English

Asema IoT Central

Data collection and analysis

ASEMA

Table of Contents

1. Introduction

In a nutshell, data collection and analysis is the process of recording IoT data and using various means to extract new insights from that data. Asema IoT Central offers a range of tools that help in doing that from the data collection stage all the way to statistical analysis of Big Data. You can freely choose which of these methods and their mixture to use. At the simplest form, Asema IoT Central only collects data i.e. interfaces with sensors and writes the captured values into database. Some other tool is then used to "mine" the data. At the other end of the spectrum is a fully automated routine that filters and corrects incoming data, runs statistical analyses on it automatically, writes the values to a report, and runs automation routines that react to the results.

Data processing is done with a set of algorithms. With Asema IoT, these algorithms can either be programmed into the system or run in external statistics programs. Asema IoT features various bridges that can connect to these programs, feed the data to them, and extract the results.

The process of data analysis starts by recording data with some data source such as a sensor. This data is recorded into Asema IoT's databases either in raw format of filtered. Filters ensure for instance coherent time periods between recorded values even in situations where sensors may provide data at uneven intervals.

Processed data and analysis results can then be stored back into the databases, presented as graphs to the users, used as triggers for logic processing, drive prices for the use of the system, or any combination of those. Various ways of building such feedback loops are described later in this manual.

2. Data collection reliability

Correct data is good data, flawed data is a nightmare. This general rule is good to keep in mind when processing data. Whenever there are possible errors in data, they should be corrected as close to the source as possible. The more correction factors are added and the farther away from the data corrections are made, the more likely there is trouble later on.

One of the very obvious ways to use data analysis and algorithms is to clean up the data. Nearly every measurement system has some inconsitencies. Outlier values here and there, missing data, skewed measurements. Tackling with these is a very common task and it is often tempting to fix the data in the same place where data is used. This is typically in the analysis phase where it is easy to spot outliers and other errors.

However, if there is any chance to fix the source of the data instead of patching it later, fixing the source should always be the primary approach. It will simply make life so much easier. This manual, and the Asema IoT software, cannot know and predict what type of problems for instance some sensoring hardware may have but there are certain things to take into account on the software and network side. The following lists some of the best practices to follow on that side to get good quality data.

2.1. Reliability of network connectivity

Missing data is usually the result of a disconnect on the network. Values simply never arrive. Software cannot remedy this but there are some things you can do. One option is timestamping values with corrected timestamps.

When you record a property value with Asema IoT, if no timestamp is present the current clock will be used as the timestamp. If values are delayed, this naturally causes a "hole" in the data. But the `CoreObject` class, which is the base object of all objects in an Asema IoT system, also supports the `recordPropertyAt()` method with timestamps. You can give both the location and the time as arguments to this method.

So if you have hardware with an unreliable network connection, instead of sending single values from the device, send value-timestamp pairs. When the timestamp is in the network message, then it is recorded with the clock of the sensor, without network delay skewing the recorded tie. Such value pairs can be sent in bursts and also out of order. Write a small driver for this hardware as a plugin and in the driver use `recordPropertyAt()` to write values into the database with appropriate timestamps.

2.2. Sensor operational reliability and accuracy

This is a tricky one as there usually is no magic bullet to fix an unreliable sensor. The sensor simply needs to be fixed. What you can do is to again try the value+timestamp approach as highlighted in the previous chapter. However, if unreliable data results in incorrect data being recorded, not missing data, this naturally is not the answer.

The best approach again is to try to fix the problem at the source, i.e. the sensor itself. The next best option is the driver as it sits closest to the hardware. Note that because drivers are loaded into memory and stay there, you can store state information in them. This way you can program state dependent value filters which change the filter values depending on the previous values received. This makes it possible to filter out values for instance based on the standard deviation of previously received values.

2.3. Data storage performance

When you send a lot of data, you may in some cases notice that the recording of the data simply cannot keep up with the rate received. This type of performance problem is common in for instance telematics.

Asema IoT has internal buffers that remedy parts of the problem when for instance disk access is slow or its speed fluctuates. Values are buffered in memory and written to database once access is available

again. This however works only if the data is bursty i.e. it does have some breaks in between where the data rate slows down.

If there is a constant problem with data writing speeds, then the answer typically lies in the database. Relational databases, especially with indices, may simply be too slow to handle the burden. For this purpose Asema IoT supports key-value databases such as Apache Cassandra. These databases are designed for speed and especially in clusters can achieve very high write rates. So if the database becomes a constant bottleneck, consider changing the type of database you use.

3. Data analysis

3.1. Sources of data

In most cases data analysis is based on recorded data, something that is in the database as a set of timestamp-value pairs. In Asema IoT, ordered sets of such pairs are called "input vectors". When you define an algorithm, you can usually define the input vector together with the algorithm. Alternatively, you can fetch those vectors dynamically in the code that uses the algorithm and feed that dynamic data to the algorithm.

However, the data analysis API is flexible in accepting any type of vector data. You can simply feed an array of values when you invoke an algorithm. This array can be sourced for instance from program logic or from a previous run from the algorithm.

Because everything in Asema IoT is an object and any object with properties can be an input to an algorithm, the choice of input data is really quite limitless. You could for instance take one input vector from a physical sensor, one from a webservice, and a third one from the results of a mathematical analysis.

Various "synthetic" i.e. generated or derived datasets often come to play when doing some more complex analysis tasks. To derive a dataset, you could for instance use the rule engine to program a set of rules that react to measurement values from a sensor. Link the rules to rule consequences that are programmed to set a property and the result is a timeseries that is derived from the original values of the sensor.

3.2. Real-time analysis

Real-time analysis is analysis of data that takes place immediately after a value is received. It is not periodic i.e. does not run with a timer (not even with a fast ticking one) but instead reacts immediately to data reception signals. There are two primary means in Asema IoT to do such analysis

1. Data cleaning, and

2. Rule based analysis

Data cleaning is something you define in the database logging configuration of an object. To access this configuration, in the Asema IoT Web admin interface go to Objects > Configure and open the Database logging settings of the object you want to configure. Here you can set both when the property values are stored to the database (store always or only when a change is detected) and in what frequency storage takes place. Setting a frequency makes the system run all property values through a filter. This filter will match the incoming data with timestamps that match the frequency, always finding the value that best matches a desired timestamp. With this method you will filter out overlapping values that arrive too fast or too irregularly and you get a nice, evenly spaced timeseries of values.

In rule based analysis you program a logical rule with the Rule Engine. You'll find the settings of the Rule Engine in Asema IoT Web admin interface under Automation. Here, add a condition that triggers when a property changes. Any incoming property value will invoke the evaluation of a condition. Then set a consequence which for instance changes the property of some other object or some other property of the same object. This property now becomes a derived timeseries of the received one and records the values of the derived series in real time.

3.3. Structured post analysis

Typical structured post analyses include for instance trend analysis and pattern finding. Structured means that there are some given points of time where past data is routinely checked with the same

algorithm to ensure that it does not (or does) show some trend that may require corrective action. Typical topics of such trends could be for instance

- Is the maximum voltage of our battery declining after each change, indicating that the battery is wearing out?
- Is the rate of excessive pressure alarms from the pump increasing over time, indicating wear in the piston?
- Is the water height of the reservoir decreasing as expected incidcating that the pumping speed is sufficient to offset the incoming water flows?

Structured analysis - as opposed to explorative analysis - is something where you "sort of know" what you are looking for in the data and want to repeat that procedure at regular intervals. In Asema IoT, algorithms is the primary tool designed for this purpose. You design an algorithm (or some other form of data processor) and make that run at regular intervals or per some regular incoming request. The chapters later in this manual will show in detail how to setup such algorithms and take them into use.

3.4. Explorative post analysis

While some of the data collected from sensors can be and is analyzed regularly, structurally and in most cases automatically, there are many use cases where data analysis is not that structured and predefined. The questions to answer from it are much more open ended, starting from the traditional "please check that there is nothing wrong with our data" to statistical data mining where new insights are sought from historical data with various mathematical methods.

Explorative analysis, as opposed to structured analysis, is not usually run at regular intervals. It is not automated as there is no clear pattern to be repeated.

In contrast, explorative, unstructured data analysis is something where researchers, data analysts and data miners get involved in. A researcher is given a dataset with an open assignment in the form of digging out anomalies, trends, correlations and other details not currently known or even anticipated.

Researchers tend to want to work with the tools they are familiar with. A MATLAB wizard most likely would like to solve the problem with Matlab while a social sciences expert will open the trusty old SPSS package. Therefore in practice, post analysis of data means creating data dumps of raw data and offering them as datasets to the researchers.

Making a basic datadump from a database suitable for such analysis is quite trivial. Databases have export tools for this purpose and many of the statistical tools can interface directly with the most common database backends. This is why Asema IoT does not have a specific data dump functionality (apart from Excel sheet export for small sets of data). The database tools already do this task very professionally. And if the data is high quality and uniform, usually nothing else is needed.

However, trouble arises when the data is not uniform. As a simple example the power consumption of electric appliances could reside in various units. Consumption of one could be expressed in wattminutes while the other has it in kilowatthours. There are two approaches to solving this problem: either unify the data while it is being exported or export it in a format that retains the essential information about for instance units. The latter may in many cases be the adviced format as it is the tryly "raw" format of the data and lets researches make conclusions also from the differences in units.

But such unification may get quite tedious. To help, this is where Smart API technology, natively supported by Asema IoT Central, comes along. Smart API contains vocabularies and data structure definitions that allow expressing the details of heterogeneous data in a rich format. The resulting data is RDF, a format appreciated by researches as it contains accurate descriptions or the data. When data is outputted as RDF over the Smart API, for instance conversions to one uniform unit can be made.

Now, apart from outputting uniform data, there are no other tools for explorative analysis in Asema IoT. This is for a reason: Asema IoT is an automation system and explorative analysis by nature is not automated.

This is why explorative analysis is usually done with "raw" database data and a direct connection to a database. This is perfectly fine and to get the data either a data dump can be made, a small wrapper programmed to interface the database, or the database can be connected to the statistics program. Many statistics programs can read for instance relational databases without further programming.

Important
If data analysis is done to a database that is simultaneously in production and connected to an Asema IoT instance, care must be taken to control concurrent access. Most proper database systems natively support concurrent access and lock the records to prevent data corruption in the case of simultaneous reads and writes. A notable exception is the SQLite database which is the default inbuilt database that is used if you have not configured any other database. As the name implies, SQLite is a light database system, only meant for small amounts of data and single user. If you do analysis on Big Data, please install and configure a database system designed for this purpose.

4. Creating data analysis algorithms

4.1. Algorithm sequences

The simples form of an algorithm is an algorithm sequence. A sequence is simply a set of mathematical functions linked together. They are evaluated in order; first one in the sequence first, then the one connected to it and so forth. Each step has some function and the input of the next step is the output of that function. This continues until the sequence completes. For example, if you have this data 10, 20, 30, 40 and then a sequence that has the functions divide by 2 and add 5, then the result would be 10, 15, 20, 25 i.e. (10/2+5), (20/2+5), (30/2+5), (40/2+5).

To create a sequence, proceed as follows:

- In Asema IoT web admin interface, go to Analysis > Algorithms and click on Add...
- In the form that opens, name your algorithm and give it an identifier. You can name it the way you want but do pay attention to the identifier as this is the string with which you point to the algorithm in various applications.
- Add the functions you want into the sequence by clicking on them in the left column of the editor.
- Connect the functions into a sequence by dragging the output point (dark blue circle) into the input point (light blue square) of the next step of the sequence.
- Input any factors required by the particular function, then press Save.

You now have a sequence to process the data. To use and test it, see more info in 5.2, “Feeding data to the algorithms”.

4.2. R scripts

"R" is the popular open source statistics package that lets you do professional, academic level data analysis at zero cost. Asema IoT Central interfaces directly with R and can use it to process data on-the-fly. To do this, Asema IoT Central uses the Rserve plugin that runs scripted analysis sequences in a separate R server. To learn how to set up your environment for this, read on.

4.2.1. Installing R on Linux

First, you need R, the software itself. In most Linux distributions, R is available in standard or statistics repositories. To install, consult the manual and the package manager of your Linux for details. For instance on OpenSuSE, R can be fond in the R-base package in the main repositiries and is installed with `zypper in R-base`. Note that the Rserve extension may need to build against your R installation so you also need the devel (source) packages of R.

Once R is installed, start it. From command prompt, this is simply

```
> R
```

Note that in order to install Rserve, you need to start R with a user that has the right to install software extensions (usually root). Next, do

```
> install.packages("Rserve")
```

Select an appropriate mirror and let the software download and compile the extension. Then exit R (ctrl+D).

You can now restart R as a normal user and then start Rserve. Like so:

```
> R
> library(Rserve)
> Rserve()
```

You now have R installed and running.

4.2.2. Testing the R installation

Rserve essentially relays R-command sequences - the R scripts from a remote system (in this case Asema IoT) into the R software. If you are familiar with R, testing the installation simply means running some R script you know and observing the results.

As an example, let's try something simple. In Asema IoT, go to Analysis > R scripts. Click on "Add new" and then type the following into the field that says "R algorithm"

```
(100+2)*3
```

Then click on "Run in R". You should now get a pop-up that says "306". If it does, your R installation is working fine.

Next, let's try some analysis function. Type into the "R algorithm" the following:

```
data <- c(10,10,10,20,30,40,50,50,50,50)
hist(data)
```

And again click on "Run in R". In the pop-up, you should get various details of the historam of those values, such as unique number present and histogram bucket sizes.

4.2.3. Processign data with input vectors

Now that R is up and running, it is time to do some actual number crunching. The standard way to do this is to feed into R some set of timeseries data. Do extract some, you add an input vector.

> **Important**
> Note that while you can put in data to R scripts with fixed input vectors, you can also use dynamic data input with R scripts in addition to this, just like with other means data analysis. While the input vectors have a predefined date range for data, with dynamic input you can define any date range (or other dynamic processing) before the data is fed to R. For details, see Chapter 5.2, “Feeding data to the algorithms” for more details.

Input vectors are located in the table just above your R script. You can have multiple of them and assign each into a separate variable. To edit, click on the editing icon. In the editor you can choose the object and the property you wish to have the data of as well as the time range.

Most importantly, here you will define the Assigned variable. This is the variable name you use to access the vector in the R script. So if you add a variable called "rawTemperature", you can then directly feed

this into the functions of R as a vector. For instance, to draw a histogram of rawTemperature, in your script you'd say `hist(rawTemperature)`. Every time you invoke the analyzer, Asema IoT will fetch fresh values from the database into the vector.

4.3. MATLAB

4.3.1. Installing and configuring MATLAB

Asema IoT Central communicates with MATLAB over the ZeroMQ (a.k.a. ZMQ or 0MQ) protocol. The method it uses is compatible with the python-matlab-bridge open source connector which you can download and modify in case you need further functionality or a precompiled messenger binary for your MATLAB installation (Asema IoT delivers only the source). You can find the project here https://github.com/arokem/python-matlab-bridge/. The project also includes instructions on installing ZMQ on various platforms (Windows, macOS, Linux).

To be able to use MATLAB with Asema IoT Central, you need to install ZMQ support into MATLAB as well as the parser for the protocol payloads. To do this, proceed as follows:

1. Install ZMQ development packages. This will also install the required ZMQ binary libraries into your workstation. The installation method depends on your operating system and package management. Example on SuSE Linux:

```
zypper in zeromq-devel
```

2. > Build the messenger. This is the binary file that runs and parses ZMQ

```
cd messenger
python make.py matlab
```

3. Copy the messenger in place. The messenger is a mex file so MATLAB will try to find it in a similar fashion as any mex file, starting from current path and then going through the search path. More info on search paths can be found here: https://se.mathworks.com/help/matlab/matlab_env/files-and-folders-that-matlab-accesses.html. An easy way to determine a proper location is to start MATLAB, then type

```
>> userpath
```

   Copy the messenger mex file to the path shown by this command.

4. Install the server MATLAB script matlabserver.m into a location where you'll find it. The location itself is somewhat irrelevant because you'll be starting MATLAB and then running this script.

5. Install the utility and user programs under usrprog and util to a location where MATLAB finds them e.g. the userpath.

6. Start MATLAB.

7. In MATLAB, start the server

```
>> matlabserver <address>
```

Note that the default address used by Asema IoT Central is `tcp://127.0.0.1:8899` so unless you've changed that setting, use this value as the address.

4.3.2. Testing and debugging the MATLAB setup

Asema IoT Central's debugging output includes the data transmitted from and to MATLAB. To view this output either

- Start Asema IoT Central with the -N flag with debug level at at least 2 (i.e. -N 2); or
- Set the debug level from system settings at System > Logging and console to "Detailed"

When you run any MATLAB algorithm with debugging on, you will see whether MATLAB responds to your queries and any errors that MATLAB may produce. If

- you only see messages being sent but none received, check that the bridge address has been set to be the same both in Asema IoT Central and MATLAB, that MATLAB is running and that you have installer the `messenger` mex file into MATLAB
- you receive errors as a response, make sure you have installed the correct version of matlabserver.m into MATLAB and that it is running
- you receive an error that the function processor is not working, ensure you have installed all the files under usrprog (and especially test_sum.m) into a location where MATLAB can find them.
- you receive an error that the algorithm you are trying to run cannot be found in MATLAB, make sure you have authored the corresponding .m file and it is in a location where MATLAB can find it

To make a simple testrun, create a MATLAB algorithm in Asema IoT Central and then click on "Run in MATLAB". You should once clicked, you will get a pop-up that displays the result of the calculation, or if your .m file does not exist or does not accept the input, an error message from MATLAB.

4.3.3. Processign data with input vectors

You may feed data from your IoT objects to MATLAB with the help of input vectors. The vectors define a timeseries input to the algorithm in the form of an object and its property as well as a date and time range from which it is extracted.

> **Important**
> Note that while you can feed data to MATLAB scripts with fixed input vectors, you can also use dynamic data input. While the input vectors have a predefined date range for data, with dynamic input you can define any date range (or other dynamic processing) before the data is fed to R. For details, see Chapter 5.2, “Feeding data to the algorithms” for more details.

Input vectors are located in the corresponding table in the form where you create your MATLAB algorithm script. You can have multiple of them and assign each into a separate variable. To edit, click on the editing icon. In the editor you can choose the object and the property you wish to have the data of as well as the time range.

In MATLAB, the input vectors appear as an array comprising of dictionaries. Each dictionary contains the variable name assigned to the input vector (with dictionary key "var") and the values of the vector as an array (with dictionary key "values"). So, for example, if you have made entered two input vectors and assigned them to "speed" and "acceleration", in MATLAB you could access them as follows:

```
function result = speed_and_acceleration(args)
  speed_values = args(1).values;
```

```
  speed_variable = args(1).var;
  acceleration_values = args(2).values;
  acceleration_variable = args(2).var;
  result = 0;
end
```

In your .m function, to return a result you assign it to the function. Asema IoT Central will convert the result to an array of values and return it to your code. Note that the return value is always an array, even if your .m produces just a single value. So for intance this simple function would return [75] i.e. an array containing one value, 75.

```
function result = simple_sum(args)
  a = 40;
  b = 35;
  result = a+b;
end
```

4.3.4. Running MATLAB algorithms

As with other algorithms, you can run the MATLAB algorithms from Web Applications or from QML scripts (Screenlets) by using the appropriate API. There are several examples of both later in this manual, but let's take a quick peek at one which is a simple Web Application.

This demo shows two types of data being fed into MATLAB: one an input vector from an actual object and its property and one that is just a JavaScript array. Similar to all other Web Application examples, it begins by creating a `DataAnalysisManager` instance in the `load()` method which is a callback that takes place once all the JavaScript libraries have loaded. It then connects the signals needed to listen to the status of the algorithm processing and starts the `DataAnalysisManager`. This process creates all the connections to the backend used in monitoring the MATLAB process.

Once the connections are done, `onAnalyzerReady()` gets called and this invokes the `analysisManager.runAlgorithm()` method. It takes as its argument the identifier of the algorithm (which you fill in into the form when you create a MATLAB algorithm) and the inputs to feed. The inputs are simply a list which may contain other lists (the actual values of a particular vector) or a map that has settigs of a vector. In the settings the GID of the object in case is inputted as well as the property to fetch. And of course the timespan that determines the period of the property values.

```
<!DOCTYPE html>
<html>
<head>
 <title>MATLAB demo</title>
 <script type="text/javascript" src="/inc/libs/jquery.min.js"></script>
 <script async defer type="text/javascript" src="jarvis-1.0/import?callback=load"></script>
</head>
<body>
 <div id="content">
  <h2>MATLAB demo</h2>
 </div>
<script type="text/javascript">
 var analysisManager = null;
 var analysisCallId = 0;

 function onAnalyzerReady()  {
  var sampleVector = {
   "assigned_variable": "depth",
   "source_object_gid": "fd5182c6458df1cfc61b4189c9ef49994e8286aa",
   "source_property": "temperature",
   "from": "2018-04-01T12:19:15",
   "until": "2018-04-26T12:19:15"
  }
  analysisCallId = analysisManager.runAlgorithm("sampleMatlabAlgorithm", {"inputs": [[10, 20, 30, 40, 50, 66], sampleVector]});
 }

 function onAnalysisStarted(id)  {
  console.log("Analysis with ID " + id + " started");
 }

 function onAnalysisDone(id, algorithmId, result)  {
  if (id == analysisCallId) {
   console.log(result.data);
```

```
  }
 }

 function onAnalysisError(id, errors)  {
  console.log("Errors in analysis with ID " + id);
  console.log(errors);
 }

 function load() {
  analysisManager = new DataAnalysisManager();
  analysisManager.start();
  analysisManager.ready.connect(onAnalyzerReady);
  analysisManager.analysisStarted.connect(onAnalysisStarted);
  analysisManager.analysisReady.connect(onAnalysisDone);
  analysisManager.analysisError.connect(onAnalysisError);
 }
</script>
</body>
</html>
```

Once the algorithm finishes, the return value is shown in JavaScript console using the `onAnalysis-Done()` callback method. And what is the return value? Well that depends on what MATLAB has been programmed to do with the values with the corresponding .m file (the name of which you enter into the form when you create the MATLAB algorithm).

4.4. JavaScript

If the statistical tools and pre-packaged algorithms are not enough for your needs, you can always program a new one! This is what JavaScript algorithms are for. They are scripts that take the input vectors and output the result.

First, note that there are many ways to script the Asema IoT and customize its behavior. One such option is Server Side Scripts which are different from analysis scripts discussed here, though the language and process of creating them is exactly the same. The primary difference between JavaScript analysis scripts and JavaScript Server Side Scripts is the supported call methods and the way the scripts are invoked.

Server Side Scripts are always assumed to be invoked with a request. They receive the request wrapped into a dedicated request object and respond with a dedicated response object. Server Side Scripts also have a path: you invoke them by calling the server at this path.

Analysis scripts do not have such a request-response pattern. They have a mandatory `run()` method which is called by the analysis processor. The analysis processor wakes up the algorithm when a call with a matching identifier is made to it.

Otherwise the scripts are vary similar (the actually internally inherit the same base class and methods). So if you can program one, you can program the other.

Note that is possible to leave a Server Side Script "running" i.e. it may have a timer that keeps the logic in memory and in the next request, provides values that have been processed in between calls. The script will remain in the engine and can do so as it is not stopped or unloaded between calls. The next request can use the values stored in memory by the previous request. However, apart from storing state variables, this kind of "stay resident" is not the primary purpose of Server Side Scripts.

Analysis on the other hand is definitely something that may take place, and is more often than not desired to take place in the background at regular intervals. Do do this with maximum flexibility with JavaScript analysis scripts, you can include a QML Timer element into your script. This timer starts when the analysis script is invoked the first time. For example, to make an analysis run once per minute, the Timer element would look like this

```
Timer {
    interval: 60000
    repeat: true
    running: true
    onTriggered: {
        run();
    }
}
```

So, with all that in mind, let's take a look at a full listing of a simple JavaScript analysis script:

```
import AnalysisEngine 1.0

AnalysisScript {
 function run(arglist) {
  console.log("Now running the analysis....");
  var resultL = "";
  for (var a in arglist) {
   resultL += arglist[a];
  }
  return resultL;
 }
}
```

The analysis script gets the input vectors as an argument to the `run()` method. It can then do whatever manipulation you want with the input data. In the example above, it just concatenates the numbers into a string and returns the result.

Data can be fed into the JavaScript AnalysisScripts as input vectors just like with other algorithm methods. However, you also have the option to perform direct queries from the database with the scripts.

Note that you also have standard JavaScript debugging methods such as console logging at your disposal. The output from the console will be directed to the logging facility you have chosen in the system (terminal window, log file, etc).

In the example below we have a script that manipulates data from some data table called "stats". To get values from it, you call the rawQuery method and construct some SQL select clause that fetches the data.

```
import AnalysisEngine 1.0

AnalysisScript {
 function run(args) {
  console.log("Do a database query");

  var resultL = "";
  var dbValues = rawQuery("SELECT * from stats");
  for (var b in dbValues) {
   var c = dbValues[b];
   for (var d in c) {
    resultL += "," + c[d];
   }
  }
  return resultL;
 }
}
```

Again, the example is simple and only concatenates datafields, but this should give you an idea of the versatility of the method.

4.5. External analysis programs

It is often the case that you actually have some little (or maybe a big one) program that has been programmed for modeling some mathematical or other analysis or data processing. This could for instance be a Python or Lua script. Trasnferring that into an algorithm to run inside Asema IoT may sometimes be too laboursome or even impossible if the program uses some very specific methods only available in that environment.

Worry not, for this purpose Asema IoT Central supports running algorithms externally. For this purpose your script should be able to be started either as a command line program or via an HTTP call. Either way, your script or program will be invoked with one of the methods, should then process the data and finally use a suitable API of Asema IoT to feed back the results.

Note the wording "a suitable API". When the external process is invoked, Asema IoT does not expect to get a reply back. The reason for this is simple: there are so many ways to process data and so many potential ways to respond to it that parsing the incoming data would be nearly impossible. Instead, the analysis process should do the calculation and then use Asema IoT's APIs, such as the JSON API, to call the system and feed the results into the properties of the objects. This way there is full freedom of outcome of the algorithms without a cumbersome parsing procedure.

The same applies to the input to your algorithm. It would be impossible to predict what type of processing your program may have at its disposal to parse the input vectors. This would in any case need to be coded in. So rather that doing that, the program is simply started. It should again use an API of Asema IoT, such as the JSON API, to fetch the data it needs for processing.

To add an external program as an analysis algorithm, in Asema IoT Web admin interface go to Analysis > External. Here, add a new algorithm by giving it a name and an identifier (the identifier is can be used to invoke the algorithm on demand). You can choose to leave the algorithm without automatic invokation and only run it through a call (from a Web application for instance) or set a timer to it so that your data gets analyzed at regular intervals.

Finally, choose whether your script is called via HTTP GET or as a subprocess. Depending on your choice, see below.

4.5.1. Calling external analyzers as processes

When you want the analyzer to start as a process (a command line program), enter the full path of the program executable into the Subprocess path field. Whenever the algorithm is invoked, Asema IoT will simply call this binary. The call starts a separate subprocess, the Asema IoT main program will not wait for it to finish but will monitor the progress and knows when the process ends.

Starting the analyzer as a process is a very simple and straighforward way to run an analysis. Note however that whenever one round of analysis is done, the whole program binary is loaded and started. Depending on your program, this might be a heavy task and take time, often unnecessarily so.

4.5.2. Calling external analyzers over HTTP

When you want the analyzer to start using HTTP GET, you need to ensure that your analyzer has a small HTTP server listening to some port and path. This path will be called with HTTP GET to start the analysis. Enter the full path to call, including the host address into the HTTP URL field. Whenever the algorithm is invoked, Asema IoT will simply call this path. The call is async so while your HTTP server should respond as soon as it gets a request, the delay will not in any way affect the running of Asema IoT main program.

Starting the analyzer using HTTP GET has the benefit of having a low overhead. The analyzer is already running and does not need to be loaded. This method is therefore lightweight and fast. Depending on the runtime of your algorithm, the method can be re-invoked at a rapid pace if so desired.

5. Running and calling algoritms

5.1. Notifying when a result is available

Before talking about how to start an algorithm, let's go through what happens **after** the algorithm has run. Better to do that here as it might affect the way you go about calling the algorithm in the first place.

Now, a common problem in Big Data calculation is that it takes time, usually an unknown amount of time. Tt therefore requires some method to inform other programs that they now have something to display or process further. So for instance, if you'd be running , say, a map-reduce algorithm with Hadoop on the data you've collected, how would you go about telling Asema IoT that now the calculation has finished?

The first, and usually worst, option is to have some polling going on. You could for intance have some interval set in the user interface that tries to redraw a graph every 10 seconds. But this is kind of counter-productive because it a) is not real-time, b) is often unreliable, and c) consumes resources unnecessarily.

A better option is to use the property system available in Asema IoT Central. it gives you real time access to the results and you can use all the benefits of integrated value propagation in the system to get your results to all interfaces and connected systems automatically. Whenever a property changes, a signal of this is sent across the system. Therefore, store values of algorithm runs into properties of objects.

So if you are running that Hadoop job, create a new object in Asema IoT system and call it for instance HadoopJob. That object could be of two types: a) a Server API object or b) Database object, depending on where and how you store your results from the Hadoop job.

If you store the values into a database that supports signaling (i.e. currently Postrgresql), choose Database object as the type. Then, choose to update the values per database signal (a feature supported by Postgresql). When you now update the values in your result database, that Database object will automatically notice this, load the values from the database, set its properties accordingly, and notify everyone of the results.

If you do not store the result in a database (or it is not a Postgresql database), create a Server API object. When your calculation script finishes, call the API of this object (see manual for details) and use the `set_property` method to set the properties of that object as the results of your calculation. Again, the object will take care of propagating the results to everyone in real-time.

5.2. Feeding data to the algorithms

To feed actual data to analyze with your algorithms, you need some object and a use case to do so. The most straightforward case is that you're collecting some data and would want to present it in a processed and analyzed format.

This chapter therefore assumes that in your IoT system

- There is some object that has properties (some hardware, an API feed, a network client, a dabase client)
- You have turned on data logging of that object so the data gets collected into the database.

If not, make sure you have these first before proceeding, otherwise there won't be any data in the database to begin with.

Next, you need programmatic access to your algorithm. The method of getting access will depend on what type of application you are building: a Web application (pure JavaScript) or a Screenlet or report (QML). In screenlet applications and PDF reports, you access algorithms thrpugh the API of theAnalysisContext. In Web applications, you use the DataAnalysisManager class.

Both approaches above take as an input the identifier of your analysis algorithm and a vector of data (or possibly multiple vectors). They then return as an output the output produced by your algorithm.

The examples below assume that you have created some algorithm and call it "myAlgorithm" (which should be the value of the **identifier** field of your algorithm when you add it with a form).

5.2.1. Feeding an arbitrary array of data

Let's start with a simple webpage. Instead of taking data from an object, we simply form the data vector on the page and feed it in to the algorithm. Store the page to the `public_html` directory of your Asema IoT installation. If you call it for instance "algorithmdemo.html", you can then access it at http://127.0.0.1:8080/pub/algorithmdemo.html (assuming you have not changed the default settings of the system internal webserver, in which case the path may differ). The sample page looks like this:

```
<html>
<head>
        <script type="text/javascript" src="/inc/libs/jquery.min.js"></script>
        <script async defer type="text/javascript" src="jarvis-1.0/import?callback=load"></script>
        <style>
                #content { text-align: center; }
                #resultdisplay { font-size: 70px; }
        </style>
</head>
<body>
<div id="content">
        <h2>Algorithm processing demo</h2>
        <div id="resultdisplay"></div>
</div>
<script type="text/javascript">
    var analysisManager = null;
    var analysisCallId = 0;

    function onAnalyzerReady()  {
        var inputVector = [1, 10, 200, 3000, 40000];
        analysisCallId = analysisManager.runAlgorithm("myAlgorithm", {"inputs": [inputVector]});
    }

    function onAnalysisStarted(id)  {
        console.log("Analysis with ID " + id + " started");
    }

    function onAnalysisDone(callId, algorithmId, result)  {
        if (id == analysisCallId) {
            $("#resultdisplay").html(result.vector_0.join(" "));
        }
    }

    function onAnalysisError(id, errors)  {
        console.log("Errors in analysis with ID " + id);
        console.log(errors);
    }

    function load() {
        analysisManager = new DataAnalysisManager();
        analysisManager.start();
        analysisManager.ready.connect(onAnalyzerReady);
        analysisManager.analysisStarted.connect(onAnalysisStarted);
        analysisManager.analysisReady.connect(onAnalysisDone);
        analysisManager.analysisError.connect(onAnalysisError);
    }
</script>
</body>
</html>
```

What the page does is that it first instantiates one instance of DataAnalysisManager and connects it to the backend so that it can receive results from the algorithm (note: you must have working WebSockets for this to happen so don't block them in a firewall or similar). Once the connection is established, it calls your algorithm in the `onAnalyzerReady` signal handler. Once the algorithm completes, `onAnalysis-Done` signal is received and the results will be displayed on screen.

Important
Note that algorithms can take multiple input vectors of various kinds as input. This is why in the example above the input to the algorithm is a list of lists i.e. a list of input vectors each of which is a list. If you don't name the inputs, the

resultset will contain the results named as vector_0, vector_1, vector_2, etc in the order you entered them.

5.2.2. Feeding property data from the database

Next, let's work with some actual recorded data. For this purpose, you need to form an input vector that says from what object and property you'd like to feed to the algorithm. So let's modify the `onAnalyzerReady()` method a bit (otherwise the code should be as is in the example above):

```
function onAnalyzerReady()  {
 var temperatureVector = {
  "source_object_gid": "fd5182c6458df1cfc61b4189c9ef49994e8286aa",
  "source_property": "temperature",
  "from": "2018-04-01T00:00:00",
  "until": "2018-04-10T00:00:00"
 }
 analysisCallId = analysisManager.runAlgorithm("myAlgorithm", {"inputs": [temperatureVector]});
}
```

In this new format, we're defining an object, with a GID and its property we're interested in. Additionally the vector defines for which period the data should be fetched. When the page loads, Asema IoT will fetch the data and feed it to your algorithm for processing.

Important
Note the format of the from and until timestamps which is strictly yyyy-MM-ddThh:mm:ss

Of course you can process multiple vectors in your algorithm. In this case it is also a good practice to name the vectors with a variable name so that your algorithm can separate them. You do this with the `assigned_variable` attribute.

Let's for example assume you've made an algorithm (called "temperatureCorrelation") in R that correlates air temperature with altitude and you have an object (say a weather balloon) that measures these two. To feed data to it, you would proceed as follows:

```
function onAnalyzerReady()  {
 var temperatureVector = {
  "assigned_variable": "temperature",
  "source_object_gid": "fd5182c6458df1cfc61b4189c9ef49994e8286aa",
  "source_property": "temperature",
  "from": "2018-04-01T00:00:00",
  "until": "2018-04-10T00:00:00"
 }
 var altitudeVector = {
  "assigned_variable": "altitude",
  "source_object_gid": "fd5182c6458df1cfc61b4189c9ef49994e8286aa",
  "source_property": "alt",
  "from": "2018-04-01T00:00:00",
  "until": "2018-04-10T00:00:00"
 }
 analysisCallId = analysisManager.runAlgorithm("temperatureCorrelation", {"inputs": [temperatureVector, altitudeVector]});
}
```

Now your R script will receive two input variables, temperature and altitude, and can do with them whatever processing is necessary in your application.

5.3. Using algoritms in Web applications

Now that you have an algorithm and know how to call it, let's do some fancier graphic with the results.

This example assumes that you have created an analysis algorithm called `my_analysis_algorithm` which provides chart data suitable for histograms. The output should be the same as what R produces

with the `hist()` function. Note that you type the `my_analysis_algorithm` name into the "Unique identifier" field of the algorithm you created at Asema IoT admin interface.

Below is a sample web page. To use it, store the page into the public_html folder of your Asema IoT central installation and then open it. If you store as, say, "chart.html", then you can access the page at the address http://127.0.0.1:8080/pub/chart.html, assuming you use localhost and have kept the port and public path settings as defaults. So here's the page:

```
<!DOCTYPE html>
<html>
<head>
 <title>Algorithm runner and result display demo</title>
 <script type="text/javascript" src="/inc/libs/jquery.min.js"></script>
 <script type="text/javascript" src="/inc/js/charts-1.0/chartjs.min.js"></script>
 <script type="text/javascript" src="/inc/js/charts-1.0/chartjs.bundle.min.js"></script>
 <script async defer type="text/javascript" src="jarvis-1.0/import?callback=load"></script>


 <style>
  body {
   background-color: #ededed;
  }
  #content {
   text-align: center;
  }
  #resultdisplay {
   font-size: 200px;
   font-weight: bold;
  }
 </style>
</head>
<body>
<div id="content">
 <h2>JavaScript algorithm processing demo</h2>
 <canvas width=340 height=200 id="resultdisplay"></canvas>
</div>
<script type="text/javascript">
 var analysisManager = null;
 var analysisCallId = 0;

 function onAnalyzerReady()  {
  analysisCallId = analysisManager.runAlgorithm("my_analysis_algorithm", {});
 }

 function onAnalysisStarted(id)  {
  console.log("Analysis with ID " + id + " started");
 }

 function onAnalysisDone(callId, algorithmId, result)  {
  if (id == analysisCallId) {
   console.log(result);
   drawGraph(result[0], result[1]);
  }
 }

 function onAnalysisError(callId, errors)  {
  console.log("Errors in analysis with ID " + id);
 }

 function drawGraph(x_labels, y_data) {
  var ctx = $("#resultdisplay");
  var myChart = new Chart(ctx, {
   type: 'bar',
   data: {
    labels: x_labels,
    datasets: [{
     label: 'My data',
     data: y_data,
     backgroundColor: [
      'rgba(255, 99, 132, 0.2)'
     ],
     borderColor: [
      'rgba(255,99,132,1)'
     ],
     borderWidth: 1
    }]
   },
   options: {
    scales: {
     yAxes: [{
      ticks: {
       beginAtZero:true
      }
     }]
    }
   }
  });
 }

 function load() {
```

```
  analysisManager = new DataAnalysisManager();
  analysisManager.start();
  analysisManager.ready.connect(onAnalyzerReady);
  analysisManager.analysisStarted.connect(onAnalysisStarted);
  analysisManager.analysisReady.connect(onAnalysisDone);
  analysisManager.analysisError.connect(onAnalysisError);
 }

</script>
</body>
</html>
```

The beef of the page is in these lines:

```
    analysisManager = new DataAnalysisManager();
    analysisManager.start();
    analysisManager.ready.connect(onAnalyzerReady);
    analysisManager.analysisStarted.connect(onAnalysisStarted);
    analysisManager.analysisReady.connect(onAnalysisDone);
```

These create a new Data Analysis Manager and attach its WebSocket signals to the backend. The signaling method ensures that your algorithm can run for as long as it takes without a fear of a browser timeout. Once the connections are ready, it starts the algorithm with `analysisManager.runAlgorithm("my_analysis_algorithm", {});`.

Once the algorithm finishes, the results are captured by `onAnalysisDone(id, result)`. This method will then take parts of the data and put it into a graph.

5.4. Using algorithms in screenlets

5.4.1. Using the AnalysisContext

The AnalysisContext is the main API for accessing algorithms from screenlet applications. The API is asynchronous: you call the algorithm and the context will signal your program with results once done. This method makes it possible to run algorithms that take a long time to process without the fear of timeouts. It makes calls to external analysis programs that may use an asynchronous API possible.

The AnalysisContext API is simple: it has one method to call, one signal for results, and one signal for errors. Nothing much more is needed as all the complexity of calculations is actually in the algorithms themselves. All you need to do it provide the data / input vector and then capture the result.

The result of a call to the AnalysisContext is a variant. What is inside the variant will depend on the algorithm. It may be a map, a list, or just a single value. The ambiguity is intentional as different algorithms may need different outputs so it gives you the maximum flexibility in designing those algorithms to suit each application. The downside of course is that you will need to know what is inside the object in order to use it.

The `runAlgorithm()` of AnalysisContext takes three arguments: 1) a freeform `callData` parameter which is returned in the response signal, 2) the name of the algorithm to run, and 3) the arguments to the algorithm. You can use the `callData` as a call identifier in case you run multiple analysis calls in parallel and want to recognize the call origin once you get results or it can be used as some other call specific callback data that you process when you get the results.

Let's have an example. In this example there is an algorithm in the system called "myAlgo" it takes a list of five values as its input an returns a map containing the average and median of those five values.

```
import QtQuick 2.3
import Jarvis 1.0

Item {
 id: screenletframe
 width: deviceContext.screenWidth
```

```
 height: deviceContext.screenHeight

 property ScreenletContext screenletContext

 function open() {
  // Just run the analysis immediately once the screenlet is ready
  runAnalysis();
 }

 function close() {}

 function getExpandedOpenParam() {
  return "";
 }

 function runAnalysis() {
  // Request the average and median of these five values
  var params = [100, 30, 200, 60, 75];
  analysisContext.runAlgorithm(0, "myAlgo", params);
 }

 Text {
  id: resultDisplay
  anchors.centerIn: parent
  color: "white"
  font.pointSize: 40
  text: ""
 }

 // React to the values returned by the algorith
 Connections {
  target: analysisContext

  onAnalysisComplete: {
   resultDisplay.text = "Average is " + result.average + " and median is " + result.median;
  }

  onAnalysisError: {
   resultDisplay.text = "Error running the algorithm: " + error;
  }
 }
}
```

5.4.2. Using object data as an input vector

If you want to use object data as an input vector, the most flexible way to do that is to fetch it from the object and then feed it back to the algorithm. For this purpose you first find and load the object and then use the `getMeasurementValueTimeSeries()` method to fetch it. Once received, you call the algorithm. Like so:

```
import QtQuick 2.3
import Jarvis 1.0

Rectangle {
 id: screenletframe
 width: deviceContext.screenWidth
 height: deviceContext.screenHeight

 property ScreenletContext screenletContext
 property variant myObject: null

 function onLoadedObjectsReady(propertyName) {
  if (propertyName == "sampleObject") {
   var now = Date();
   // Fetch one hour of data
   var start = now.addSeconds(-3600);

   myObject = screenletContext.objects.motorObject.list[0];
   myObject.measurementValueTimeSeriesReady.connect(onPropertyValuesReceived);
   myObject.getMeasurementValueTimeSeries("moisture", start, now);
  }
 }

 // Run the analysis whenever we receive a timeseries
 function onPropertyValuesReceived(key, vals) {
  analysisContext.runAlgorithm(0, "sampleAnalysis", vals);
 }

 function open() {
  // At the start of the screenlet, find the object to operate with
  screenletContext.objectsReady.connect(onLoadedObjectsReady);
  objectManager.findObjectByComponentId(screenletContext, "sampleObject", "mySample");
 }

 function close() {}
```

```
 Connections {
  target: analysisContext

  onAnalysisComplete: {
   console.log(result)
  }
 }
}
```

5.4.3. Feeding data back to objects

Once you've done the analysis, you'll probably want to do something useful with the results. The obvious alternative is to display the results to a user. This is reasonably straightforward and the example in the previous chapter shows a simple means of doing this i.e. just place the result in a text element. Alternatively you could use some graphs or other graphics, etc.

But what if you'd like to store those results back in a database or cause some action in an object? For this purpose, you'll need a handle to an object that is the target and then change a property of that object. If you have set database logging on for that object, the value will be written in the database. If the object is hooked to a hardware device, the device will react to the property change. So in most such applications, you'd simply create the objects with which you like the actions to happen and then use the ObjectManager to locate them and operate them. For a database write you could for instance create an obhect called "MyAnalysisResults". Writing the properties of that object will give you a timeseris of analysis results in the database.

Let's have an example again. In this fictional example we will be analyzing the performance curve of a motor. The curve performance is modeled by an algorithm that takes in the torque values of the past minute. If the values drop during that minute in an abnormal way, the algorithm result contains a warning. If so, our sample application drops the power input by 10%. If no warning is received, things continue without change. A timer is set to run the analysis from the application every 30 seconds.

```
import QtQuick 2.3
import Jarvis 1.0

Rectangle {
 id: screenletframe
 width: deviceContext.screenWidth
 height: deviceContext.screenHeight

 property ScreenletContext screenletContext
 property variant motor: null

 function onMotorPropertyChanged(key, value) {
  console.log("The motor " + key + " chnaged to " + value);
 }

 function onLoadedObjectsReady(propertyName) {
  if (propertyName == "motorObject") {
   motor = screenletContext.objects.motorObject.list[0];
   motor.propertyChanged.connect(onMotorPropertyChanged);
   motor.measurementValueTimeSeriesReady.connect(onPropertyValuesReceived);

   // We have an object to run the analysis with. Start the timer
   // and off we go!
   analysisTimer.running = true;
  }
 }

 // Run the analysis whenever we receive a timeseries from the motor
 function onPropertyValuesReceived(key, vals) {
  analysisContext.runAlgorithm(0, "motorTorqueAnalyzer", vals);
 }

 function open() {
  screenletContext.objectsReady.connect(onLoadedObjectsReady);
  objectManager.findObjectByComponentId(screenletContext, "motorObject", "myMotorObject");
 }

 function close() {}

 function getExpandedOpenParam() {
  return "";
 }

 // This timer will run our algorithm at regular intervals
 Timer {
  running: false;
```

```
   interval: 30000
   repeat: true
   onTriggered: {
    // Fetch motor timeseries. When the values are received, this
    // triggers a call to the algorithm
    if (motor != null) {
     var now = Date();
     var start = now.addSeconds(-60);
     motor.getMeasurementValueTimeSeries("torque", start, now);
    }
   }
  }

  // React to the values returned by the algorithm
  Connections {
   target: analysisContext

   onAnalysisComplete: {
    if (result.alert) {
     alertDisplay.color = "red";
     alertDisplay.text = "Engine torque drop! Reducing power";
     if (motor != null) {
      motor.properties.power = motor.properties.power * 0.9;
     }
    } else {
     alertDisplay.color = "green";
     alertDisplay.text = "Engine running OK!";
    }
   }

   onAnalysisError: {
    alertDisplay.text = "Error running the algorithm: " + error;
    alertDisplay.color = "lightgray";
   }
  }

  Text {
   id: alertDisplay
   anchors.centerIn: parent
   color: "red"
   font.pointSize: 40
   text: ""
  }
 }
```