposal for future development)

With a comma as a separator in this example. The separator could be made configurable.

```
1550045469,
INFO,
applicationABC,
This is an example log,
key1=value1 key2=value with a space
```

Key-value pairs (proposal for future development)

With a colon as a separator in this example. The separator could be made configurable.

```
timestamp=1550045469:
severity=INFO:
logger=applicationABC:
message=This is an example log:
key1=value1:
key2=value with a space
```