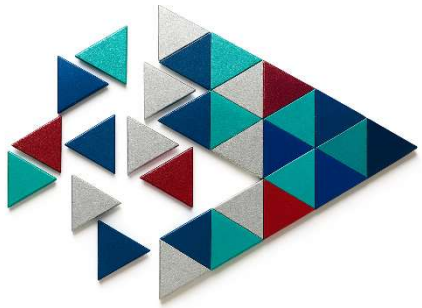
DIME CONNECTOR FIELD MANUAL

Revision 11/2025



CONTENTS

What is DIME? 5
How DIME Works 5
Polling source vs queueing source connectors 5
How data flows through the framework 6
How DIME is configured 6
Getting Started 8
Windows 8
Console 8
Windows Service 9
Running Multiple Instances 9
Docker 9
Configure Port Mappings and Volumes 9
Example Docker-Compose 10
Logging 10
Status Server 12
Viewing Metrics 13
Configuration Management 14
Configuration 15
App Configuration Section 15
Sinks Configuration Section 15
Sources Configuration Section 15
Items 17
Scripting 19
Script Execution Flow 20
The Result Variable 21
Caches 21
This Variable 22
Available Functions 22
Exercises 23

Writing to console and log 23
Split configuration across multiple files. 24
Write data to an embedded MTConnect Agent. 24
Add another source. 25
Write data to an external MTConnect Agent 25
Emit messages. 26
Add MTConnect Availability. 27
Add an extended MTConnect data item 27
Add data from Haas MTConnect Agent. 27
Add data from an ASC Autoclave 28
Reference 29
Connector Configuration Layout 30
Common Layout - Source Connector 30
Common Layout - Sink Connector 32
ActiveMQ 33
ASC CPC 34
Beckhoff ADS 35
Claude Code 36
Console 37
CSV Writer 38
Ethernet/IP 39
Fanuc Robot 40
Haas SHDR 42
HTTP Client 44
HTTP Server – Source 45
HTTP Server – Sink 46
I3X Web Server 47
InfluxLP 48
JSONWebScraper 49
Logger 50

ModbusTCP 51
MongoDB 52
MQTT – Source 53
MQTT – Sink 55
MSSQL 56
MTConnect Agent – Source 57
MTConnect Agent – SInk 58
MTConnect SHDR 59
MTConnect MQTT 60
Nws Weather 61
OPC-DA 62
OPC-UA - Source 63
OPC-UA – Sink 65
Postgres 66
Redis – Source 67
Redis – Sink 68
ROS2 69
Script 70
Siemens S7 71
SNMP 72
SparkplugB – Source 73
SparkplugB – Sink 74
Splunk EH SDK (Version 1.0) 75
Splunk EH SDK (Version 2.0) 76
Splunk HEC 77
TcpASCII 78
TImescaleWS 79
TrakhoundHTTP 80
UDP Server 81
WebsocketServer 82

Wintriss SmartPAC 83
XMLWebScraper 84
Yaskawa Robot 85
Connector Lifecycle 87
Fanuc Driver for Fanuc CNC Machines 90

WHAT IS DIME?

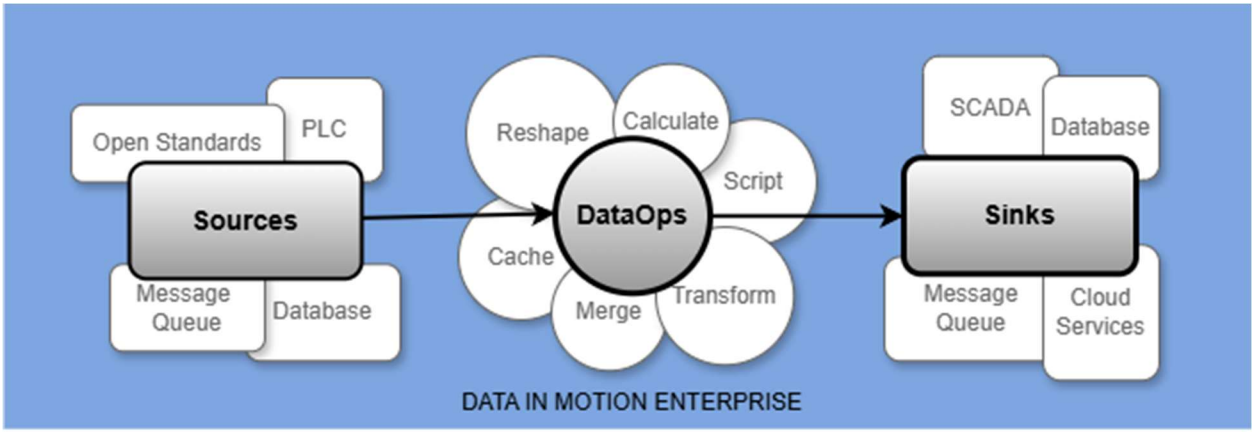
Data In Motion Enterprise (DIME) Connector is an extensible software framework to move and reshape data from enterprise and industrial sources to message queues, databases, and other sinks.

Features:

- 50+ enterprise and industrial sources and sinks in a single platform.
- Python, Lua, Scriban and Liquid scripting.
- Deployable as a console application, Windows Service or Docker container.
- Embeddable into other applications.
- Extensible framework allows quick addition of new source and sink connectors.
- User interface for modeling and testing configurations.

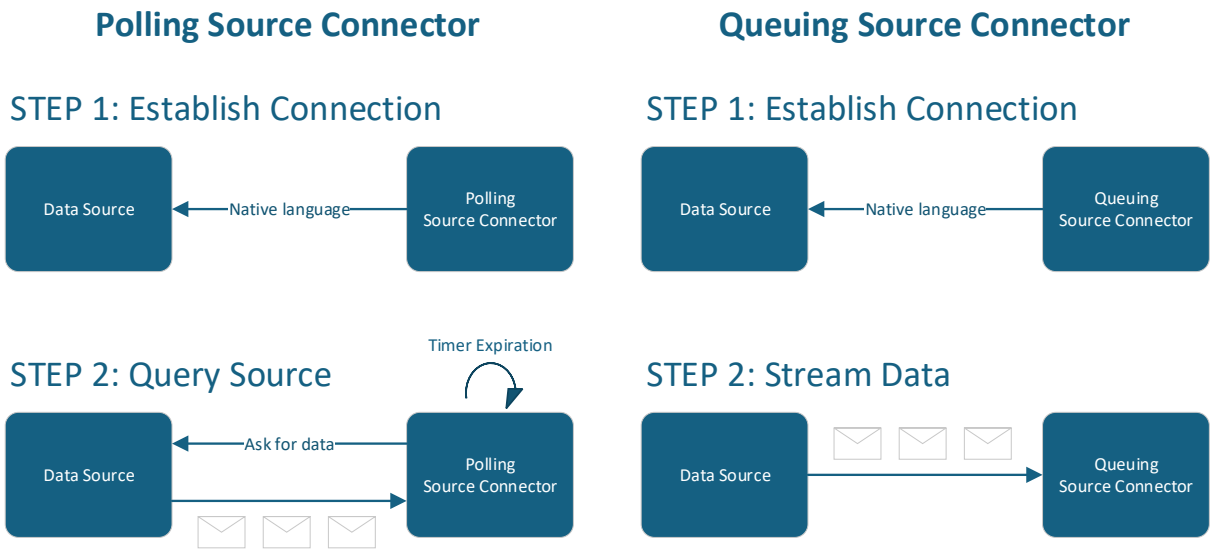
HOW DIME WORKS

There are data source and data sink connectors. Data moves from a source to a sink, while typically being transformed in the process. Both sources and sinks speak their own languages and protocols. DIME's connectors adapt each unique source and sink to a common format where calculation, data shaping, and transformation can take place.



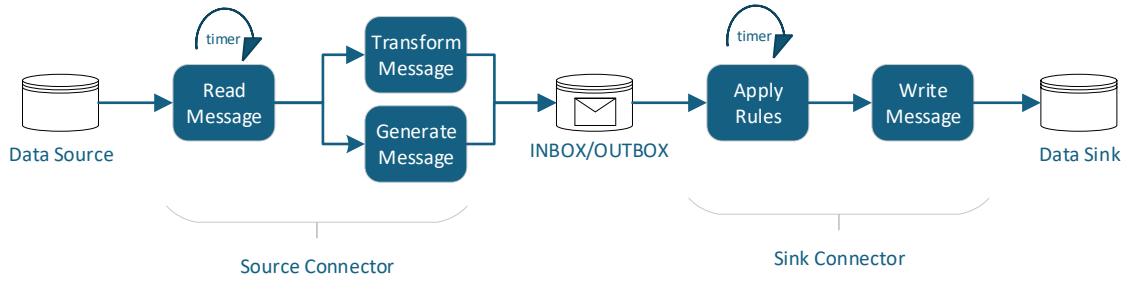
Polling source vs queueing source connectors

There are two types of source connectors, polling and queuing. A polling source is pull based where a timer runs at a specified interval, querying the data source on its expiration. Sources such as databases and PLCs are typically polling sources. A queueing source, on the other hand, is push based where the data arrives at the source connector as it's produced at the source. Message queues and some sockets are typically queuing sources.



How data flows through the framework

Once data arrives at the source connector, it is transformed into a common format and placed inside of a message inbox. At a predetermined interval, contents of the message inbox are published to each sink connector's message outbox. Each sink connector processes the outbox messages and delivers them to its target device.



How DIME is configured

A set of YAML files located in `./Configs` are used to manage the framework's behavior. To the right is a top-level example of a `main.yaml` file. This file has three sections:

- `app` – Framework-level configuration.
- `sinks` – Data sinks.
- `sources` – Data sources.

```yaml
app:
  license: 572E-74CD-F761-D0E4-0385-0991-9CD8-AF1A
  ring_buffer: !!int 4096
  http_server_uri: http://127.0.0.1:9999/
  ws_server_uri: ws://127.0.0.1:9998/
sinks:
  - *shdrSink1
  - *console
sources:
  - *s7Source1
```

main.yaml

In the above example, the configuration is split across four YAML files (`main.yaml, shdr.yaml, console.yaml, s7.yaml`) with the help of YAML anchors where ampersand (`&`) is used to define a value and asterisk (`*`) is used to reference a value. This is not required; however, it makes the configuration more modular and succinct. The only required file is `main.yaml` which can contain the contents of all four files.

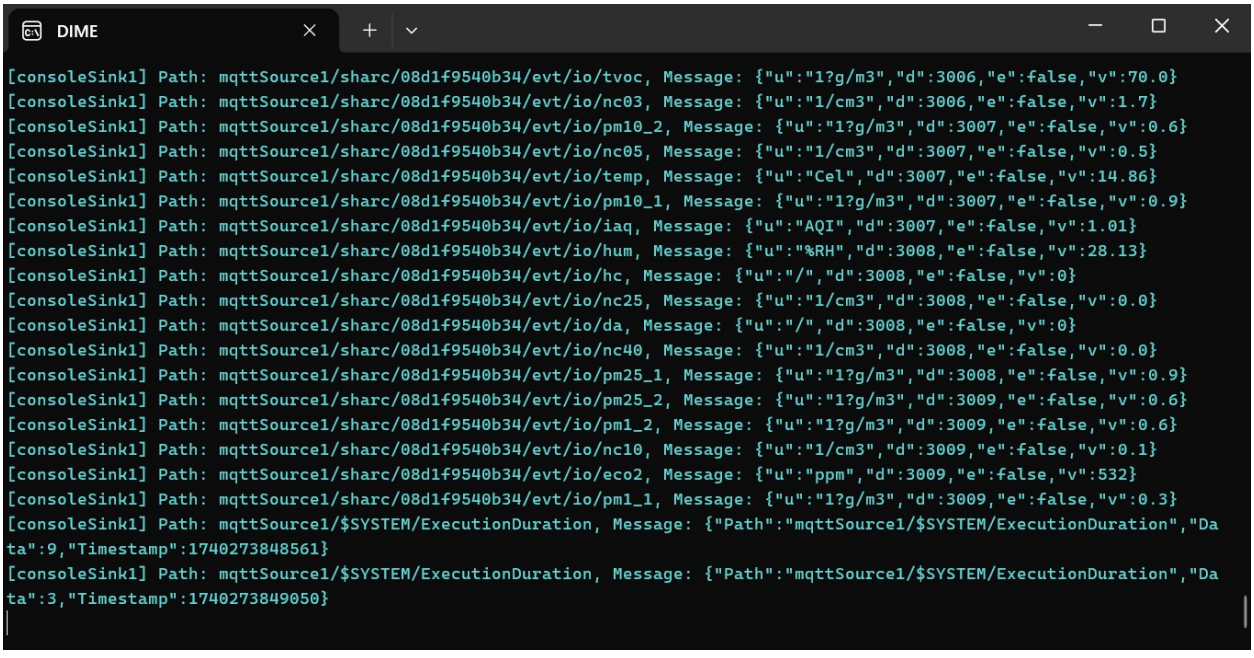
GETTING STARTED

Windows

DIME releases are maintained on Github: https://github.com/DataInMotionEnterprise/DIME-Releases. Download the latest Windows release and unzip the file. The 32-bit binary supports legacy protocols, such as OPC-DA.

CONSOLE

Run `DIME.exe`. To exit, press `CTRL-C`. The default configuration in `./Configs` folder moves data from an MQTT broker and writes the results to the console.



WINDOWS SERVICE

Open command prompt as administrator. To install the service, type `DIME.exe install`. To start the service either use the Services snap-in or type `DIME.exe start`. To stop and uninstall the service, type `DIME.exe stop` followed by `DIME.exe uninstall`.

RUNNING MULTIPLE INSTANCES

You can run multiple instances of the same DIME executable by pointing each one to a different configuration folder. By default, the `./Configs` solution folder is the startup configuration. This can be overwritten with `DIME.exe /config:c:/myCustomConfigs` or a relative path such as `DIME.exe /config:./Configs/myFirstConfig`. Similarly, you can pass a custom configuration path when installing the service, for example, `DIME.exe install /instance:DimeOne /config:c:/myFirstConfig` and `DIME.exe install /instance:DimeTwo /config:c:/mySecondConfig`.

Docker

DIME images are maintained on Docker Hub: https://hub.docker.com/r/datainmotionenterprise. Pull the latest `datainmotionenterprise/connector` image, `docker pull datainmotionenterprise/connector:latest`. Create and run the container, `docker run --name dime -d datainmotionenterprise/connector:latest`. The default configuration moves data from an MQTT broker and displays it on the screen. You can follow the container logs with `docker logs -f --tail 100 dime`. Once you are done, you can stop the container, `docker container stop dime`, and remove the container, `docker container rm dime`.

CONFIGURE PORT MAPPINGS AND VOLUMES

For more control over our configuration we will map ports and volumes, and use docker compose to deploy a DIME instance.

1. Clone the repository to your home folder.

```
cd ~
git clone https://github.com/DataInMotionEnterprise/DIME-Connector.Docker
```

2. Create directories for the volumes.

```
cd ~
mkdir -p volumes/dime/connector/configs
mkdir -p volumes/dime/connector/lua
mkdir -p volumes/dime/connector/python
mkdir -p volumes/dime/connector/logs
```

3. Copy files into the directories.

```
cp DIME-Connector.Docker/nlog.config volumes/dime/connector/nlog.config
cp -r DIME-Connector.Docker/Configs/Basic/* volumes/dime/connector/configs
cp -r DIME-Connector.Docker/Lua/* volumes/dime/connector/lua
cp -r DIME-Connector.Docker/Python/* volumes/dime/connector/python
```

4. Create and start the container.

```
docker run \
   -p 5000:5000 \
```

```
   -p 7878:7878 \
   -p 8080:8080 \
   -p 8081:8081 \
   -p 8082:8082 \
   -p 9998:9998 \
   -p 9999:9999 \
   -v ~/volumes/dime/connector/nlog.config:/app/nlog.config \
   -v ~/volumes/dime/connector/configs:/app/Configs \
   -v ~/volumes/dime/connector/lua:/app/Lua \
   -v ~/volumes/dime/connector/python:/app/Python \
   -v ~/volumes/dime/connector/logs:/app/Logs \
   datainmotionenterprise/connector:latest
```

EXAMPLE DOCKER-COMPOSE

```
services:
  dime:
    container_name: dime
    image: datainmotionenterprise/connector:latest
    restart: unless-stopped
    networks:
      - dime
    ports:
      - "5000:5000"
      - "7878:7878"
      - "8080:8080"
      - "8081:8081"
      - "8082:8082"
      - "9998:9998"
      - "9999:9999"
    volumes:
      - /etc/localtime:/etc/localtime:ro
      - /etc/timezone:/etc/timezone:ro
      - ~/volumes/dime/connector/nlog.config:/app/nlog.config:ro
      - ~/volumes/dime/connector/configs:/app/Configs:rw
      - ~/volumes/dime/connector/lua:/app/Lua:ro
      - ~/volumes/dime/connector/python:/app/Python:ro
      - ~/volumes/dime/connector/logs:/app/Logs:rw
    logging:
      driver: "json-file"
      options:
        max-file: "5"
        max-size: "1m"

networks:
  dime:
    name: dime
```

Logging

Logging is configured in the `./nlog.config` file. Logs are written to the `./Logs` subdirectory.

Multiple loggers are configured by default to a low verbosity of Warning.

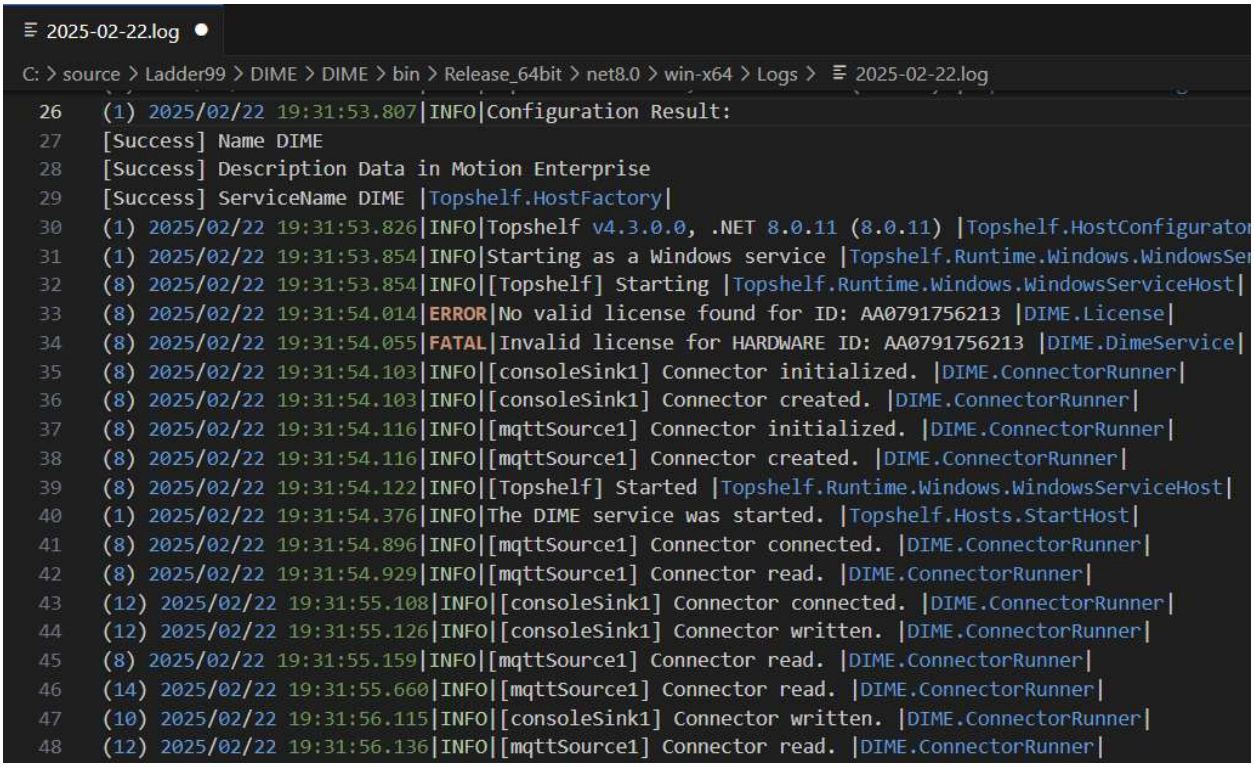
```
<logger name="Topshelf.*" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.License" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.LuaRunner" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.PropertyBag" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.SinkMessageHandler" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.FilesystemYamlConfigurationProvider" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.DimeService" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.Configurator.*" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.ConnectorRunner" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.Connectors.*.Source" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.Connectors.*.Sink" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.ConnectorSupport.*" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
```

To increase verbosity, you can either set each logger's `minlevel` to `INFO`, `DEBUG`, or `TRACE`, or comment all loggers and use a single wildcard logger.

```
<!--<logger name="Topshelf.*" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.License" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.LuaRunner" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.PropertyBag" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.SinkMessageHandler" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.FilesystemYamlConfigurationProvider" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.DimeService" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.Configurator.*" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.ConnectorRunner" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.Connectors.*.Source" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.Connectors.*.Sink" minlevel="WARN" writeTo="fileTarget,consoleTarget" />
<logger name="DIME.ConnectorSupport.*" minlevel="WARN" writeTo="fileTarget,consoleTarget" />-->
<logger name="*" minlevel="TRACE" writeTo="fileTarget,consoleTarget" />
```

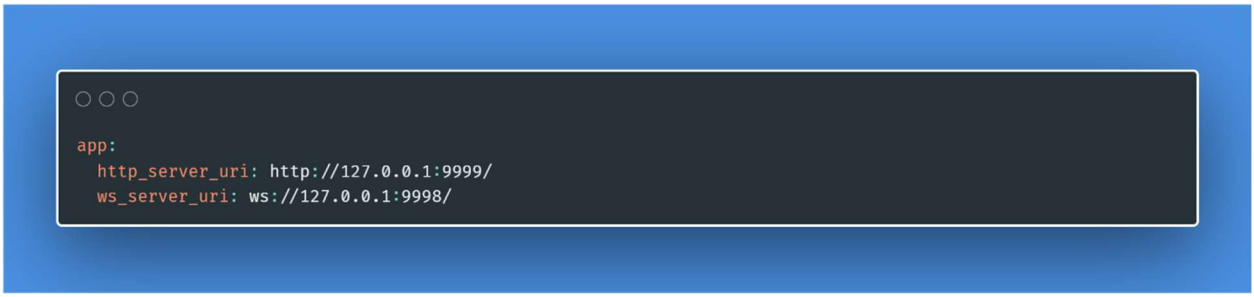
Here is an example log file. Each line contains the following elements:

- Executing thread ID.
- Timestamp.
- Severity.
- Message.
- Logger source.



Status Server

Metrics about the running DIME instance are available over an HTTP and Websocket servers. The `app` section of the `main.yaml` allows you to configure these endpoints.



By default, these endpoints are only available to the `localhost`. You can make the servers available externally by modifying the configuration.

```
app:
  http_server_uri: http://*:9999/
  ws_server_uri: ws://0.0.0.0:9998/
```

VIEWING METRICS

While a DIME instance is running, open your browser to http://localhost:9999/status. Here you can view various metrics about each source and sink connector.

A live feed of the metrics is available on the Websocket endpoint.

This helps you understand how many messages passed through each connector and the execution times of the connector.

CONFIGURATION MANAGEMENT

The current running configuration can be retrieved by opening your browser to http://localhost:9999/config/yaml.

Uploading a new configuration is possible as well. You can HTTP POST your new YAML configuration to http://localhost:9999/config/yaml. A service restart is required for the new configuration to take effect. The services can be restarted by opening your browser to http://localhost:9999/service/restart. Alternatively, you can close and reopen your DIME instance.

Individual connectors can be stopped and started by visiting http://localhost:9999/collectors/stop/{connectorName} and http://localhost:9999/collectors/start/{connectorName}, respectively.

CONFIGURATION

Configuration files are stored in the `./Configs` solution folder. On DIME startup, all `*.yaml` files are read from the folder, including `main.yaml`. The entire configuration can be stored as `main.yaml`, or can be split across multiple files, but `main.yaml` must exist as it is the last read that can contain YAML references to other files. There are three sections in a configuration file: `app`, `sinks`, and `sources`.

App Configuration Section

The `app` configuration section contains basic runtime information.

- `license` – Application license key. DIME will run for 120 minutes without a valid license.
- `ring_buffer` – Message buffer size between sources and sinks.
- `http_server_uri` – Status server HTTP endpoint. Set to http://0.0.0.0:9999/ to make available externally.
- `ws_server_uri` – Status server Websocket endpoint. Set to ws://0.0.0.0:9998/ to make available externally.

Sinks Configuration Section

The `sinks` configuration section is a list of configured sinks. Each sink contains information required for execution.

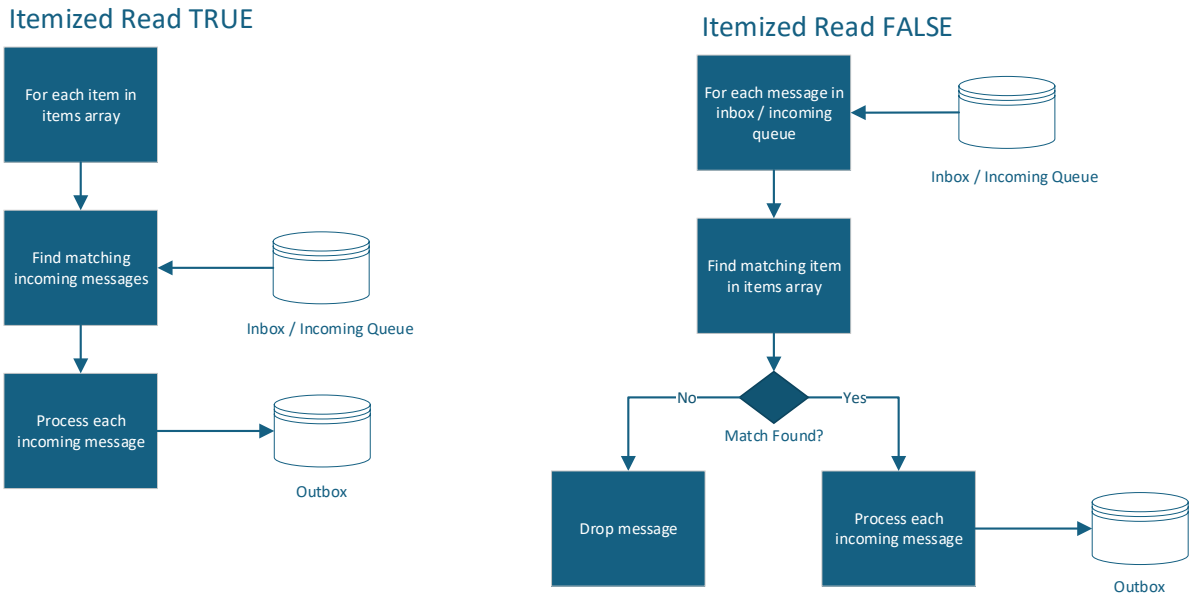
- `name` – Unique connector name. The connector name must be unique across the entire configuration.
- `enabled` – Is the connector enabled to run?
- `connector` – Connector type, see the Reference section for individual connector types.
- `scan_interval` – Message scanning frequency in milliseconds. When a scan interval is too small, you will be warned about execution overruns in the logs.
- `exclude_filter` – A list of message paths to exclude from output. Message paths are regular expressions in the format of: `connectorName/itemName` or `connectorName/\$SYSTEM/systemProperty`.
- `include_filter` – A list of regular expression message paths to include in output. Inclusion filter takes precedence over the exclusion filter.
- `use_sink_transform` – Should the sink transform defined on source connector be executed?

Sources Configuration Section

The `sources` configuration section is a list of configured sources.

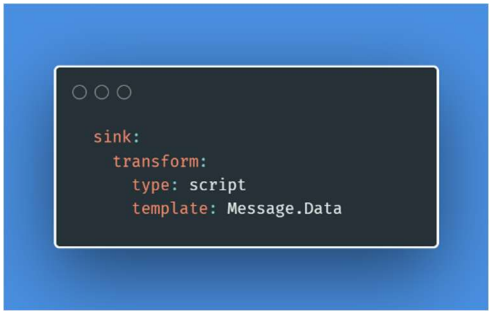
- `name` – Unique connector name. The connector name must be unique across the entire configuration.

- `enabled` – Is the connector enabled to run?
- `connector` – Connector type, see the Reference section for individual connector types.
- `scan_interval` - Message scanning frequency in milliseconds. When a scan interval is too small, you will be warned about execution overruns in the logs.
- `rbe` – Report data by exception. Only sends data to sink when the value changes from the previous value.
- `itemized_read` – Process incoming data based on items array. Itemized read is relevant to queuing source connectors only. When set to true, all connector items are iterated and a matching incoming message is found in the inbox/incoming buffer. Unmatched inbox messages are dropped. When set to false, the inbox/incoming buffer is iterated and a matching item is found in the items array. If no item is found, the raw incoming message value is passed to the outbox.



- `lang_script` – Python or Lua scripting language selection.
- `paths_script` – Additional absolute or relative paths for external script locations.
- `init_script` – Connector startup script.
- `deinit_script` – Connector shutdown script.
- `enter_script` – Connector execution loop entry script.
- `exit_script` – Connector execution loop exit script.
- `item_script` – Script executed for each item when undefined at item level.
- `strip_path_prefix` – Remove connector name from message path placed in the outbox. For example if the full message path is `connectorName/itemName`, the message path will be shortened to `itemName`.

- `create_dummy_message_on_startup` – Create a zero value message for each item on connector startup. This is helpful for queuing source connectors to make the sink aware of all possible message paths.
- `ignore_errors_on_read` – Ignore exceptions on read and continue reading remaining items.
- `wait_for_connectors` - Block execution until the specified dependency connectors complete their scan cycle, ensuring cache() access to fresh data and preventing race conditions when connectors run at the same interval.
- `sink` – Sink metadata contains information necessary to support the sink `use_sink_transform` property. When combined with `use_sink_transform: true`, the `transform template` will execute for every message in the outbox, resulting in varying output to the sink. Some sinks, such as Redis, require the message to contain the path, value, and timestamp. Other sinks, such as MTConnect, require the message to contain only the data since the message is handled differently.



Example message without sink transform:

```
Path: haas1/ActiveIconsList,
Message: {"Path":"haas1/Icons","Data":"IDLE","Timestamp":1740334909}
```

Example message with sink transform:

```
Path: haas1/ActiveIconsList,
Message: "IDLE"
```

ITEMS

The items list contains individual items to read and process from the data source.

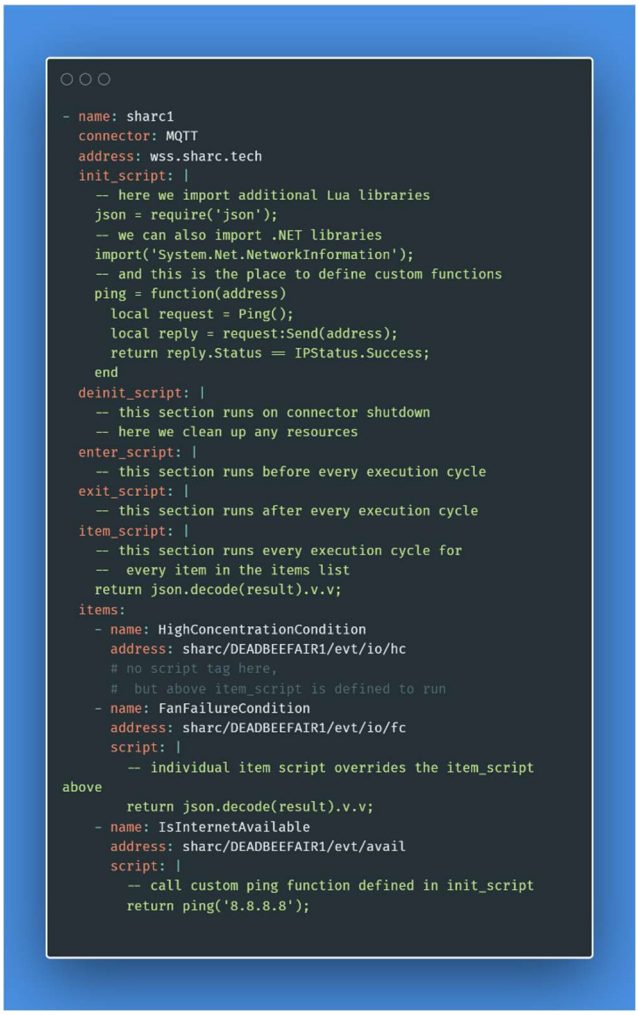
- `name` – Unique item name.
- `enabled` – Is item enabled?
- `rbe` – Report data by exception override at item level.
- `every` – Process this item every X occurences of the connector's scan interval.
- `address` – Source data address. Formatting is specific to connector type.
- `script` – Lua script executed after data source is read. The `result` variable contains the read content.
- `sink` – Sink metadata. See the connector sink property. Support for MTConnect DataItem paths is handled through individual item's sink property. Here is an example.

```yaml
items:
- name: CPU
  enabled: !!bool true
  address: CPU
  sink:
    mtconnect: Device[name=device1]/Controller/Load[category=Sample]
```

SCRIPTING

Lua or Python scripting languages can be used throughout the configuration. Each configuration's scriptable properties include:

- `init_script` – Runs on connector startup. Here is where we want to import any additional Lua or .NET libraries, define any custom functions, or create persistent variables.
- `deinit_script` – Runs on connector shutdown. Here is where we clean up any resources before shutdown.
- `enter_script` – Runs before every execution cycle.
- `exit_script` – Runs after every execution cycle.
- `item_script` – Runs every execution for every item in the `items` list, unless the `item` has a `script` property defined.
- `items[].script` – Runs every execution for that `item`. This property takes precedence over the top level `item_script`.



```
- name: sharc1
  connector: MQTT
  address: wss.sharc.tech
  init_script: |
    -- here we import additional Lua libraries
    json = require('json');
    -- we can also import .NET libraries
    import('System.Net.NetworkInformation');
    -- and this is the place to define custom functions
    ping = function(address)
      local request = Ping();
      local reply = request:Send(address);
      return reply.Status == IPStatus.Success;
    end
  deinit_script: |
    -- this section runs on connector shutdown
    -- here we clean up any resources
  enter_script: |
    -- this section runs before every execution cycle
  exit_script: |
    -- this section runs after every execution cycle
  item_script: |
    -- this section runs every execution cycle for
    --  every item in the items list
    return json.decode(result).v.v;
  items:
    - name: HighConcentrationCondition
      address: sharc/DEADBEEFAIR1/evt/io/hc
      # no script tag here,
      #  but above item_script is defined to run
    - name: FanFailureCondition
      address: sharc/DEADBEEFAIR1/evt/io/fc
      script: |
        -- individual item script overrides the item_script
above
        return json.decode(result).v.v;
    - name: IsInternetAvailable
      address: sharc/DEADBEEFAIR1/evt/avail
      script: |
        -- call custom ping function defined in init_script
        return ping('8.8.8.8');
```

Various Lua libraries are included in the `./Lua` solution folder.

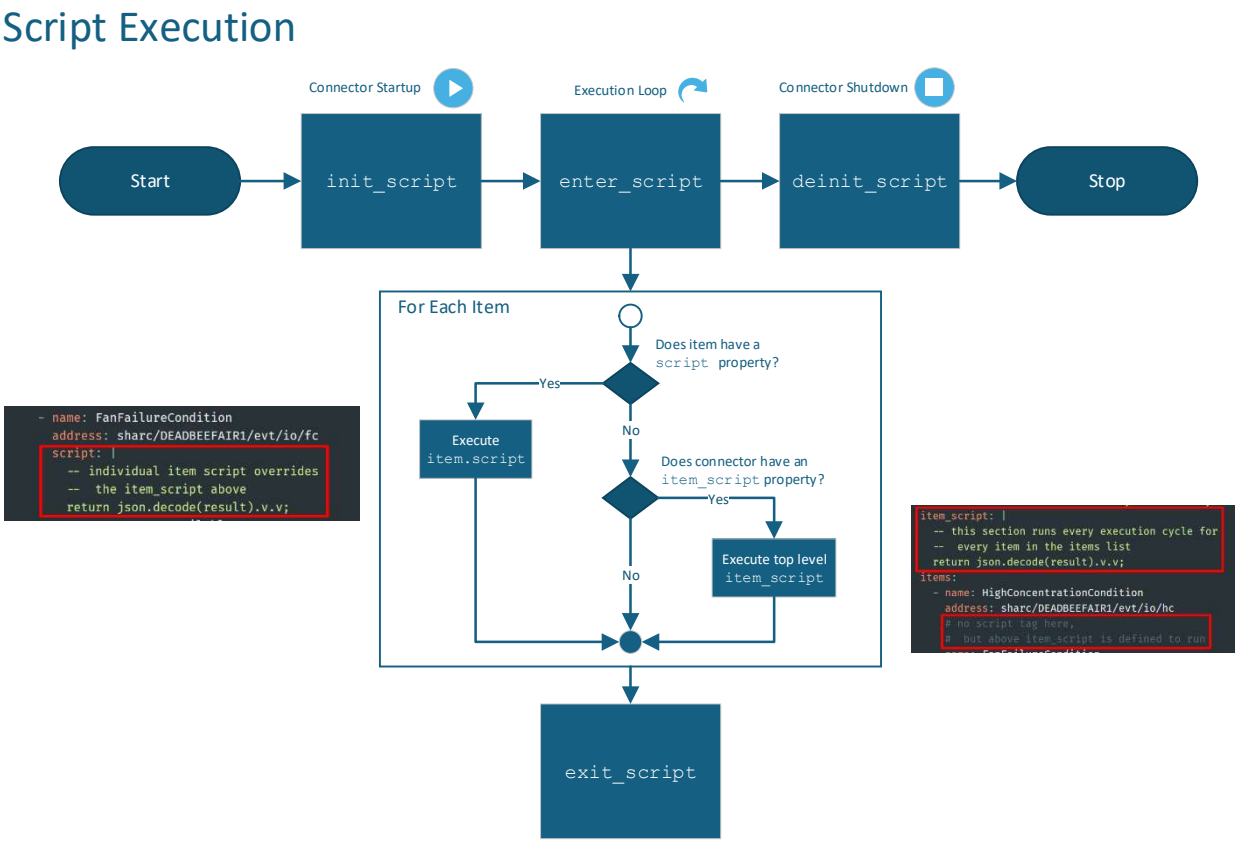
- json.lua - https://github.com/rxi/json.lua
- struct.lua - https://github.com/iryont/lua-struct
- moses.lua - https://github.com/Yonaba/Moses
- penlight - https://github.com/lunarmodules/Penlight

Python includes all standard 3.4 libraries.

In addition, because Lua and Python are embedded languages, the full .NET framework is also available to Lua (https://github.com/NLua/NLua) and Python (https://ironpython.net/documentation/dotnet) scripts.

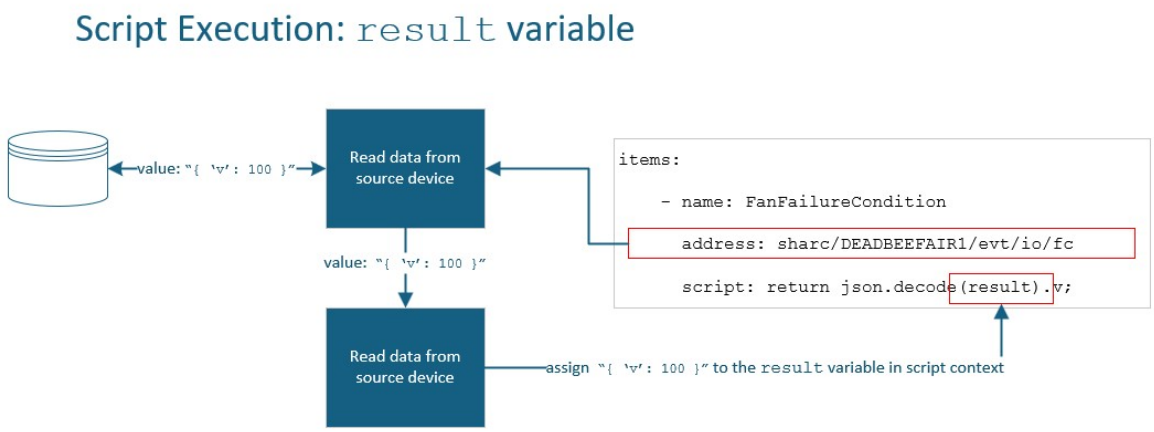
Script Execution Flow

Below diagram illustrates the script execution flow.



The Result Variable

When a source device is read, the value returned from the device becomes available to the script context as the `result` variable.



Caches

When a value is read from the source device it is stored in cache. This means that the latest value can be recalled by its path within a script.

In the example on the right, we are adding two numbers together and then deciding if that number is big or not. `Value2` call to `set(name, value)` puts the value into user cache. Returning `nil` means that the result of `value2` item will be dropped before it reaches a sink. The `sum` item recalls both `value1` and `value2` items and adds them together by calling `cache(path, defaultValue)`. Notice that there are multiple ways to address a cached item:

- Absolute path – Name of the connector followed by the item name (e.g. `script1/value1`). This means that you can access values from other connectors as well.
- Relative path – Period, referring to the current connector, followed by the item name (e.g. `./value1`).
- Shorthand – Just the item name (e.g. `value1`).

```
- name: script1
  connector: Script
  items:
    - name: value1
      script: |
        return math.random(10);
    - name: value2
      script: |
        val = math.random(100);
        set('value2', val);
        return nil;
    - name: sum
      script: |
        return cache('script1/value1', 0) +
               cache('value2', 0);
    - name: isBigNumber
      script: |
        if cache('./sum', 0) > 100 then
            return true
        else
            return false
        end
```

This Variable

Variable `this` has a different meaning in different contexts. In the `init_script`, `deinit_script`, `enter_script`, `exit_script`, `this` is a reference to the connector instance. In the `item_script` and item's `script`, `this` is a reference to the item instance.

Available Functions

Retrieve value from caches. Caches are shared across connectors.

```
Lua> value = cache(path, defaultValue)
Python> value = dime.cache(path, defaultValue)
```

Retrieve value and timestamp from caches.

```
Lua> value, timestamp = cache_ts(path, defaultValue)
Python> value, timestamp = dime.cache_ts(path, defaultValue)
```

Set value into user cache.

```
Lua> value = set(key, value)
Python> value = dime.set(key, value)
```

Retrieve environment variable.

```
Lua> value = env(name, defaultValue)
Python> value = dime.env(name, defaultValue)
```

Create a new sample.

```
Lua> value = emit(path, value)
Python> value = dime.emit(path, value)
```

Create a new sample with MTConnect sink metadata.

```
Lua> value = emit_mtconnect(path, value, mtcPath)
Python> value = dime.emit_mtconnect(path, value, mtcPath)
```

Deserialize JSON string.

```
Lua> object = from_string(string)
Python> object = dime.from_string(string)
```

Serialize object to JSON string.

```
Lua> string = to_string(object)
Python> string = dime.to_string(object)
```

Retrieve the connector instance.

```
Lua> connector = connector()
Python> connector = dime.connector()
```

Retrieve the connector's configuration instance.

```
Lua> configuration = configuration()
Python> configuration = dime.configuration()
```

Convert Lua table to a .NET array.

```
Lua> value = array_from_table(table)
```

EXERCISES

Writing to console and log

Let's start with something simple. We will use a Script source to generate random numbers and start with a single file configuration.

```
# main.yaml

console: &console
  name: console
  connector: Console

log: &log
  name: log
  connector: Logger

script: &script
  name: script
  connector: Script
  items:
    - name: number1
      script: return math.random(100);
    - name: number2
      script: return math.random(200);

app:
  license: AAAA-BBBB-CCCC-DDDD-EEEE-FFFF-GGGG-HHHH
  ring_buffer: !!int 4096
  http_server_uri: http://127.0.0.1:9999/
  ws_server_uri: ws://127.0.0.1:9998/
sinks:
  - *console
  - *log
sources:
  - *script
```

Now let's disable the Logger sink and exclude system messages from the output.

```
# main.yaml

console: &console
  name: console
  connector: Console
  exclude_filter:
    - script/\$SYSTEM

log: &log
  name: log
  connector: Logger
  enabled: !!bool false

script: &script
  name: script
  connector: Script
  items:
    - name: number1
      script: return math.random(100);
    - name: number2
      script: return math.random(200);

app:
  license: AAAA-BBBB-CCCC-DDDD-EEEE-FFFF-GGGG-HHHH
```

```
  ring_buffer: !!int 4096
  http_server_uri: http://127.0.0.1:9999/
  ws_server_uri: ws://127.0.0.1:9998/
sinks:
  - *console
  - *log
sources:
  - *script
```

Split configuration across multiple files.

Splitting your configuration across multiple files makes it more modular and manageable. Let's take the above configuration and create multiple files.

```
# main.yaml

app:
  license: AAAA-BBBB-CCCC-DDDD-EEEE-FFFF-GGGG-HHHH
  ring_buffer: !!int 4096
  http_server_uri: http://127.0.0.1:9999/
  ws_server_uri: ws://127.0.0.1:9998/
sinks:
  - *console
sources:
  - *script
```

```
# console.yaml

console: &console
  name: console
  connector: Console
  exclude_filter:
    - script/\$SYSTEM
```

```
# script.yaml

script: &script
  name: script
  connector: Script
  items:
    - name: number1
      script: return math.random(100);
    - name: number2
      script: return math.random(200);
```

Write data to an embedded MTConnect Agent.

Let's add a MTConnectAgent sink to our configuration.

```
# agent.yaml

agent: &agent
  name: agent
  connector: MTConnectAgent
  port: !!int 5000
  exclude_filter:
    - script/\$SYSTEM
```

Run DIME and navigate to http://localhost:5000/current. You will notice that none of our custom data items are there. We must mark up our sources with MTConnect syntax.

```
# script.yaml
```

```
script: &script
  name: script
  connector: Script
  items:
    - name: number1
      script: return math.random(100);
      sink:
        mtconnect: Device[name=device1]/Controller/Mass[category=Sample]
    - name: number2
      script: return math.random(200);
      sink:
        mtconnect: Device[name=device1]/Controller/Load[category=Sample]
```

Add another source.

Let's bring in data from a Haas controller and merge it with our Script source.

```
# haas.yaml
haas: &haas
  name: haas
  connector: TcpASCII
  address: 192.168.111.216
  port: !!int 5051
  read_delay: !!int 500
  init_script: |
    stringx = require('pl.stringx');

    clean_response = function(response)
      response = stringx.replace(response, '>', '');
      response = stringx.replace(response, '\r\n', '');
      return response;
    end

    get_value = function(response)
      response = stringx.split(response, ',');
      response = stringx.strip(response[2]);
      return response;
    end
  item_script: |
    local response = clean_response(result);
    --print('clean: ' .. response);
    local value = get_value(response);
    --print('value: ' .. value);
    return value;
  items:
    - name: PartCount1
      address: ?Q402
      sink:
        mtconnect: Device[name=device1]/Controller/PartCount[category=Event,subType=All]
```

Run DIME and navigate to http://localhost:5000/current.

Write data to an external MTConnect Agent.

Let's add a MTConnectSHDR sink to our configuration.

```
# shdr.yaml

shdr: &shdr
  name: shdr
  connector: MTConnectSHDR
  port: !!int 7878
  device_key: ~
```

```
  heartbeat_interval: !!int 10000
  filter_duplicates: !!bool true
```

Run DIME and `telnet localhost 7878`. You will notice that the system messages from both haas and script sources are being output. Let's exclude those.

```
# shdr.yaml
shdr: &shdr
  name: shdr
  connector: MTConnectSHDR
  port: !!int 7878
  device_key: ~
  heartbeat_interval: !!int 10000
  filter_duplicates: !!bool true
  exclusion_filter:
    - script/\$SYS
    - haas/\$SYSTEM
```

Emit messages.

Sometimes querying a single piece of data can result in multiple observations. For this we can use the `emit` and `emit_mtconnect` scripting functions.

```
# haas.yaml

haas: &haas
  name: haas
  connector: TcpASCII
  address: 192.168.111.216
  port: !!int 5051
  read_delay: !!int 500
  init_script: |
    stringx = require('pl.stringx');

    clean_response = function(response)
      response = stringx.replace(response, '>', '');
      response = stringx.replace(response, '\r\n', '');
      return response;
    end

    get_value = function(response)
      response = stringx.split(response, ',');
      response = stringx.strip(response[2]);
      return response;
    end
  item_script: |
    local response = clean_response(result);
    --print('clean: ' .. response);
    local value = get_value(response);
    --print('value: ' .. value);
    return value;
  items:
    - name: PartCount1
      address: ?Q402
      sink:
        mtconnect: Device[name=device1]/Controller/PartCount[category=Event,subType=All]
    - name: ThreeInOne
      address: ?Q500
      script: |
        result = clean_response(result);
        result = stringx.split(result, ',');
        --emit('./Program', stringx.strip(result[2]));
        emit_mtconnect('./Program', stringx.strip(result[2]),
'Device[name=device1]/Controller/Program[category=Event,subType=Main]');
        --emit('./Status', stringx.strip(result[3]));
        emit_mtconnect('./Status', stringx.strip(result[3]),
```

```
'Device[name=device1]/Controller/Execution[category=Event]');
        --emit('./PartCount3', stringx.strip(result[5]));
        emit_mtconnect('./PartCount3', stringx.strip(result[5]),
'Device[name=device1]/Controller/PartCount[category=Event,subType=Good]');
        return nil;
```

Run DIME and navigate to http://localhost:5000/current.

Add MTConnect Availability.

One thing we missed is to set the Availability of our MTConnect devices. For this we will use one of the system messages, `$SYSTEM/IsConnected`. For brevity, we will only modify the script source.

```
# script.yaml

script: &script
  name: script
  connector: Script
  items:
    - name: availability
      script:
        this_avail = cache('./$SYSTEM/IsConnected', false) and 'AVAILABLE' or 'UNAVAILABLE';
        that_avail = cache('haas/$SYSTEM/IsConnected', false) and 'AVAILABLE' or 'UNAVAILABLE';
        emit_mtconnect('./avail1', this_avail,
'Device[name=device1]/Availability[category=Event]');
        emit_mtconnect('./avail2', that_avail,
'Device[name=device2]/Availability[category=Event]');
    - name: number1
      script: return math.random(100);
      sink:
        mtconnect: Device[name=device1]/Controller/Mass[category=Sample]
    - name: number2
      script: return math.random(200);
      sink:
        mtconnect: Device[name=device1]/Controller/Load[category=Sample]
```

Add an extended MTConnect data item.

Sometimes the observation we want to represent is not part of the MTConnect standard. For this we will use the `x:` namespace.

```
# script.yaml (partial)

- name: number2
  script: return math.random(200);
  sink:
    mtconnect: Device[name=device1]/Controller/x:Cpu[category=Sample]
```

Add data from Haas MTConnect Agent.

We are currently collecting data from our asset using Haas Q commands. New generation controllers also expose a pseudo-MTConnect Agent on port 8082. Let's use a XML Web Scraper connector to grab its data.

```
# haas2.yaml

haas2: &haas2
  name: haas2
  scan_interval: !!int 5000
  connector: XMLWebScraper
  uri: http://192.168.111.216:8082/current
```

```
  namespaces:
    mt: urn:mtconnect.org:MTConnectStreams:1.2
  item_script: |
    return result.InnerText
  items:
    - name: SpindleLoad
      address: //mt:Message[@dataItemId='sp2maxpwr']
      sink:
        mtconnect: Device[name=device1]/Controller/Load[category=Sample]
```

Add data from an ASC Autoclave

Create the TCPListener object in your CPC instance per https://www.aschome.com/administrator/images/support/pdf/CPC%20-%20TCP%20Talker%20and%20Listener.pdf.

```
# asc.yaml

ascCpcSource1: &ascCpcSource1
  name: autoclave1
  enabled: !!bool true
  scan_interval: !!int 2500
  connector: AscCPC
  address: 192.168.111.12
  port: !!int 9999
  bypass_ping: !!bool true
  items:
    - name: Availability
      script: |
        local n = cache('./$SYSTEM/IsConnected', nil);
        return n and 'Available' or 'Unavailable';
      sink:
        mtconnect: Device[Name=autoclave1]/Availability[Category=Event]
    - name: Temperature
      address: .Autoclave.Inputs.AIRTC\Value
      sink:
        mtconnect: Device[Name=autoclave1]/Controller/Path/Temperature[Category=Sample]
    - name: Pressure
      address: .Autoclave.Inputs.PRESS\Value
      sink:
        mtconnect: Device[Name=autoclave1]/Controller/Path/Pressure[Category=Sample]
    - name: PumpHours
      address: .Autoclave.Scripts.MotorHours.CoolPumpAOn\Value
      sink:
        mtconnect: Device[Name=autoclave1]/Controller/Path/Hours[Category=Sample]
    - name: UserOperator
      address: .Autoclave.Variables.OperatorName\Value
    - name: Program
      address: .Autoclave.RecipeProcessor.Recipe.RecipeData\Description
    - name: ControlPowerConditionTriggered
      address: .Autoclave.Alarms.ControlPower\Condition
      script: |
        set('cptrig', result);
        return nil;
    - name: ControlPowerAlarmActive
      address: .Autoclave.Alarms.ControlPower\AlarmActive
      script: |
        set('cpalm', result);
        return nil;
    - name: ControlPowerCondition
      address: ~
      script: |
        local cptrig = cache('cptrig', false);
        local cpalm = cache('cpalm', false);
        return cpalm and 'Fault' or (cptrig and 'Warning' or 'Normal');
```

REFERENCE

Connector Configuration Layout

COMMON LAYOUT - SOURCE CONNECTOR

Name	Type	Default Value	Description
name	string	"Unnamed"	Unique connector name.
enabled	boolean	TRUE	Is connector enabled.
connector	string	"Undefined"	Connector type.
scan_interval	int	1000	Scanning frequency in milliseconds.
rbe	boolean	TRUE	Report data by exception, or unique value.
itemized_read	boolean	FALSE	Process incoming data based on items array.
lang_script	string	Lua	Python or Lua scripting language.
paths_script	string[]	Empty	Absolute or relative additional paths to use for scripting libraries.
init_script	string	Empty	Startup script.
deinit_script	string	Empty	Shutdown script.
enter_script	string	Empty	Execution loop entry script.
exit_script	string	Empty	Execution loop exit script.
item_script	string	Empty	Script executed for each item when undefined at item level.
sink	dictionary	Empty	Sink metadata.
strip_path_prefix	boolean	FALSE	Remove connector name from messages placed in outbox.
create_dummy_messages_on_startup	boolean	FALSE	For itemized_read connectors, create a zero value message for each item on connector startup.
ignore_errors_on_read	boolean	FALSE	Ignore exceptions on read and continue reading remaining items.
wait_for_connectors	list	Empty	Block execution until the specified dependency connectors complete their scan cycle, ensuring cache() access to fresh data and preventing race conditions when connectors run at the same interval.
items	list	Empty	List of items to read from the source.
items.name	string	"Unnamed"	Unique item name.
items.enabled	boolean	TRUE	Is item enabled.
items.rbe	boolean	TRUE	Report data by exception override at item level.
items.every	int	1	Execute item every x scan_interval.
items.address	string	Empty	Source data address, formatting specific to connector type.
items.script	string	Empty	Lua script executed after source data is read.

items.sink	dictionary	Empty	Sink metadata override at item level.

References

“A Gentle Introduction to the YAML Format”: https://dev.to/kalkwst/a-gentle-introduction-to-the-yaml-format-bi6

Source Example

```
- name: script1
  enabled: !!bool true
  connector: ActiveMQ
  scan_interval: !!int 1000
  rbe: !!bool true
  itemized_read: !!bool true
  init_script: |
    print('hello from Lua');
  deinit_script: ~
  enter_script: print('entering loop');
  exit_script: print('exiting loop');
  sink:
    transform:
      type: script
      template: Message.Data
  strip_path_prefix: !!bool false
  create_dummy_messages_on_startup: !!bool false
  items:
    - name: randomNumber1
      enabled: !!bool true
      rbe: !!bool true
      every: !!int 1
      script: return math.random(10);
```

COMMON LAYOUT - SINK CONNECTOR

Name	Type	Default Value	Description
name	string	"Unnamed"	Unique connector name.
enabled	boolean	TRUE	Is connector enabled.
connector	string	"Undefined"	Connector type.
scan_interval	int	1000	Scanning frequency in milliseconds.
exclude_filter	list	Empty	Message path exclusion filter.
include_filter	list	Empty	Message path inclusion filter.
use_sink_transform	boolean	FALSE	Execute transform defined on the source connector.

Sink Example

```
- name: console1
  enabled: !!bool true
  connector: Console
  scan_interval: !!int 1000
  exclude_filter:
    - script1/\$SYSTEM
  #include_filter:
  # - script1/randomNumber1
  use_sink_transform: !!bool true
```

ACTIVEMQ

Subscribes to ActiveMQ topics and queues.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "ActiveMQ".
address	string	Empty	Broker URI.
username	string	Empty	Broker username.
password	string	Empty	Broker password.
itemized_read	bool	FALSE	When TRUE, streaming data from broker is matched against the 'items' list and processed, unmatched 'item.address' are dropped and not forwarded to sinks. When FALSE, data streaming from broker is matched against the 'items' list and processed, unmatched 'item.address' are forwarded to sinks.

References

ActiveMQ NMS OpenWire Github: https://github.com/apache/activemq-nms-openwire

Source Example

```
- name: amq
  connector: ActiveMQ
  address: activemq:tcp://172.24.56.104:61616
  username: artemis
  password: artemis
  items:
    - name: FooBar
      address: topic://FOO.BAR
    - name: BarFoo
      address: queue://BAR.FOO
```

ASC CPC

Reads data from ASC autoclave systems.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "AscCPC".
address	string	Empty	CPC instance hostname or IP address.
port	string	9999	CPC instance port.
bypass_ping	bool	FALSE	Ping CPC host before reading to determine connector connection status.

References

"CPC – TCP Talker and Listener": https://www.aschome.com/administrator/images/support/pdf/CPC%20-%20TCP%20Talker%20and%20Listener.pdf

Source Example

```
- name: ascCpcSource1
  connector: AscCPC
  address: 192.168.111.12
  port: !!int 9999
  bypass_ping: !!bool true
  items:
    - name: Temperature
      address: .Autoclave.Inputs.AIRTC\Value
```

BECKHOFF ADS

Reads data from Beckhoff PLCs.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "BeckhoffADS".
local_netid	string	Empty	Local AMS Net ID.
address	string	Empty	Remote AMS Net ID.
target_ip	string	Empty	IPv4 address to remote AMS Net ID.
port	int	851	ADS port.
items.type	string	Empty	PLC register type ('bool', 'sbyte', 'short', 'int', 'long', 'float', 'string').

References

TwinCAT 3 | ADS Basics: https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_ads_intro/index.html&id=

Source Example

```
- name: ads1
  connector: BeckhoffADS
  local_netid: 1.1.1.1.1.1
  target_ip: 192.168.111.191
  address: 192.168.111.191.1.1
  port: !!int 851
  items:
    - name: boolTag1
      type: bool
      address: MAIN.someBool
    - name: intTag2
      type: int
      address: MAIN.someInt
```

CLAUDE CODE

Queries Claude AI with a custom prompt and data set.

(Alpha Preview)

CONSOLE

Writes data to the console.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "Console".

Sink Example

```
- name: consoleSink1
  connector: Console
```

CSV WRITER

Writes data to a CSV file.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "CSVWriter".
filename	string	Empty	Absolute or relative file path and file name to write to.
filter_duplicate_paths	bool	FALSE	Filter paths with same name, outputting only one instance of each path.

Sink Example

```
- name: csvSink1
  connector: CSVWriter
  filename: ./Output/airsharc2.csv
  filter_duplicate_paths: !!bool false
```

ETHERNET/IP

Reads data from Allen-Bradley PLCs.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "EthernetIP".
type	string	ControlLogix	PLC type ('ControlLogix', 'Plc5', 'Slc500', 'LogixPccc', 'Micro800', 'MicroLogix', 'Omron')
address	string	Empty	PLC hostname, IP address.
path	string	1,0	Connection path.
log	int	0	Library log level (0: None ... 5: Verbose)
timeout	int	1000	PLC read timeout in milliseconds.
bypass_ping	bool	FALSE	Ping PLC before reading to determine connector connection status.
items.type	string	Empty	PLC register type ('bool', 'ubyte', 'byte', 'ushort', 'short', 'uint', 'int', 'ulong', 'long', 'float', 'double', 'string').

References

Libplctag.NET Github: https://github.com/libplctag/libplctag.NET

Source Example

```
- name: plcSource1
  connector: EthernetIP
  type: MicroLogix
  address: 192.168.111.20
  path: 1,0
  log: !!int 0
  timeout: !!int 1000
  bypass_ping: !!bool true
  items:
    - name: boolTag1
      type: bool
      address: B3:0/2
    - name: intTag2
      type: int
      address: N7:1
```

FANUC ROBOT

Reads R-J3iB, R-30iA, R-30iB Fanuc Robot controllers using SNPX.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "FanucRobot".
address	string	Empty	Robot hostname, IP address.
items.address	string	Empty	Variable to read.

Variable Addresses.

Address	Address	Address	Address
worldCartesianPosition.R	worldJointPosition.J1	AI.{index}	WSI.{index}
worldCartesianPosition.P	worldJointPosition.J2	AO.{index}	PMC_K.{index}
worldCartesianPosition.W	worldJointPosition.J3	GI.{index}	PMC_D.{index}
worldCartesianPosition.X	worldJointPosition.J4	GO.{index}	PMC_R.{index}
worldCartesianPosition.Y	worldJointPosition.J5	SI.{index}	StringSystemVariables.{index}
worldCartesianPosition.Z	worldJointPosition.J6	SO.{index}	IntegerSystemVariables.{index}
worldCartesianPosition.E1	worldJointPosition.J7	UI.{index}	PositionSystemVariables.{index}
worldCartesianPosition.E2	worldJointPosition.J8	UO.{index}	NumericRegisters.{index}
worldCartesianPosition.E3	worldJointPosition.J9	RDI.{index}	PositionRegisters.{index}
worldCartesianPosition.T4		RDO.{index}	StringRegisters.{index}
worldCartesianPosition.T5		SDI.{index}	
worldCartesianPosition.T6		SDO.{index}	

Source Example

```
- name: fanuc1
  connector: FanucRobot
  address: 192.168.111.20
  items:
    - name: UI1
      address: UI.1
```

HAAS SHDR

Receives SHDR-like steaming data from a Haas controller over an undocumented port.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "HaasSHDR".
itemized_read	bool	FALSE	When TRUE, streaming data from controller is matched against the 'items' list and processed, unmatched 'item.address' are dropped and not forwarded to sinks. When FALSE, data streaming from controller is matched against the 'items' list and processed, unmatched 'item.address' are forwarded to sinks.
address	string	Empty	Controller IP hostname or address.
port	int	9998	Controller port number.
timeout	int	1000	Connection timeout in milliseconds.
heartbeat_interval	int	4000	Heartbeat frequency in milliseconds.
retry_interval	int	10000	Retry frequency in milliseconds.

Notes

This source connector is experimental and communicates with the Haas controller using an undocumented port. It is known that in newer releases of the NGC software this port has been closed, in which case the Haas Q or XML Web Scraper source connectors should be used. There is also a known issue where the controller stops streaming data and the NGC must be restarted.

Source Example

```
- name: haasSource1
  connector: HaasSHDR
  itemized_read: !!bool true
  address: 192.168.111.221
  port: !!int 9998
  timeout: !!int 1000
  heartbeat_interval: !!int 4000
  retry_interval: !!int 10000
  items:
    - name: CPU
      enabled: !!bool true
      address: CPU
      script: |
        if tonumber(result) > 0.5 then
          return 'HIGH';
        else
          return 'LOW';
        end
```

HTTP CLIENT

Posts data to an HTTP server.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "HTTPClient".
uri	string	http://localhost/	Resource URL to POST JSON payload to.
headers	dictionary	Empty	Dictionary of request headers.

References

HTTP POST: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/POST

Sink Example

```
- name: httpClientSink1
  connector: HttpClient
  uri: https://webhook.site/0e10dc3d-6bec-45fa-952d-dba785bc3109
  headers:
    Content-Type: text/plain;
    Authorization: None
```

HTTP SERVER – SOURCE

Starts an HTTP server and listens to HTTP POST requests. Posted data is matched to individual items by the relative path where it was posted.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "HTTPServer".
uri	string	http://localhost:8081/	URL to listen for POST requests.
items.address	string	Empty	URI path where to get POST data from.

Notes

You can specify a less restrictive URI by providing the system's IP address or listen on all adapters by specifying http://*:8081.

Source Example

```
- name: httpServerSource1
  connector: HTTPServer
  uri: http://localhost:8081/
  init_script: |
    json = require('json');
  items:
    - name: postData
      address: post/data
      script: |
        return json.decode(result).hello;
```

HTTP SERVER – SINK

Starts an HTTP server and serves all data items to external HTTP clients.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "HTTPServer".
uri	string	http://localhost:8080/	URL to serve items.

Notes

You can specify a less restrictive URI by providing the system's IP address or listen on all adapters by specifying http://*:8080.

`/items` – returns all items as a dictionary.

`/list` – returns all items as a list.

`/items/*` - returns a specific item that starts with path. Example: /items/haas1, /items/haas1/C, /items/haas1/CPU.

Sink Example

```
- name: httpServerSink1
  connector: HttpServer
  uri: http://localhost:8080/
```

I3X WEB SERVER

Presents data via a CESMII I3X compliant web server.

(Alpha Preview)

INFLUXLP

Writes data to an Influx Data bucket.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "InfluxLP".
address	string	Empty	URL to your Influx instance.
token	string	Empty	Influx token.
org_name	string	Empty	Influx organization name.
bucket_name	string	Empty	

References

InfluxDB 3 C# Client Github: https://github.com/InfluxCommunity/influxdb3-csharp

Sink Example

```
- name: influxLpSink1
  connector: InfluxLP
  address: https://us-east-1-1.aws.cloud2.influxdata.com
  token: abc123
  org_name: mriiot
  bucket_name: bucket1
```

JSONWEBSCRAPER

Reads data from an JSON web page.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "JSONWebScraper".
uri	string	http://localhost/	Web page address.
items.address	string	Empty	Jsonata query.

References

“Jsonata”: https://jsonata.org/

Source Example

```
- name: json1
  connector: JsonWebScraper
  uri: https://raw.githubusercontent.com/jpadfield/simple-site/refs/heads/master/build/mirador.json
  init_script: json = require('json')
  items:
    - name: node1
      address: $.catalog.manifestID
      script: |
        return json.decode(result)[1];
```

LOGGER

Writes data to an NLog logger.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "Logger".

Sink Example

```
- name: loggerSink1
  connector: Logger
```

MODBUSTCP

Reads data from a Modbus/TCP device.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "ModbusTCP".
address	string	Empty	Device hostname or IP address.
port	int	502	Port number.
slave	int	1	Modbus slave ID.
timeout	int	1000	Read timeout in milliseconds.
items.address	string	Empty	Register address.
items.type	string	1	Register type (1 - coil, 2 - input, 3 - holding register, 4 - input register).
items.count	int	1	Number of consecutive registers to read.

Source Example

```
- name: modbusSource1
  connector: ModbusTCP
  address: 192.168.111.20
  port: !!int 502
  slave: !!int 1
  timeout: !!int 1000
  init_script: struct = require('struct')
  items:
    - name: coilTags
      type: !!int 1
      address: !!int 1
      count: !!int 10
    - name: holdingTags
      type: !!int 3
      address: !!int 24
      count: !!int 2
      script: |
         -- https://www.scadacore.com/tools/programming-calculators/online-hex-converter/
         return struct.unpack('<I', struct.pack('<HH', result[0], result[1]));
```

MONGODB

Writes data to a MongoDB collection.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "MongoDB".
address	string	Empty	Database connection string.
database	string	Empty	Database name.
collection	string	Empty	Collection name.

References

MongoDB Driver: https://www.mongodb.com/docs/drivers/csharp/current/

Sink Example

```
- name: mongo
  connector: MongoDB
  address: mongodb+srv://user:password@cluster0.h7xod.mongodb.net/?retryWrites=true&w=majority&appName=Cluster0
  database: DIME
  collection: TS
```

MQTT – SOURCE

Subscribes to MQTT broker topics.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "MQTT".
address	string	Empty	Broker hostname or IP address.
port	int	1883	Broker port.
qos	int	0	Quality of Service (0, 1, 2).
itemized_read	bool	FALSE	When TRUE, streaming data from broker is matched against the 'items' list and processed, unmatched 'item.address' are dropped and not forwarded to sinks. When FALSE, data streaming from broker is matched against the 'items' list and processed, unmatched 'item.address' are forwarded to sinks.
clean_session	bool	TRUE	MQTT clean session.
tls	bool	FALSE	SSL/TLS connection.
tls_insecure	bool	FALSE	Allow untrusted certificates.
client_cert_path	string	Empty	Relative or absolute path to the client certificate (.pfx).
client_cert_password	string	Empty	Client certificate password.
ca_cert_path	string	Empty	Relative or absolute path to the certificate authority certificate.
username	string	Empty	Username.
password	string	Empty	Password.
items.address	string	Empty	Single or wildcard subscription topic.

References

MQTTnet Github: https://github.com/dotnet/MQTTnet

Source Example

```
- name: mqttSource1
  connector: MQTT
  address: wss.sharc.tech
  port: !!int 1883
  items:
    - name: subscribe1
      address: sharc/+/evt/#
```

MQTT – SINK

Publishes data to an MQTT broker.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "MQTT".
address	string	Empty	Broker hostname or IP address.
port	int	1883	Broker port.
qos	int	0	Quality of Service (0, 1, 2).
retain	bool	TRUE	Retain published messages.
base_topic	string	dime	Topic prefix for published messages.
clean_session	bool	TRUE	MQTT clean session.
tls	bool	FALSE	SSL/TLS connection.
tls_insecure	bool	FALSE	Allow untrusted certificates.
client_cert_path	string	Empty	Relative or absolute path to the client certificate (.pfx).
client_cert_password	string	Empty	Client certificate password.
ca_cert_path	string	Empty	Relative or absolute path to the certificate authority certificate.
username	string	Empty	Username.
password	string	Empty	Password.

References

MQTTnet Github: https://github.com/dotnet/MQTTnet

Sink Example

```
- name: mqttSink1
  connector: MQTT
  address: wss.sharc.tech
  port: !!int 1883
  base_topic: ids
  qos: !!int 0
  retain: !!bool true
```

MSSQL

Reads data from a Microsoft SQL database.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "MSSQL".
connection_string	string	Empty	Database connection string.
command_text	string	Empty	SQL query.
items.address	string	Empty	DataTable column name.

Source Example

```
- name: msSqlSource1
  connector: MSSQL
  connection_string: Server=172.16.10.5;Database=Tykma;User
Id=datareader;Password=datareader;Encrypt=True;TrustServerCertificate=True;
  command_text: select top 5 * from dbo.SiliconeRubberOrders;
  items:
    - name: OrderNumber
      address: ManufacturingOrderNumber
      script: return result[0];
    - name: OrderQuantity
      address: OrderQuantity
      script: return result[0];
```

MTCONNECT AGENT – SOURCE

Reads streaming data from an external MTConnect Agent.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "MTConnectAgent".
address	string	Empty	Agent hostname or IP address.
port	int	5000	Agent port.
device	string	Empty	Device name to query.
itemized_read	bool	FALSE	When TRUE, streaming data from agent is matched against the 'items' list and processed, unmatched 'item.address' are dropped and not forwarded to sinks. When FALSE, data streaming from agent is matched against the 'items' list and processed, unmatched 'item.address' are forwarded to sinks.
items.address	string	Empty	DataItem ID to read.

References

MTConnect.NET Github: https://github.com/TrakHound/MTConnect.NET

Source Example

```
- name: mtConnectSource1
  connector: MTConnectAgent
  address: mtconnect.mazakcorp.com
  port: !!int 5719
  device: HCN001
  interval: !!int 100
  items:
    - name: PathPositionSample
      address: pathpos
      script: |
        return result[0].Value;
```

MTCONNECT AGENT – SINK

Serve MTConnect embedded Agent.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "MTConnectAgent".
address	string	Empty	Agent hostname or IP address.
port	int	5000	Agent port.

References

MTConnect.NET Github: https://github.com/TrakHound/MTConnect.NET

Sink Example

```
- name: mtConnectSink1
  connector: MTConnectAgent
  port: !!int 5000
```

MTCONNECT SHDR

Stream SHDR to an external Agent.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "MTConnectSHDR".
port	int	7878	SHDR listening port.
device_key	string	Empty	Device key.
heartbeat_interval	int	10000	Ping/Pong frequency in milliseconds.
filter_duplicates	bool	TRUE	Filter duplicates.
output_folder	string	./Output/MTConnect	Absolute or relative folder path where to write Devices.xml.

References

MTConnect.NET Github: https://github.com/TrakHound/MTConnect.NET

Sink Example

```
- name: shdrSink1
  connector: MTConnectSHDR
  port: !!int 7878
  device_key: ~
  heartbeat_interval: !!int 10000
  filter_duplicates: !!bool true
```

MTCONNECT MQTT

Publishes MTConnect semantic payloads to an MQTT broker.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "MTConnectMQTT".
address	string	localhost	Broker hostname or IP address.
port	int	1883	Broker port.

Sink Example

```
- name: mqtt
  connector: MTConnectMqtt
  address: localhost
  port: !!int 1883
```

NWS WEATHER

Reads weather at the specified location.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "NwsWeather".
address	string	https://api.weather.gov	Server hostname or IP address.
agent	string	(MyWeatherApp, contact@example.com)	Unique user-agent.
items.address	string	Empty	Longitude and Latitude, comma separated.
items.forecast	string	daily	Daily or hourly forecast.

Source Example

```
-   name: nwsweather
    enabled: !!bool true
    scan_interval: !!int 10000
    connector: NwsWeather
    address: https://api.weather.gov
    agent: (DimeWeather, contact@dime.com)
    items:
      - name: NewYork
        address: 40.7128, -74.0060
        forecast: daily
        script: |
          print(result.properties.periods[0].temperature)
          return result.properties.periods[0].temperature;
```

OPC-DA

Reads items from an OPC-DA server.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "OpcDA".
address	string	Kepware.KEPServerEX.V6	Server node.
items.address	string	Empty	Node ID to read.

Notes

DIME 32bit build is required to use OPC-DA connector.

References

TitaniumAs.OPC.Client Github: https://github.com/titanium-as/TitaniumAS.Opc.Client

Source Example

```
- name: opcDaSource1
  connector: OpcDA
  address: Kepware.KEPServerEX.V6
  items:
    - name: DateTime
      address: _System._DateTime
```

OPC-UA - SOURCE

Reads items from an OPC-UA server.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "OpcUA".
address	string	Empty	Server hostname or IP address.
port	int	49320	Server port.
path	string	Empty	Server path.
timeout	int	1000	Timeout in milliseconds.
mode	int	None	1=None, 2=Sign, 3=SignAndEncrypt
policy	int	None	1=None, 2=Basic256, 3=Basic128Rsa15, 4=Basic256Sha256
anonymous	bool	FALSE	Connect anonymously.
username	string	Empty	Username.
password	string	Empty	Password.
allow_status_codes	list	Empty	List of allowed status codes. Codes in this list will not cause connector to throw an error.
items.address	string	Empty	Node ID to read.
items.namespace	int	2	Namespace ID to read.

References

LibUA Github: https://github.com/nauful/libua

Source Example

```
- name: opcUaSource1
  connector: OpcUA
  address: localhost
  port: !!int 49320
  timeout: !!int 1000
  anonymous: !!bool false
  username: chris
  password: passwordpassword
  allowed_status_codes:
    - 2156724224
  items:
    - name: DateTime
      namespace: !!int 2
```

```
      address: _System._DateTime
    - name: Random
      namespace: !!int 2
      address: Simulation Examples.Functions.Random6
```

OPC-UA – SINK

Writes items to an OPC-UA server.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "OpcUA".
address	string	Empty	Server hostname or IP address.
port	int	49320	Server port.
timeout	int	1000	Timeout in milliseconds.
mode	int	None	1=None, 2=Sign, 3=SignAndEncrypt
policy	int	None	1=None, 2=Basic256, 3=Basic128Rsa15, 4=Basic256Sha256
anonymous	bool	FALSE	Connect anonymously.
username	string	Empty	Username.
password	string	Empty	Password.

References

LibUA Github: https://github.com/nauful/libua

Source Example

```
- name: opcUaSource1
  connector: OpcUA
  address: localhost
  port: !!int 49320
  timeout: !!int 1000
  anonymous: !!bool false
  username: chris
  password: passwordpassword
```

POSTGRES

Reads data from a PostgreSQL database.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "Postgres".
connection_string	string	Empty	Database connection string.
command_text	string	Empty	SQL query.
items.address	string	Empty	DataTable column name.

References

Npgsql Github: https://github.com/npgsql/npgsql

Source Example

```
- name: postgresSource1
  connector: Postgres
  connection_string: Host=172.16.10.43;Port=5342;Username=postgres;Password=postgres;Database=pg;
  command_text: select * from public.fedex limit 3;
  items:
    - name: TrackingNumber
      address: package_tracking_number
      script: return result[0];
    - name: ShipToName
      address: ship_to_name
      script: return result;
```

REDIS – SOURCE

Reads data from Redis.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "Redis".
address	string	Empty	Server hostname or IP address.
port	int	6379	Server port.
database	int	0	Database ID.
items.address	string	Empty	Cache path.

References

Redis Github: https://stackexchange.github.io/StackExchange.Redis/

Source Example

```
- name: redisSink1
  connector: Redis
  address: 172.24.56.104
  port: !!int 6379
  database: !!int 0
  items:
    - name: plcGoodPartCount
      address: eipSource1/GoodPartCount
```

REDIS – SINK

Writes data to Redis.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "Redis".
address	string	Empty	Server hostname or IP address.
port	int	6379	Server port.
database	int	0	Database ID.

References

Redis Github: https://stackexchange.github.io/StackExchange.Redis/

Sink Example

```
- name: redisSink1
  connector: Redis
  address: 172.24.56.104
  port: !!int 6379
  database: !!int 0
```

ROS2

Subscribes to ROS2 topics.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "ROS2".
message_libraries	string[]	Empty	Array of relative paths to ROS2 message structure DLLs.
items.address	string	Empty	Topic.
items.type	string	Empty	Qualified type and assembly of the ROS2 message structure type.

Source Example

```
- name: ros2
  connector: Ros2
  message_libraries:
    - ./Configs/Ros2cs-ros2-sensor_msgs.dll
    - ./Configs/Ros2cs-clearpathrobotics-clearpath_msgs.dll
  items:
    - name: Chatter
      type: Ros2cs.Messages.Std.String, DIME
      address: /chatter
      script: |
        print(result)
        return result.Data
    - name: EmergencyStop
      type: Ros2cs.Messages.Std.Bool, DIME
      address: /A300_XXXXX/platform/emergency_stop
      script: return result
    - name: BatteryStatus
      enabled: !!bool true
      type: Ros2cs.Messages.Sensor.BatteryState, Ros2cs-ros2-sensor_msgs
      address: /A300_XXXXX/platform/bms/status
      script: return result
    - name: MotorStatus
      enabled: !!bool false
      type: Ros2cs.Messages.ClearpathMotor.LynxMultiStatus, Ros2cs-clearpathrobotics-clearpath_msgs
      address: /A300_XXXXX/platform/motors/status
      script: return result
```

SCRIPT

Executes arbitrary scripts.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "Script".

References

"Programming in Lua": https://www.lua.org/pil/1.html

Source Example

```
- name: scriptSource1
  connector: Script
  init_script: ~
  deinit_script: ~
  enter_script: ~
  exit_script: ~
  items:
    - name: Temperature
      script: |
        return math.random(100);
```

SIEMENS S7

Reads registers from a Siemens S7 PLC.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "SiemensS7".
type	string	S71200	PLC type ('S71200', 'S7200', 'S7300', 'S7400', 'S7200Smart', 'S71500', 'Logo0BA8').
address	string	Empty	PLC hostname or IP address.
port	int	102	PLC port.
rack	int	0	PLC rack.
slot	int	0	PLC slot.
bypass_ping	bool	FALSE	Ping PLC before reading to determine connector connection status.
items.address	string	Empty	Register address.
items.type	string	Empty	PLC register type ('bool', 'sbyte', 'short', 'int', 'long', 'float', 'string').

References

S7netplus Github: https://github.com/S7NetPlus/s7netplus

Source Example

```
- name: plcSource1
  connector: SiemensS7
  type: S71200
  address: 192.168.111.20
  port: !!int 102
  rack: !!int 0
  slow: !!int 0
  bypass_ping: !!bool true
  items:
    - name: input0
      type: bool
      address: I0.0
    - name: output0
      type: bool
      address: Q0.0
```

SNMP

Reads SNMP device.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "SNMP".
address	string	Empty	Device hostname or IP address.
port	int	161	Device port.
community	string	public	Community.
timeout	int	1000	Read timeout.
items.address	string	Empty	Item OID.

References

SNMP Library Github: https://docs.lextudio.com/sharpsnmplib/

Source Example

```
- name: snmpSource1
  connector: SNMP
  address: 192.168.150.143
  port: !!int 161
  community: public
  timeout: !!int 1000
  items:
    - name: Temperature
      address: 1.3.6.1.4.1.6574.1.2.0
    - name: Model
      address: 1.3.6.1.4.1.6574.1.5.1.0
    - name: SerialNumber
      address: 1.3.6.1.4.1.6574.1.5.2.0
```

SPARKPLUGB – SOURCE

Subscribes to a SparkplugB host.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "SparkplugB".
address	string	Empty	Host hostname or IP address.
port	int	1883	Host port.
username	string	Empty	Username.
password	string	Empty	Password.
clean_session	bool	TRUE	Clean session.
qos	int	0	Quality of service (0, 1, 2)
items.address	string	Empty	SpB topic.

Source Example

```
- name: spb
  connector: SparkplugB
  address: localhost
  port: !!int 1883
  username: user
  password: password
  clean_session: !!bool true
  qos: !!int 0
  items:
    - name: F1D1
      address: spBv1.0/Chicago/DDATA/Factory1/DIME1
```

SPARKPLUGB – SINK

Publishes data to a SparkplugB host.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "SparkplugB".
address	string	Empty	Host hostname or IP address.
port	int	1883	Host port.
username	string	Empty	Username.
password	string	Empty	Password.
host_id	string	dime	Host ID.
group_id	string	dime	Group ID.
node_id	string	dime	Node ID.
device_id	string	dime	Device ID.
reconnect_interval	int	15000	Reconnect interval in millseconds.
birth_delay	int	10000	Delay birth certificate creation in milliseconds.

References

SparkplugNet Github: https://github.com/SeppPenner/SparkplugNet

Sink Example

```
- name: sparkplugBSink1
  connector: SparkplugB
  address: localhost
  port: !!int 1883
  username: admin
  password: admin
  host_id: dime
  group_id: dime
  node_id: dime
  device_id: dime
  reconnect_interval: !!int 15000
  birth_delay: !!int 10000
```

SPLUNK EH SDK (VERSION 1.0)

Writes data to Splunk via Splunk EdgeHub SDK.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "SplunkEhSdk1".
address	string	http://host.docker.internal	Internal address.
port	int	50051	Internal port.
numbers_to_metrics	bool	FALSE	Write numbers as metrics.

Sink Example

```
- name: splunkEhSdk
  connector: SplunkEhSDK1
  address: http://host.docker.internal
  port: !!int 50051
  numbers_to_metrics: !!bool true
```

SPLUNK EH SDK (VERSION 2.0)

Writes data to Splunk via Splunk EdgeHub SDK.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "SplunkEhSdk2".
address	string	http://host.docker.internal	Internal address.
port	int	50051	Internal port.
topic	string	Empty	EdgeHub SDK ingestion topic.

Sink Example

```
- name: splunkEhSdk
  connector: SplunkEhSDK2
  address: http://host.docker.internal
  port: !!int 50051
  topic: dime/dev
```

SPLUNK HEC

Writes data to Splunk via Splunk HEC.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "SplunkHEC".
address	string	Empty	Splunk hostname or IP address.
port	int	8088	Splunk HEC port.
use_ssl	bool	FALSE	Use HTTP or HTTPS.
token	string	Empty	Splunk HEC token.
event_or_metric	string	event	Send as 'event' or 'metric'.
source	string	Empty	Source.
source_type	string	_json	Source type.

Sink Example

```
- name: splunkHecSink1
  connector: SplunkHEC
  address: localhost
  port: 8088
  use_ssl: false
  token: abc123
  event_or_metric: event
  source: source1
  source_type: _json
```

TCPASCII

Sends ASCII commands over a socket and reads back the response.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "TcpASCII".
address	string	Empty	Hostname or IP address.
port	int	23	Port.
read_delay	int	0	Delay in milliseconds between socket write and socket read.
reuse_connection	bool	TRUE	Maintain a single connection.

Source Example

```
- name: haas1
  connector: TcpASCII
  address: 192.168.111.216
  port: !!int 5051
  read_delay: !!int 400
  items:
    - name: SerialNumber
      address: ?Q100
```

TIMESCALEWS

Subscribes to data from a Timescale Historian via a WebSocket.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "TimebaseWS".
address	string	Empty	Hostname or IP address.
port	int	4511	Port.
items.address	string	Empty	Path to historian item.
items.group	string	Empty	Historian item group.

Source Example

```
- name: timebaseWsSource1
  connector: TimebaseWS
  address: localhost
  port: 4511
  items:
    - name: plcExecution
      group: MQTT Data
      address: dime/eipSource1/Execution/Data
```

TRAKHOUNDHTTP

Writes data to a Trakhound server.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "TrakhoundHTTP".
address	string	Empty	Trakhound hostname or IP address.
port	int	8472	Trakhound port.
use_ssl	bool	FALSE	Use HTTPS or HTTP.
router	string	Empty	Router name.
host_path	string	Empty	Host path.
base_path	string	Empty	Base path.

Sink Example

```
- name: trakhoundHttpSink1
  enabled: !!bool false
  scan_interval: !!int 1000
  connector: TrakHoundHTTP
  address: localhost
  port: 8472
  use_ssl: false
  router: default
  base_path: Ladder99:/DIME/HttpSink
```

UDP SERVER

Reads data from a UDP socket.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "UDPServer".
address	string	Empty	IP address, hostname is not accepted.
port	int	2323	Port to listen on.
items.address	string	Empty	Here 'message' represents the byte array received.

Source Example

```
- name: udpserver
  enabled: !!bool true
  scan_interval: !!int 1000
  connector: UDPServer
  address: 0.0.0.0
  port: 2323
  items:
    - name: message
      address: message
      script: return result[0]
```

WEBSOCKETSERVER

Serve data from a Websocket Server.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "WebsocketServer".
uri	string	ws://127.0.0.1:8082/	Websocket server address.

Notes

URI can be changed to ws://0.0.0.0:8082/ to serve data on all adapters.

Sink Example

```
- name: wsServerSink1
  connector: WebsocketServer
  uri: ws://127.0.0.1:8082/
```

WINTRISS SMARTPAC

Reads data from a Wintriss SmartPAC controller

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "SmartPAC".
address	string	Empty	Hostname or IP address.
port	int	1007	Port.

Source Example

```
- name: smartpacSource1
  connector: SmartPAC
  address: 172.16.200.18
  port: !!int 1007
  items:
    - name: PressType
      enabled: !!bool true
      script: return result[0];
    - name: PressName
      enabled: !!bool true
      script: return result[1];
```

XMLWEBSCRAPER

Reads data from an XML web page.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "XMLWebScraper".
uri	string	http://localhost/	Web page address.
namespaces	dictionary	Empty	Namespace mappings.
items.address	string	Empty	XPath query.

References

“XPath Syntax”: https://www.w3schools.com/Xml/xpath_syntax.asp

Source Example

```
- name: xml1
  connector: XMLWebScraper
  uri: http://192.168.111.216:8082/current
  namespaces:
    mt: urn:mtconnect.org:MTConnectStreams:1.2
  item_script: |
    return result.InnerText
  items:
    - name: node1
      address: //mt:Message[@dataItemId='sp2maxpwr']
```

YASKAWA ROBOT

Reads data from DX200, YRC1000, YRC1000 Micro Yaskawa controllers.

Name	Type	Default	Description
connector	string	"Undefined"	Connector type, "Yaskawa".
address	string	Empty	Robot IP address or hostname.
items.address	string	Empty	Robot variable address.

Variable Addresses.

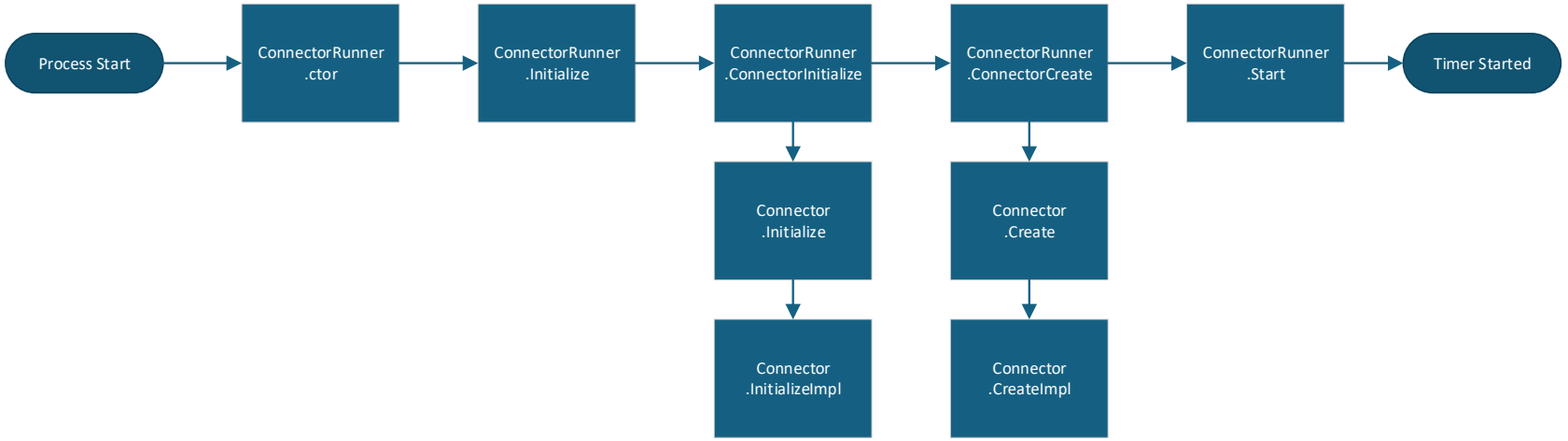
Address	Address	Address
cartesianPositions.X	torque.Axis1	status.CommandRemote
cartesianPositions.Y	torque.Axis2	status.InHoldStatusPendant
cartesianPositions.Z	torque.Axis3	status.InHoldStatusExternally
cartesianPositions.Rx	torque.Axis4	status.InHoldStatusByCommand
cartesianPositions.Ry	torque.Axis5	status.Alarming
cartesianPositions.Rz	torque.Axis6	status.ErrorOccurring
jointPositions.Axis1	alarm.Code	status.ServoOn
jointPositions.Axis2	alarm.Data	job.Name
jointPositions.Axis3	alarm.Type	job.Line
jointPositions.Axis4	alarm.OccurringTime	job.Step
jointPositions.Axis5	alarm.Text	job.SpeedOverride
jointPositions.Axis6	status.Step	
positionError.Axis1	status.Cycle	
positionError.Axis2	status.Automatic	
positionError.Axis3	status.Running	
positionError.Axis4	status.InGuardSafeOperation	
positionError.Axis5	status.Teach	
positionError.Axis6	status.Play	

Source Example

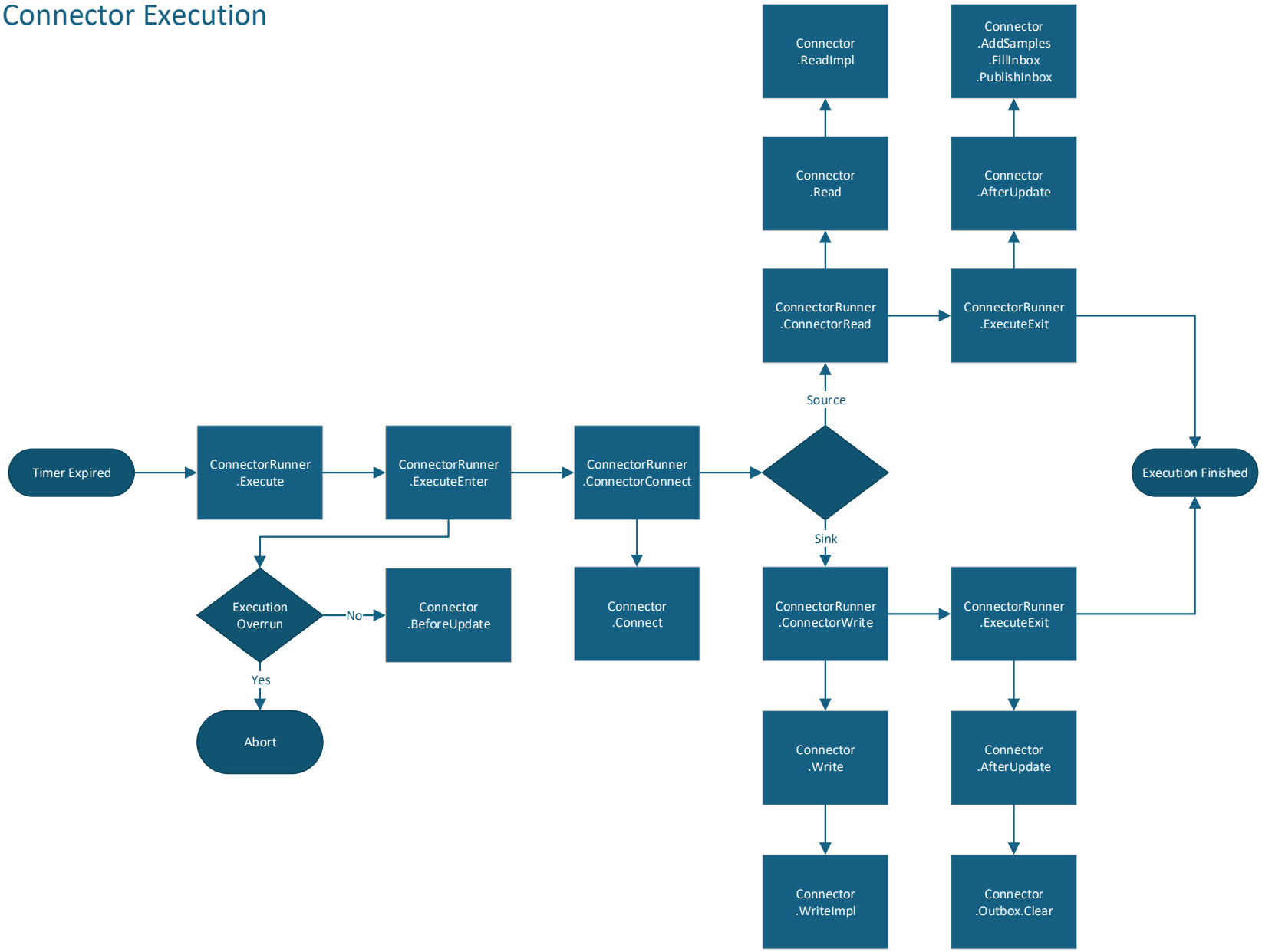
```
- name: yaskawa1
  connector: Yaskawa
  uri: 10.1.1.200
  items:
    - name: xpos
      address: cartesianPositions.X
```

Connector Lifecycle

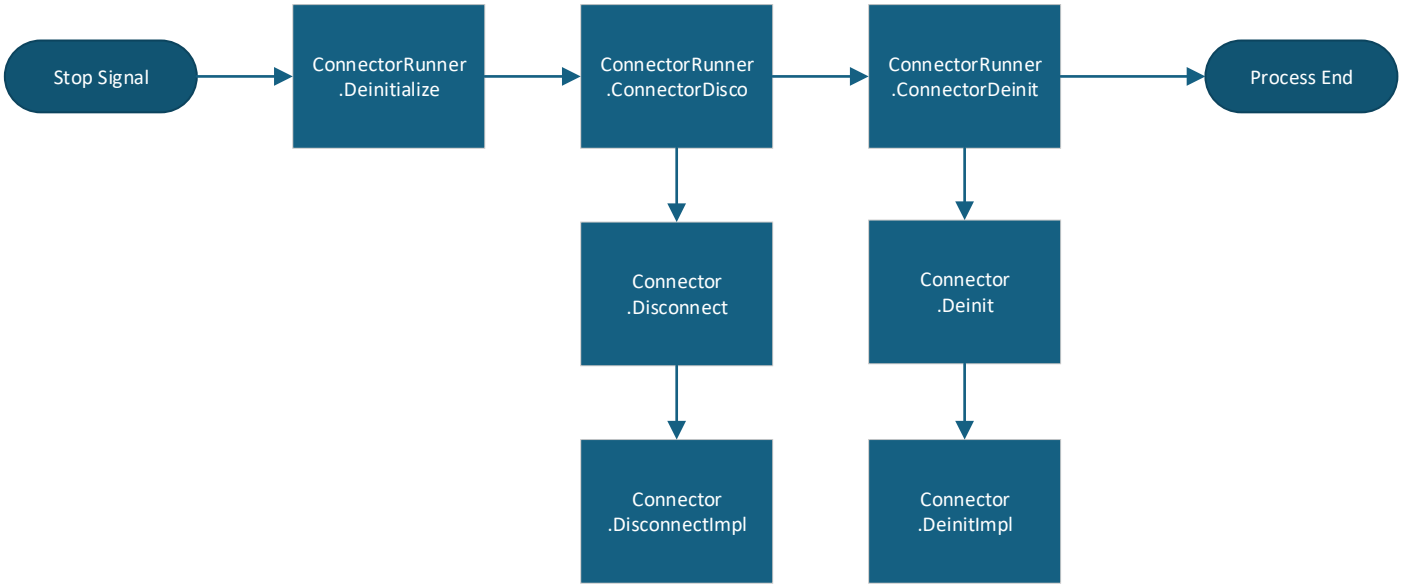
Connector Startup



Connector Execution



Connector Shutdown



Fanuc Driver for Fanuc CNC Machines

Fanuc driver is not part of DIME. Below are links to help you get started.

Windows and Docker releases: https://github.com/Ladder99/fanuc-driver/releases

Documentation: https://docs.ladder99.com/en/latest/page/drivers/fanuc/index.html

Videos:

- Fanuc-Driver Windows Setup https://www.youtube.com/watch?v=_sZ0qCVkTkM
- Fanuc-Driver Version 0.2 Changes: https://www.youtube.com/watch?v=yhe_apuiJVY
- Fanuc-Driver + MTConnect https://www.youtube.com/watch?v=j1gXHXA2RW0