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

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

transformers

This usage of a `Tracer m a` can later be adapted to include filtering tracer transformers such as:

```
let logTrace' = condTracing checkP logTrace

condTracing :: (a -> Bool) -> Tracer m a
                           -> Tracer m a
```

Or, to forward to the more elaborate logging and monitoring capabilities of `Trace m a` via the `toLogObject` transformer, without changing the call site.

```
toLogObject :: Tracer m (LogObject a) -> Tracer m a
```

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.

Example Trace with severity annotation

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

leads to outputs:

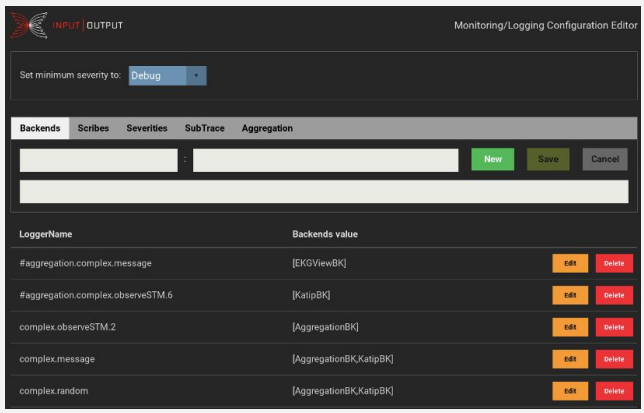
[iohk.test:Info:ThreadId 104] [2019-03-27 07:30:32.37 UTC] "Info"
[iohk.test:Error:ThreadId 104] [2019-03-27 07:30:32.37 UTC] "E..

Configuration - output selection

The configuration is loaded on startup from a YAML file.

```
baseTrace <- 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.



cabal new-run example-complex and access the editor on http://localhost:13789
Changes are effective immediately, because the changed configuration will be queried on arrival of every traced message.