INPUT | OUTPUT

LOGGING

contra-tracer

A minimal implementation of `Tracer m a` is sufficient to start tracing observables in any code. Simply create a trace and add transformers to process the observables.

Create a tracer

In this example, the terminal transformer outputs a text representation to the console:

```
let logTrace = showTracing stdoutTracer

showTracing
  :: (Show a)
  => Tracer m String
  -> Tracer m a

stdoutTracer
  :: (MonadIO m)
  => Tracer m String
```

Usage of the tracer argument in a function

For example:

```
handleBlockHeader logTrace = do
  ...
  curBlockHeader :: Header <- ...
  traceWith logTrace curBlockHeader
  ...
```

Hierarchy of traces

The derived Trace m a type accepts LogObject for further processing and routing depending on the context name.

A hierarchy of traces can be created by entering a local namespace with:

```
trace' <- appendName "deeper" trace
```

Observables on the new trace are annotated with the extended name.

Privacy annotation

The logged observables can be annotated with either `Confidential` or `Public`. This will be taken into account when dispatching to the `katip` backend and its output scribes which are itself annotated to only process observables of a certain kind.

Trace with severity annotation

This code:

```
trace <- setupTrace (Left "config.yaml")
                    "test"
logInfo trace "Info"
logError trace "Error"
```

leads to outputs:

```
[iohk.test:Info:ThreadId 104] [2019-03-27
   07:30:32.37 UTC] "Info"
[iohk.test:Error:ThreadId 104] [2019-03-27
    07:30:32.37 UTC] "Error"
```

Configuration

The configuration is loaded on startup from a YAML file.

```
bTrace <- setupTrace (Left "config.yaml")
                     "base"
```

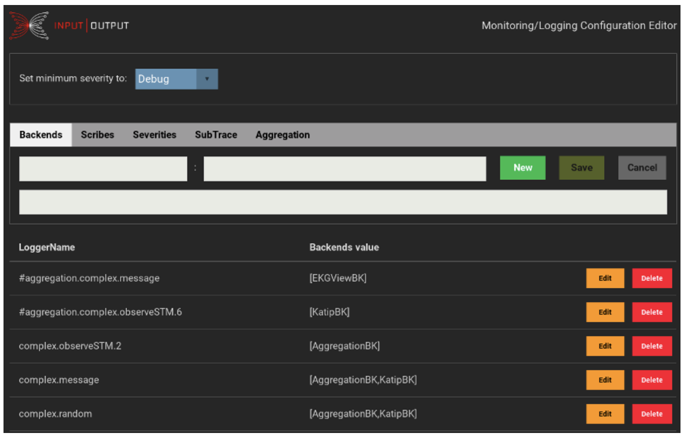
The returned trace in the namespace `"base"` will route all messages to the `Switchboard` which dispatches the messages to the backends according to configuration. The `katip` backend handles output to the console or log files, and also includes the log rotator.

The configuration contains mappings from the named context `LoggerName` to values changing the trace's behaviour.

Run:

```
cabal new-run example-complex
```

and access the editor on http://localhost:13789.



Set minimum severity to: Debug

LoggerName	Backends value
#aggregation.complex.message	[EKGViewBK]
#aggregation.complex.observeSTM.6	[KatipBK]
complex.observeSTM.2	[AggregationBK]
complex.message	[AggregationBK,KatipBK]
complex.random	[AggregationBK,KatipBK]

Changes are effective immediately, because the changed configuration will be queried on arrival of every traced message.