ETH zürich

Software Tutorial
Signal Logger

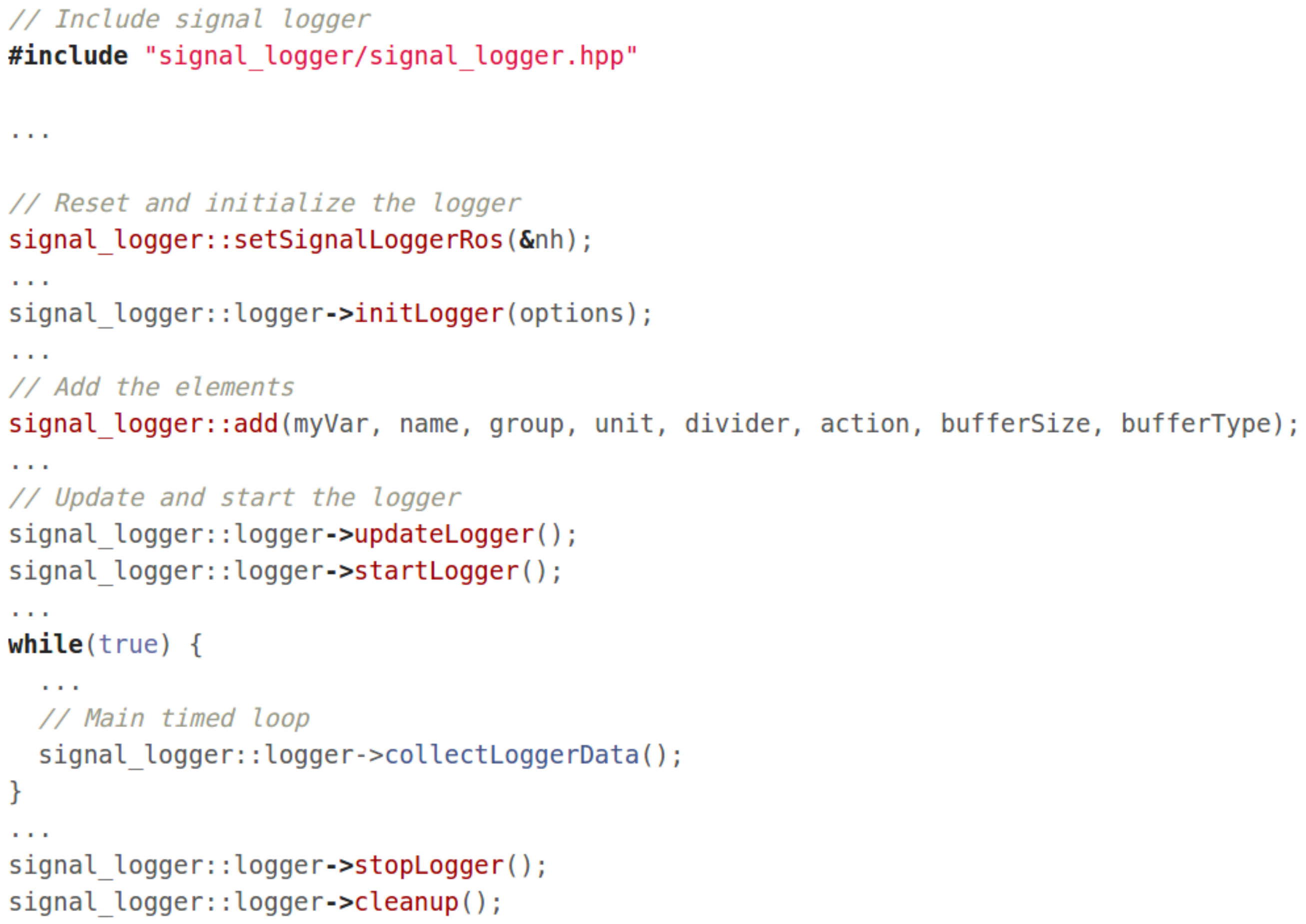
Gabriel Hottiger

Robotic Systems Lab

What should a logger do?

- Log ‘any’ type, including custom types
- Log elements at different frequencies
- Customize logging at post-compile time / runtime
- Fast data collection (usually happens in main thread)
- Log ‘exact’ time of collection
- Save data to a file, with easy-to-parse file format
- Simple interface that allows quickly adding / removing elements to the logger
- Publish logged elements via topic, record data to a bag

Code snippet



```
// Include signal logger
#include "signal_logger/signal_logger.hpp"

...

// Reset and initialize the logger
signal_logger::setSignalLoggerRos(&nh);
...
signal_logger::logger->initLogger(options);
...
// Add the elements
signal_logger::add(myVar, name, group, unit, divider, action, bufferSize, bufferType);
...
// Update and start the logger
signal_logger::logger->updateLogger();
signal_logger::logger->startLogger();
...
while(true) {
  ...
  // Main timed loop
  signal_logger::logger->collectLoggerData();
}
...
signal_logger::logger->stopLogger();
signal_logger::logger->cleanup();
```

Logger Options

Update Frequency	Frequency at which collectLoggerData() is called.	f_c
Max Logging Time	Maximal logging time.	t_{max}
Script Filename	Yaml file format, customizes the logger by setting element options.	*“myScript.yaml”*
Prefix	Prefix to the log element names.	*Default: “/log”*

Logger Element Options

Name	Full name (ns + element name)	*“/log/ns/foo”*
Unit	Physical unit	*“m/s”, “kg”*
Divider	Defines logging frequency (f_E) relative to the collection frequency (f_C) (Pos. Int)	divider = $d = f_C / f_E$ $d = 2$ → *log at half* f_C
Action	Defines how element is processed	*SAVE, PUBLISH, BOTH*
Buffer Size	Size of the buffer, number of elements to be stored	$b = t_{max} * f_c / d$ t_{max} → *maximal logging time*
Buffer Type	Type of the buffer	LOOPING, FIXED, GROWING

Robotic Systems Lab

Circular Buffer Implementation

- Thread-safe implementation of boost::circular_buffer
- Three Buffer Types

Fixed Size	Has a fixed size. Once the buffer is full, no elements can be added to the buffer.
Looping	Has a fixed size. Once the buffer is full, the oldest element is overwritten by the new one.
Growing	Once the buffer is full, the buffer is resized in an exponential manor. **Increasing buffer size (reallocating) is very time inefficient!**

Log time

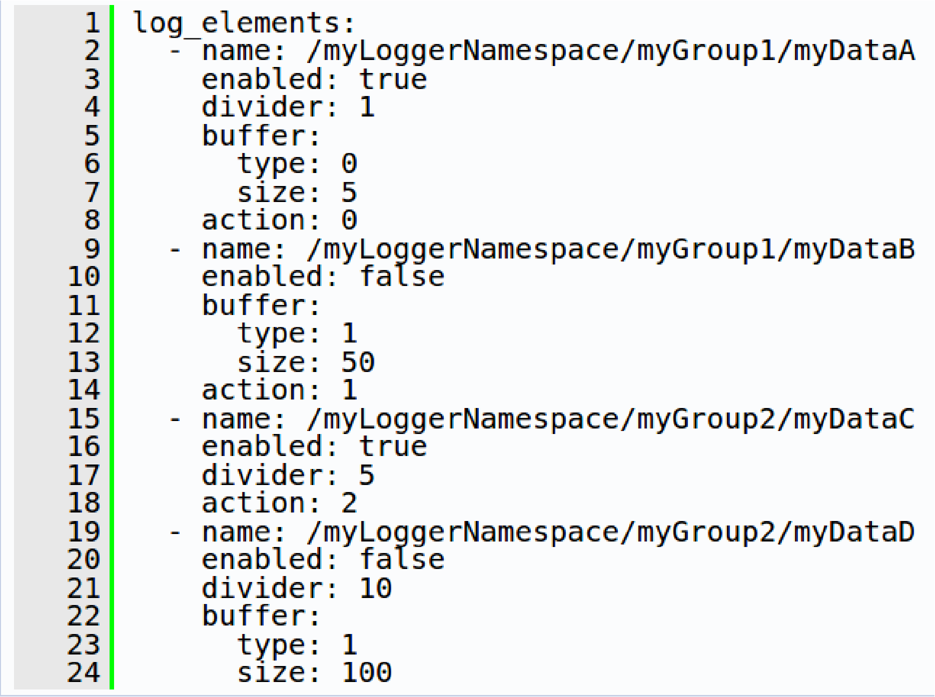
- A single time element is logged for all log elements
- Time is matched to element in publish
- LoggerStd logs system clock, LoggerRos logs ros::Time
- Three different buffers, depending on settings

t_{max} != 0	Fixed size buffer of size $n = f_c * t_{max}$. Once the buffer is full logging automatically stops.
t_{max} == 0 (default)	Growing buffer with initial size of $n = 10 * f_c$. **Very time inefficient → reallocation.**
Buffertype == Looping for all elements	Looping buffer of size n = $max(d * b)$.

Robotic Systems Lab

Script File

- Yaml file that lists all logger elements with options
- If an element is not listed it is automatically logged

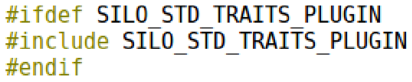


```yaml
log_elements:
  - name: /myLoggerNamespace/myGroup1/myDataA
    enabled: true
    divider: 1
    buffer:
      type: 0
      size: 5
    action: 0
  - name: /myLoggerNamespace/myGroup1/myDataB
    enabled: false
    buffer:
      type: 1
      size: 50
    action: 1
  - name: /myLoggerNamespace/myGroup2/myDataC
    enabled: true
    divider: 5
    action: 2
  - name: /myLoggerNamespace/myGroup2/myDataD
    enabled: false
    divider: 10
    buffer:
      type: 1
      size: 100
```

Log custom Types

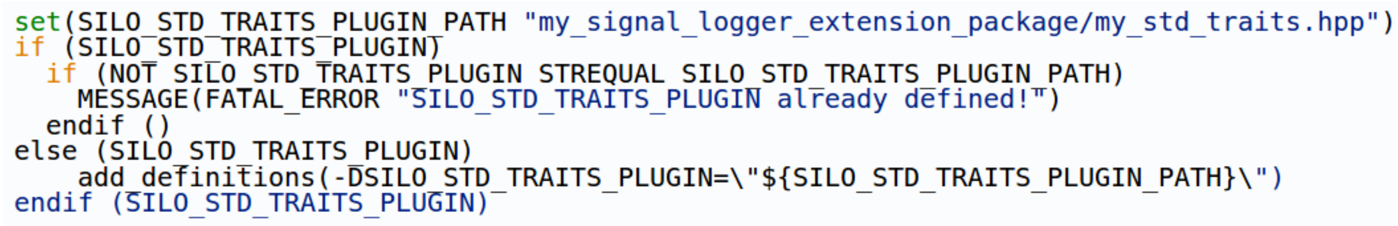
- Heavily uses traits for saving / publishing data
  → provide traits for your custom type

- Use same „trick“ as Eigen uses to extend MatrixBase
  - Include header via define



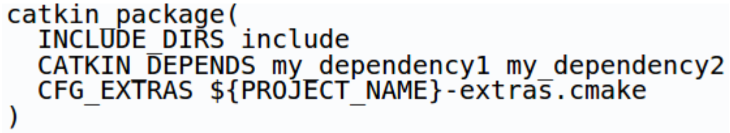
```
#ifdef SILO_STD_TRAITS_PLUGIN
#include SILO_STD_TRAITS_PLUGIN
#endif
```

  - provide a cmake-extras file (e.g ${PROJECT_NAME}-extras.cmake)



```
set(SILO_STD_TRAITS_PLUGIN_PATH "my_signal_logger_extension_package/my_std_traits.hpp")
if (SILO_STD_TRAITS_PLUGIN)
  if (NOT SILO_STD_TRAITS_PLUGIN STREQUAL SILO_STD_TRAITS_PLUGIN_PATH)
    MESSAGE(FATAL_ERROR "SILO_STD_TRAITS_PLUGIN already defined!")
  endif ()
else (SILO_STD_TRAITS_PLUGIN)
    add_definitions(-DSILO_STD_TRAITS_PLUGIN=\"${SILO_STD_TRAITS_PLUGIN_PATH}\")
endif (SILO_STD_TRAITS_PLUGIN)
```

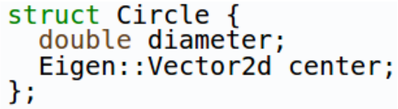
  - Add the file to the CFG_EXTRAS of your package



```
catkin_package(
  INCLUDE_DIRS include
  CATKIN_DEPENDS my_dependency1 my_dependency2
  CFG_EXTRAS ${PROJECT_NAME}-extras.cmake
)
```

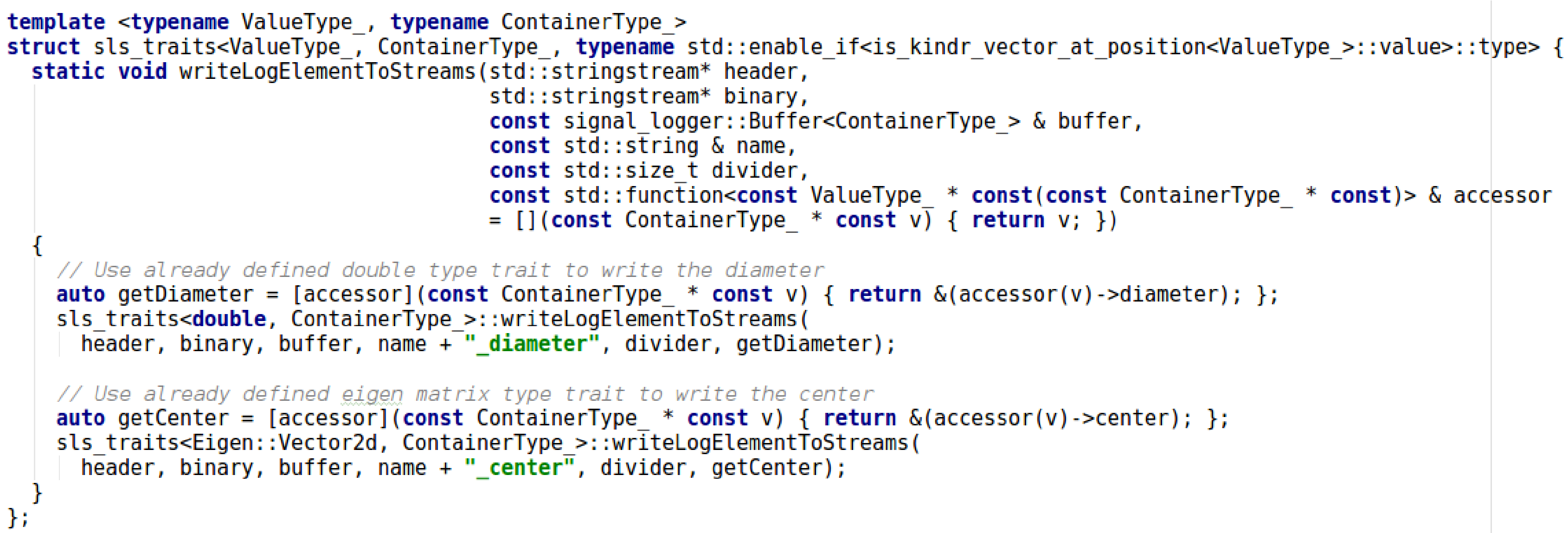
Writing custom Traits

- Example: Log a Circle type



```
struct Circle {
  double diameter;
  Eigen::Vector2d center;
};
```

- Complex Syntax, but very easy to use (already defined traits can be reused)



```
template <typename ValueType_, typename ContainerType_>
struct sls_traits<ValueType_, ContainerType_, typename std::enable_if<is_kindr_vector_at_position<ValueType_>::value>::type> {
  static void writeLogElementToStreams(std::stringstream* header,
                                       std::stringstream* binary,
                                       const signal_logger::Buffer<ContainerType_> & buffer,
                                       const std::string & name,
                                       const std::size_t divider,
                                       const std::function<const ValueType_ * const(const ContainerType_ * const)> & accessor
                                       = [](const ContainerType_ * const v) { return v; })
  {
    // Use already defined double type trait to write the diameter
    auto getDiameter = [accessor](const ContainerType_ * const v) { return &(accessor(v)->diameter); };
    sls_traits<double, ContainerType_>::writeLogElementToStreams(
      header, binary, buffer, name + "_diameter", divider, getDiameter);

    // Use already defined eigen matrix type trait to write the center
    auto getCenter = [accessor](const ContainerType_ * const v) { return &(accessor(v)->center); };
    sls_traits<Eigen::Vector2d, ContainerType_>::writeLogElementToStreams(
      header, binary, buffer, name + "_center", divider, getCenter);
  }
};
```

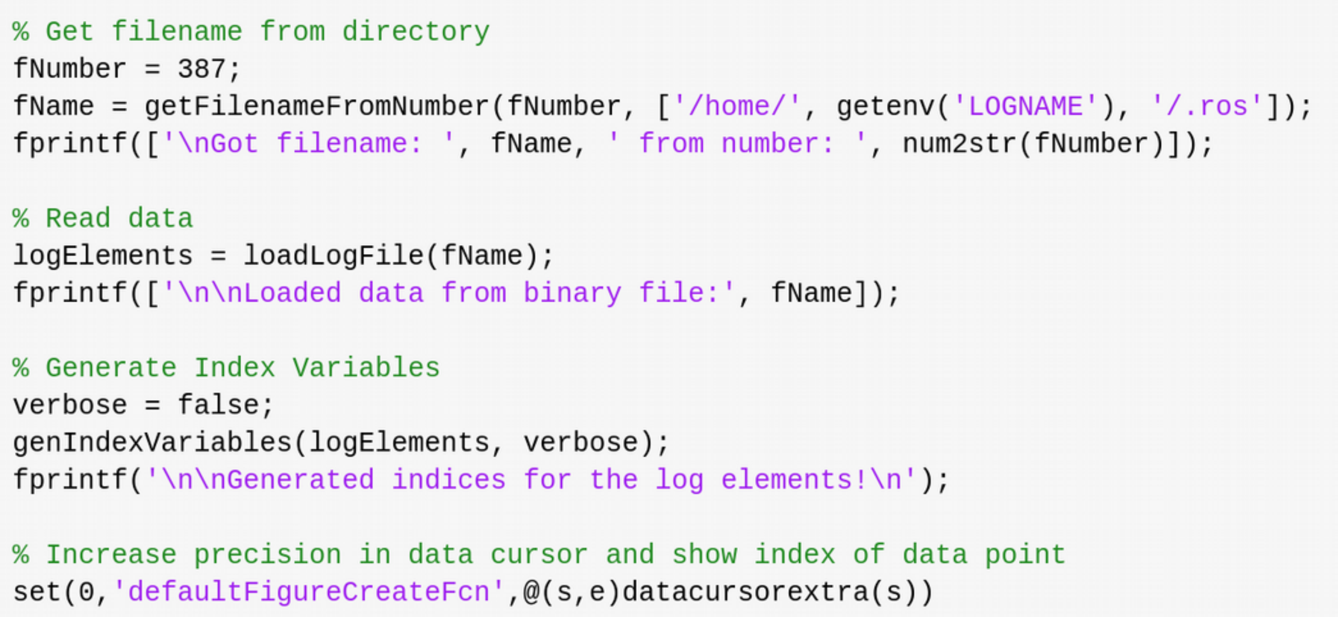
Log File

- The std logger saves the data in a binary file
- File name convention: silo_#y#m#d_#H-#M-#S_#NR (e.g. silo_20160913_12-13-49_00113)

```
#  --------------------------------------------------------- #
# | DO NOT EDIT THIS FILE! It could corrupt the binary data! |
#  --------------------------------------------------------- #
# Log File: silo_20170406_14-50-10_00033
# Time synchronization offset:
1
# Number of Log Elements:
504
# (Element Name) (Data Size In Bytes) (No Data Points) (Divider) (Buffer looping (1 or 0))
(Data Type)
/log/time_s 8 1 1 1 int64
/log/time_ns 8 1 1 1 int64
/log/iReallyWantToLogThisDouble 8 134 1 1 double
```

Post-process Log File

- Matlab parser of log file is provided



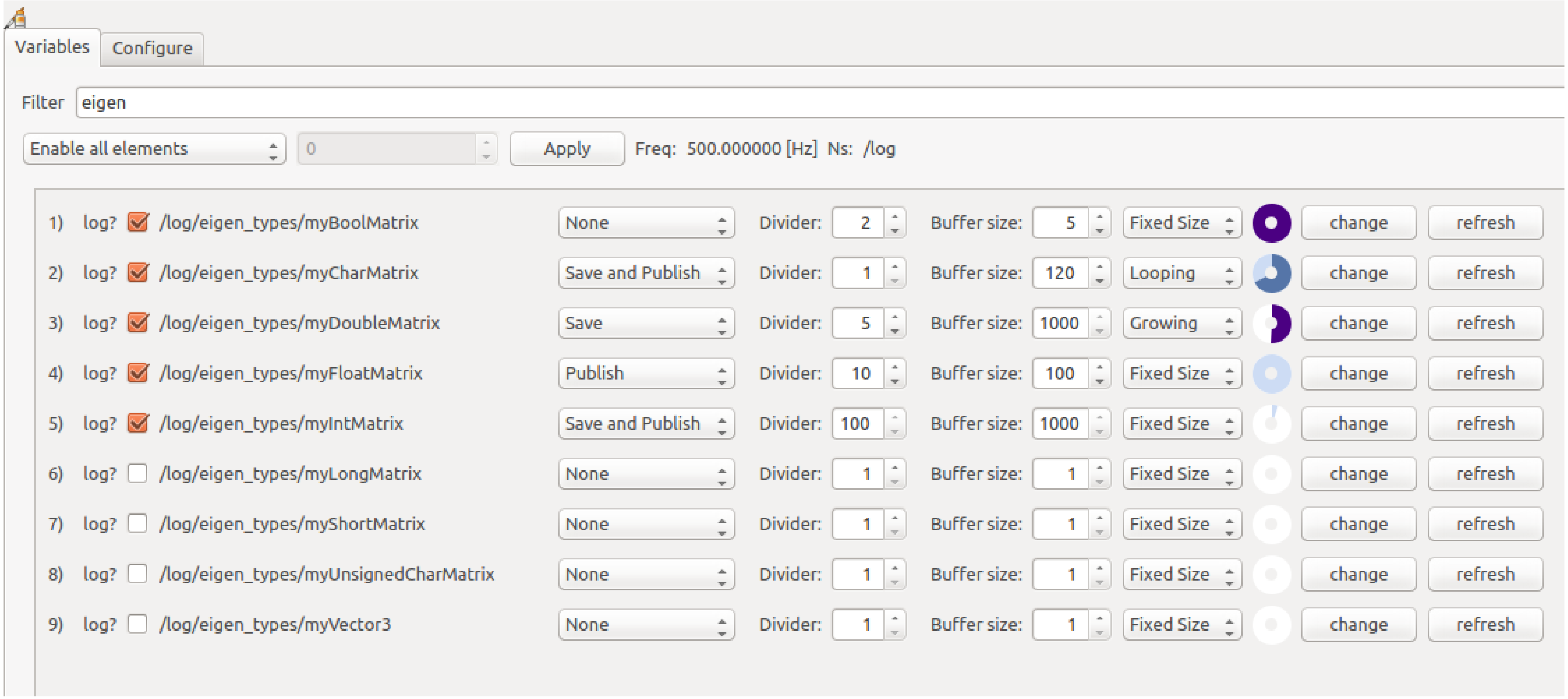
```
% Get filename from directory
fNumber = 387;
fName = getFilenameFromNumber(fNumber, ['/home/', getenv('LOGNAME'), '/.ros']);
fprintf(['\nGot filename: ', fName, ' from number: ', num2str(fNumber)]);

% Read data
logElements = loadLogFile(fName);
fprintf(['\n\nLoaded data from binary file:', fName]);

% Generate Index Variables
verbose = false;
genIndexVariables(logElements, verbose);
fprintf('\n\nGenerated indices for the log elements!\n');

% Increase precision in data cursor and show index of data point
set(0,'defaultFigureCreateFcn',@(s,e)datacursorextra(s))
```

RQT Plugin



Variables | Configure

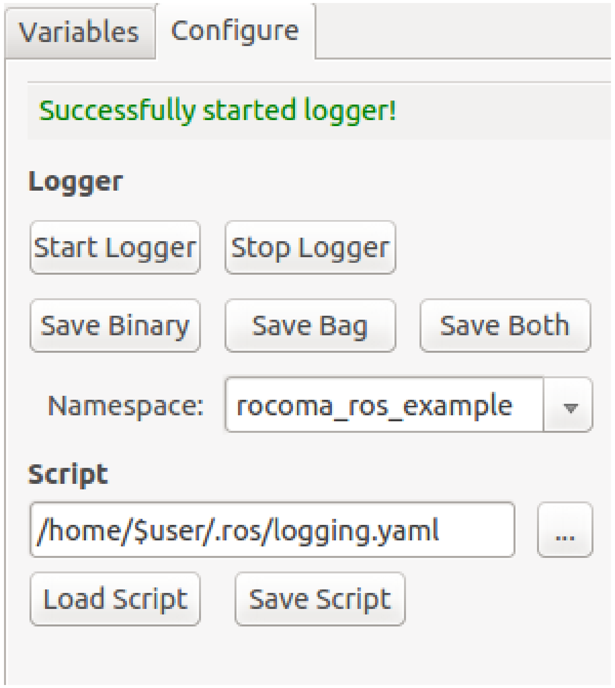
Filter: eigen

Enable all elements | 0 | Apply | Freq: 500.000000 [Hz] Ns: /log

1)	log?	☑	/log/eigen_types/myBoolMatrix	None	Divider:	2	Buffer size:	5	Fixed Size	●	change	refresh
2)	log?	☑	/log/eigen_types/myCharMatrix	Save and Publish	Divider:	1	Buffer size:	120	Looping	●	change	refresh
3)	log?	☑	/log/eigen_types/myDoubleMatrix	Save	Divider:	5	Buffer size:	1000	Growing	●	change	refresh
4)	log?	☑	/log/eigen_types/myFloatMatrix	Publish	Divider:	10	Buffer size:	100	Fixed Size	●	change	refresh
5)	log?	☑	/log/eigen_types/myIntMatrix	Save and Publish	Divider:	100	Buffer size:	1000	Fixed Size	○	change	refresh
6)	log?	□	/log/eigen_types/myLongMatrix	None	Divider:	1	Buffer size:	1	Fixed Size	○	change	refresh
7)	log?	□	/log/eigen_types/myShortMatrix	None	Divider:	1	Buffer size:	1	Fixed Size	○	change	refresh
8)	log?	□	/log/eigen_types/myUnsignedCharMatrix	None	Divider:	1	Buffer size:	1	Fixed Size	○	change	refresh
9)	log?	□	/log/eigen_types/myVector3	None	Divider:	1	Buffer size:	1	Fixed Size	○	change	refresh

RQT Plugin

- Select the correct namespace
- Script will be stored on the robot
- Variables are only changeable if the logger is not running


Successfully started logger!

Namespace: rocoma_ros_example

Script: /home/$user/.ros/logging.yaml