Unofficial Snippet Guide

-work in progress -

1. Control Flow

1.1. if

Description: This will insert a simple if clause
Documentation link: https://www.w3schools.com/js/js_if_else.asp
Example usage:

```
if (x<=0)
{
logger.warn("Value is too low");
}
```

1.2. if/else

Description: This will insert a simple if clause followed by an else
Documentation link: https://www.w3schools.com/js/js_if_else.asp
Example usage:

```
if (x<=0)
{
logger.warn("Value is too low");
}
else
{
logger.warn("Value is in normal range");
}
```

1.3. try/catch

Description: This will add a try/catch block which allows for handling of exceptions. The try statement will always be executed, and if an exception is raised, the code within the catch will be executed. Note that it is possible within the script editor to use the Javascript throw statement. This is particularly helpful for handling errors, and raising user friendly exceptions which can be shown during run-time. For example: try { // Add Code here } catch(err) { // Add error handling here throw('This is a custom exception thrown from within my custom service');}
Documentation link: https://www.w3schools.com/js/js_errors.asp
Example usage:

```
try {
loggr.warn("incorrect logger statement");
} catch(err) {
logger.error("Error calling function");
}
```

2. Infotable

2.1. Add Field Definition

Description: Used to add a fieldDefinition to a JSON infotable. This will have the effect on adding a new column on the definition of an infotable.
Documentation link: https://community.thingworx.com/docs/DOC-2380
Example Usage (use this in a service with the result type JSON):

```
var result =  me.GetPropertyDefinitions({
        category: undefined /* STRING */,
        type: undefined /* BASETYPENAME */,
        dataShape: undefined /* DATASHAPENAME */
});
result=result.ToJSON();
result.dataShape.fieldDefinitions['MyNewField'] = { name: 'MyNewField', baseType: 'STRING'};
```

2.2. Add Row[]

Description: This allows adding a new row to a JSON infotable.
Documentation link: https://community.thingworx.com/docs/DOC-2380
Example Usage (use this in a service with the result type JSON):

```
var result =  me.GetPropertyDefinitions({
        category: undefined /* STRING */,
        type: undefined /* BASETYPENAME */,
        dataShape: undefined /* DATASHAPENAME */
});
result=result.ToJSON();

var row = new Object();

row.isReadOnly=false;
row.isPersistent=false;
row.isLogged=false;
row.description="testvalue";
row.baseType="NUMBER";
result.rows.push(row);
```

2.3. AddField(object)

Description: Used to add a fieldDefinition to a non-JSON infotable. These are all the infotables generated by any service who has an output of type INFOTABLE (regardless if the services generate inside of them JSON Infotables).
Documentation link: https://community.thingworx.com/docs/DOC-2380
Example Usage (use this in a service with the result type INFOTABLE):

```
var result =  me.GetPropertyDefinitions({
        category: undefined /* STRING */,
        type: undefined /* BASETYPENAME */,
        dataShape: undefined /* DATASHAPENAME */
});
```

```
var newField = new Object();
newField.name = "MyNewField";
newField.baseType = 'STRING';
result.AddField(newField);
```

2.4. AddRow(object)

Description: This allows adding a new row to a non-JSON infotable. This method achieves the same purpose as the Snippet AddRow[]. The difference is on the type of infotables that it can be executed upon.

Documentation link: https://community.thingworx.com/docs/DOC-2380

Example Usage (use this in a service with the result type INFOTABLE):

```
var result =  me.GetPropertyDefinitions({
        category: undefined /* STRING */,
        type: undefined /* BASETYPENAME */,
        dataShape: undefined /* DATASHAPENAME */
});
var row = new Object();

row.isReadOnly=false;
row.isPersistent=false;
row.isLogged=false;
row.description="testvalue";
row.baseType="NUMBER";

result.AddRow(row);
```

2.5. Create empty JSONObject InfoTable

Description: Creates an empty JSON Infotable

Documentation link: https://community.thingworx.com/docs/DOC-2380

Example Usage :

```
var result = { dataShape: { fieldDefinitions : {} }, rows: [] };
//after You create the Infotable, you can use any of the methods that work with JSON Infotables, like the one below
result.dataShape.fieldDefinitions['MyNewField'] = { name: 'MyNewField', baseType: 'STRING'};
```

2.6. Create infotable entry from datashape

Description: This allows creating a new row object, based on an existing DataShape, that can be then added to an Infotable, DataTable or Stream.

Documentation link:

Example Usage (we create an Infotable based on a DataShape, generate one row based on the same DataShape and then add the row to the Infotable; service output must be INFOTABLE):

```
var params = {
  infoTableName : "InfoTable",
  dataShapeName : "AlertDefinition"
};
```

```
// CreateInfoTableFromDataShape(infoTableName:STRING("InfoTable"),
dataShapeName:STRING):INFOTABLE(AlertDefinition)
var result = Resources["InfoTableFunctions"].CreateInfoTableFromDataShape(params);

// The row below is generated if we select the AlertDefinition Datashape

// AlertDefinition entry object
var newEntry = new Object();
newEntry.alertAttributes = undefined; // INFOTABLE
newEntry.alertType = undefined; // STRING
newEntry.name = undefined; // STRING [Primary Key]
newEntry.description = undefined; // STRING
newEntry.priority = undefined; // INTEGER
newEntry.enabled = undefined; // BOOLEAN

result.AddRow(newEntry);
```

2.7. Create infotable from datashape

Description: this will generate an empty non-JSON Infotable, based on an existing DataShape. You can use on it any of the methods that work with non-JSON Infotables.
Documentation link:
Example Usage (we create an Infotable based on a DataShape, generate one row based on the same DataShape and then add the row to the Infotable; service output must be INFOTABLE):

```
var params = {
   infoTableName : "InfoTable",
   dataShapeName : "AlertDefinition"
};

// CreateInfoTableFromDataShape(infoTableName:STRING("InfoTable"),
dataShapeName:STRING):INFOTABLE(AlertDefinition)
var result = Resources["InfoTableFunctions"].CreateInfoTableFromDataShape(params);

// The row below is generated if we select the AlertDefinition Datashape

// AlertDefinition entry object
var newEntry = new Object();
newEntry.alertAttributes = undefined; // INFOTABLE
newEntry.alertType = undefined; // STRING
newEntry.name = undefined; // STRING [Primary Key]
newEntry.description = undefined; // STRING
newEntry.priority = undefined; // INTEGER
newEntry.enabled = undefined; // BOOLEAN

result.AddRow(newEntry);
```

2.8. Create query for an Infotable

Description: Creates a query object that can be applied to an infotable or can be passed as a parameter to any Service which needs a Query type input. This is the way in which you can perform SQL-like filtering in ThingWorx.

Example Usage (we retrieve all the Application Log entried in the platform, then we create a Query object that will display only Error entries, then apply that Query object to the Infotable)

```
// result: INFOTABLE dataShape: "LogEntry"
var result =  Logs["ApplicationLog"].QueryLogEntries({
        maxItems: undefined /* NUMBER */,
        searchExpression: undefined /* STRING */,
        fromLogLevel: undefined /* STRING */,
        instance: undefined /* STRING */,
        endDate: undefined /* DATETIME */,
        origin: undefined /* STRING */,
        thread: undefined /* STRING */,
        ascendingSearch: undefined /* BOOLEAN */,
        oldestFirst: undefined /* BOOLEAN */,
        toLogLevel: undefined /* STRING */,
        user: undefined /* USERNAME */,
        sortFieldName: undefined /* STRING */,
        startDate: undefined /* DATETIME */
});

var query = {
  "filters": {
    "fieldName": "level",
    "type": "LIKE",
    "value": "*Error*"
  }
};

var params = {
        t: result /* INFOTABLE */,
        query: query /* QUERY */
};

// result: INFOTABLE
var result = Resources["InfoTableFunctions"].Query(params);
```

2.9. Delete rows by value filter

Description: Deletes all rows from an Infotable based on the row that is passed as a parameter. This works with all non-JSON Infotables.

In order to use this snippet you must create a row, populating the the columns that you want to use as a filter for the deletion mechanism.

The snippet will delete all the rows that match all the column values that you have inserted.

It is not necessary to define all columns from that Datashape, but only the ones you need to filter.

Example Usage (we query the Application Log then delete all rows which match the delete row,service result must be type INFOTABLE):

```
// result: INFOTABLE dataShape: "LogEntry"
var result =  Logs["ApplicationLog"].QueryLogEntries({
    maxItems: undefined /* NUMBER */,
    searchExpression: undefined /* STRING */,
    fromLogLevel: undefined /* STRING */,
    instance: undefined /* STRING */,
    endDate: undefined /* DATETIME */,
    origin: undefined /* STRING */,
    thread: undefined /* STRING */,
    ascendingSearch: undefined /* BOOLEAN */,
    oldestFirst: undefined /* BOOLEAN */,
    toLogLevel: undefined /* STRING */,
    user: undefined /* USERNAME */,
    sortFieldName: undefined /* STRING */,
    startDate: undefined /* DATETIME */
});
// LogEntry entry object
var newEntry = new Object();
newEntry.content = "missing ; before statement (Line#1)"; // STRING
newEntry.origin="E.c.t.r.a.ScriptServices";

result.Delete(newEntry);
```

Example 2 (we delete all rows with level WARN):

```
// result: INFOTABLE dataShape: "LogEntry"
var result =  Logs["ApplicationLog"].QueryLogEntries({
    maxItems: undefined /* NUMBER */,
    searchExpression: undefined /* STRING */,
    fromLogLevel: undefined /* STRING */,
```

```
    instance: undefined /* STRING */,
    endDate: undefined /* DATETIME */,
    origin: undefined /* STRING */,
    thread: undefined /* STRING */,
    ascendingSearch: undefined /* BOOLEAN */,
    oldestFirst: undefined /* BOOLEAN */,
    toLogLevel: undefined /* STRING */,
    user: undefined /* USERNAME */,
    sortFieldName: undefined /* STRING */,
    startDate: undefined /* DATETIME */
});
// LogEntry entry object
var newEntry = new Object();
newEntry.level="WARN";

result.Delete(newEntry);
```

2.10. **Filter**

Description: Filters an Infotable based on the row that is passed as a parameter. This works with all non-JSON Infotables.

In order to use this snippet you must create a row, populating the the columns that you want to use as a filter.

The snippet will delete all the rows from the infotable which do not match all the column values that you have populated. The result will be that the infotable on which you apply the snippet will contain only the rows that match the conditions.

It is not necessary to define all columns from that Datashape, but only the ones you need to filter.

This filtering mechanism can achieve similar results to the query mechanism. Queries are more powerfull because you can construct more powerfull filters that you can't do in the Filter snippet.

Example Usage (**we query the Application Log then filter and display all rows which match the filter row,service result must be type INFOTABLE**:

```
// result: INFOTABLE dataShape: "LogEntry"
var result =  Logs["ApplicationLog"].QueryLogEntries({
    maxItems: undefined /* NUMBER */,
    searchExpression: undefined /* STRING */,
```

```
    fromLogLevel: undefined /* STRING */,
    instance: undefined /* STRING */,
    endDate: undefined /* DATETIME */,
    origin: undefined /* STRING */,
    thread: undefined /* STRING */,
    ascendingSearch: undefined /* BOOLEAN */,
    oldestFirst: undefined /* BOOLEAN */,
    toLogLevel: undefined /* STRING */,
    user: undefined /* USERNAME */,
    sortFieldName: undefined /* STRING */,
    startDate: undefined /* DATETIME */
});
// LogEntry entry object
var newEntry = new Object();
newEntry.level="WARN";

result.Filter(newEntry);
```

2.11. Find row by value filter

Description: This snippet returns a Value Collection (JSON) that contains the first row that matches all the conditions from the input row.

Works with all non-JSON infotables.

It is not necessary to define all columns from that Datashape, but only the ones you need to filter.

This searching mechanism can achieve similar results to the query mechanism. Queries are more powerfull because you can construct more powerfull filters that you can't do in the Find snippet.

Example usage (we get the Application Log and then we find the first row which has the level „WARN"; use this in a service with basetype JSON):

```
// result: INFOTABLE dataShape: "LogEntry"
var result =  Logs["ApplicationLog"].QueryLogEntries({
    maxItems: undefined /* NUMBER */,
    searchExpression: undefined /* STRING */,
    fromLogLevel: undefined /* STRING */,
    instance: undefined /* STRING */,
```

```
    endDate: undefined /* DATETIME */,
    origin: undefined /* STRING */,
    thread: undefined /* STRING */,
    ascendingSearch: undefined /* BOOLEAN */,
    oldestFirst: undefined /* BOOLEAN */,
    toLogLevel: undefined /* STRING */,
    user: undefined /* USERNAME */,
    sortFieldName: undefined /* STRING */,
    startDate: undefined /* DATETIME */
});
// LogEntry entry object
var newEntry = new Object();
newEntry.level="WARN";

var result=result.Find(newEntry);
```

2.12. getRow(rowIndex)

Description: this returns a row from an existing infotable based on the used Index.

This works with non-JSON based infotables.

Example usage (we get the Application Log, then we retrieve the row with index 0):

```
// result: INFOTABLE dataShape: "LogEntry"
var result =  Logs["ApplicationLog"].QueryLogEntries({
    maxItems: undefined /* NUMBER */,
    searchExpression: undefined /* STRING */,
    fromLogLevel: undefined /* STRING */,
    instance: undefined /* STRING */,
    endDate: undefined /* DATETIME */,
    origin: undefined /* STRING */,
    thread: undefined /* STRING */,
    ascendingSearch: undefined /* BOOLEAN */,
    oldestFirst: undefined /* BOOLEAN */,
    toLogLevel: undefined /* STRING */,
    user: undefined /* USERNAME */,
```

```
    sortFieldName: undefined /* STRING */,
    startDate: undefined /* DATETIME */
});
// LogEntry entry object
var result = result.getRow(0);
```

2.13. **getRowCount**()

Description: this snippet returns the row count for a non-JSON Infotable. In cases you see the return as 500, please note that most of the platform services returning Infotables have a maxItems parameters, which is 500 if it's not set. To get an accurate count in such cases you would need to increase the maxItems to a higher value.

Example usage (we get the Application Log, with the maxItems set to 1000000, then we get the count of entries that are present in the log):

```
// result: INFOTABLE dataShape: "LogEntry"
var result =  Logs["ApplicationLog"].QueryLogEntries({
    maxItems: 1000000 /* NUMBER */,
    searchExpression: undefined /* STRING */,
    fromLogLevel: undefined /* STRING */,
    instance: undefined /* STRING */,
    endDate: undefined /* DATETIME */,
    origin: undefined /* STRING */,
    thread: undefined /* STRING */,
    ascendingSearch: undefined /* BOOLEAN */,
    oldestFirst: undefined /* BOOLEAN */,
    toLogLevel: undefined /* STRING */,
    user: undefined /* USERNAME */,
    sortFieldName: undefined /* STRING */,
    startDate: undefined /* DATETIME */
});
// LogEntry entry object
var result = result.getRowCount();
```

2.14. Infotable for loop

Description: this will insert a scheleton for loop for an infotable where you will need to modify only the Infotable variable name. Used when you need to do processing based on each of the row items.

Example Usage (we get the first 5 entries from the Application Log, then we log the Content of each of the entries in the Script log)

```
// result: INFOTABLE dataShape: "LogEntry"
var result =  Logs["ApplicationLog"].QueryLogEntries({
    maxItems: 5 /* NUMBER */,
    searchExpression: undefined /* STRING */,
    fromLogLevel: undefined /* STRING */,
    instance: undefined /* STRING */,
    endDate: undefined /* DATETIME */,
    origin: undefined /* STRING */,
    thread: undefined /* STRING */,
    ascendingSearch: undefined /* BOOLEAN */,
    oldestFirst: undefined /* BOOLEAN */,
    toLogLevel: undefined /* STRING */,
    user: undefined /* USERNAME */,
    sortFieldName: undefined /* STRING */,
    startDate: undefined /* DATETIME */
});
var tableLength = result.rows.length;
for (var x=0; x < tableLength; x++) {
  var row = result.rows[x];
  logger.warn(row.content);
}
```

2.15. Iterate infotable datashape fields

Description: This snippet iterates through the fields of a datashape from an Infotable. It works with both JSON and non-JSON based infotables. It is useful when trying to discover Datashape fields from Infotables you don't know their datashape in advance (eg: when they are generated dynamically in another service and then they are passed as a parameter to your service).

Example Usage (we get the Application Log entries as an Infotable then we iterate through each of the fields from that Datashape and log the Field Name and Basetype in the Script log):

```
// result: INFOTABLE dataShape: "LogEntry"
```

```
var result =  Logs["ApplicationLog"].QueryLogEntries({
    maxItems: undefined /* NUMBER */,
    searchExpression: undefined /* STRING */,
    fromLogLevel: undefined /* STRING */,
    instance: undefined /* STRING */,
    endDate: undefined /* DATETIME */,
    origin: undefined /* STRING */,
    thread: undefined /* STRING */,
    ascendingSearch: undefined /* BOOLEAN */,
    oldestFirst: undefined /* BOOLEAN */,
    toLogLevel: undefined /* STRING */,
    user: undefined /* USERNAME */,
    sortFieldName: undefined /* STRING */,
    startDate: undefined /* DATETIME */
});
// infotable datashape iteration
var dataShapeFields = result.dataShape.fields;
for (var fieldName in dataShapeFields) {
   logger.warn('field name is ' + dataShapeFields[fieldName].name);
   logger.warn('field basetype is ' + dataShapeFields[fieldName].baseType);
}
```

2.16. **RemoveAllRows**()

Description: This removes all the rows from an non-JSON Infotable. The result will be an empty Infotable, but with the same DataShape.

Example Usage (we get the Application Log entries as an Infotable, then we remove all the rows from the resulting Infotable):

```
// result: INFOTABLE dataShape: "LogEntry"
var result =  Logs["ApplicationLog"].QueryLogEntries({
    maxItems: undefined /* NUMBER */,
    searchExpression: undefined /* STRING */,
    fromLogLevel: undefined /* STRING */,
    instance: undefined /* STRING */,
```

```
    endDate: undefined /* DATETIME */,
    origin: undefined /* STRING */,
    thread: undefined /* STRING */,
    ascendingSearch: undefined /* BOOLEAN */,
    oldestFirst: undefined /* BOOLEAN */,
    toLogLevel: undefined /* STRING */,
    user: undefined /* USERNAME */,
    sortFieldName: undefined /* STRING */,
    startDate: undefined /* DATETIME */
});
// infotable datashape iteration
result.RemoveAllRows();
```

2.17. RemoveField(fieldName)

Description: Removes a Column from an Infotable by passing the name of the Column. Works only with non-JSON Infotables. Note that the column values from each row are also lost, as it does not only removes the DataShape.

Example Usage (We get the Application Log, then we remove the „content" column from it):

```
// result: INFOTABLE dataShape: "LogEntry"
var result =  Logs["ApplicationLog"].QueryLogEntries({
    maxItems: undefined /* NUMBER */,
    searchExpression: undefined /* STRING */,
    fromLogLevel: undefined /* STRING */,
    instance: undefined /* STRING */,
    endDate: undefined /* DATETIME */,
    origin: undefined /* STRING */,
    thread: undefined /* STRING */,
    ascendingSearch: undefined /* BOOLEAN */,
    oldestFirst: undefined /* BOOLEAN */,
    toLogLevel: undefined /* STRING */,
    user: undefined /* USERNAME */,
    sortFieldName: undefined /* STRING */,
    startDate: undefined /* DATETIME */
```

```
});
// infotable datashape iteration
result.RemoveField("content");
```

2.18. RemoveRow(rowIndex)

Description: This service will remove a specific row from a non-JSON Infotable based on the provided index.

Example Usage (get the Application Log Entries then remove the row with index 0):

```
// result: INFOTABLE dataShape: "LogEntry"
var result =  Logs["ApplicationLog"].QueryLogEntries({
    maxItems: undefined /* NUMBER */,
    searchExpression: undefined /* STRING */,
    fromLogLevel: undefined /* STRING */,
    instance: undefined /* STRING */,
    endDate: undefined /* DATETIME */,
    origin: undefined /* STRING */,
    thread: undefined /* STRING */,
    ascendingSearch: undefined /* BOOLEAN */,
    oldestFirst: undefined /* BOOLEAN */,
    toLogLevel: undefined /* STRING */,
    user: undefined /* USERNAME */,
    sortFieldName: undefined /* STRING */,
    startDate: undefined /* DATETIME */
});
// infotable datashape iteration
result.RemoveRow(0);
```

2.19. Sort

Description: this will sort a non-JSON Infotable based on the provided parameter, which contains a field name and the direction for sorting.

Example (we get the Application Log entries, then we Sort based on the column named „content" in ascending direction):

```
// result: INFOTABLE dataShape: "LogEntry"
```

```
var result =  Logs["ApplicationLog"].QueryLogEntries({
    maxItems: undefined /* NUMBER */,
    searchExpression: undefined /* STRING */,
    fromLogLevel: undefined /* STRING */,
    instance: undefined /* STRING */,
    endDate: undefined /* DATETIME */,
    origin: undefined /* STRING */,
    thread: undefined /* STRING */,
    ascendingSearch: undefined /* BOOLEAN */,
    oldestFirst: undefined /* BOOLEAN */,
    toLogLevel: undefined /* STRING */,
    user: undefined /* USERNAME */,
    sortFieldName: undefined /* STRING */,
    startDate: undefined /* DATETIME */
});
// infotable datashape iteration

var sort = new Object();
sort.name = "content";
sort.ascending = true;
result.Sort(sort);
```

2.20. **ToJSON**()

Description: this returns a non-JSON Infotable as a JSON object. Does not work with JSON based Infotables. You can then use all snippets which work only with JSON Infotables (eg: 2.1 Add Field Definition and 2.2 AddRow[]). You can use this snippet to pass the infotable to an external REST service which accepts a JSON parameter. It is also the only way to add dynamically field definitions to an Infotable.

Example (We get the Property Definitions for the parent Thing and we convert the resulting infoTable to JSON; to use in a service with output type JSON):

```
var result =  me.GetPropertyDefinitions({
    category: undefined /* STRING */,
    type: undefined /* BASETYPENAME */,
    dataShape: undefined /* DATASHAPENAME */
```

```
});
result=result.ToJSON();
```

3. Logger

3.1. debug

Description: this will insert DEBUG level log messages to the ScriptLog. Calling this function will write the message directly to the disk in the ThingworxStorage/Logs/ScriptLog.log file. High rate of calls to any of the Logger functions will impact the application.

Depending on the Script Log level setting it is possible the entry will not be written to disk.

The standard Script Log is WARN, so TRACE, INFO, DEBUG calls won't be seen&written in the log OOTB.

Example Usage (Set ScriptLog to ALL, TRACE or DEBUG to see the entry):

```
logger.debug("This is a debug message");
```

3.2. error

Description: this will insert ERROR level log messages to the ScriptLog. Calling this function will write the message directly to the disk in the ThingworxStorage/Logs/ScriptLog.log file. High rate of calls to any of the Logger functions will impact the application.

Example Usage:

```
logger.error("This is an error message");
```

3.3. info

Description: this will insert INFO level log messages to the ScriptLog. Calling this function will write the message directly to the disk in the ThingworxStorage/Logs/ScriptLog.log file. High rate of calls to any of the Logger functions will impact the application.

Depending on the Script Log level setting it is possible the entry will not be written to disk.

The standard Script Log is WARN, so TRACE, INFO, DEBUG calls won't be seen&written in the log OOTB.

Example Usage (Set ScriptLog to ALL, TRACE, DEBUG or INFO to see the entry):

```
logger.info("This is an info message");
```

3.4. warn

Description: this will insert WARN level log messages to the ScriptLog. Calling this function will write the message directly to the disk in the ThingworxStorage/Logs/ScriptLog.log file. High rate of calls to any of the Logger functions will impact the application.

Depending on the Script Log level setting it is possible the entry will not be written to disk.

The standard Script Log is WARN, so this will be seen&written in the log OOTB.

Example Usage (Set ScriptLog to ALL, TRACE, DEBUG, INFO or WARN to see the entry):

```
logger.warn("This is a warn message");
```

4. Stream, Blog, DataTable

4.1. Add Blog Entry

Description: adds an Entry to a Blog that already exists in the Platform. It calls the AddBlogEntry method of the Blog, so you can use that with the same effect, but this snippet provides additional constructors for some parameters like location, timestamp, tags which makes it easier to consume that function.

Example usage:

```
// tags:TAGS
var tags = new Array();

// timestamp:DATETIME
var timestamp = new Date();

// location:LOCATION
var location = new Object();
location.latitude = 0;
location.longitude = 0;
location.elevation = 0;
location.units ="WGS84";

var params = {
  tags : tags,
  title : "Enter Title Here",
  content : "Enter Content Here",
  timestamp : timestamp,
  source : me.name,
  location : location
};
```

```
// AddBlogEntry(tags:TAGS, title:STRING("Enter Title Here"), content:STRING(""), timestamp:DATETIME, source:STRING("me.name"), location:LOCATION):STRING

var result = Things["TestBlog"].AddBlogEntry(params);
```

4.2. Add Stream Entry

Description: adds an Entry to a Stream that already exists in the Platform. It calls the AddStreamEntry method of the Stream, so you can use that with the same effect, but this snippet provides additional constructors for some parameters like location, timestamp, tags which makes it easier to consume that function. The snippet will generate the row fields based on the DataShape assigned and you need to provide values for those columns.

When calling the Add Stream Entry, the entry itself will be lazy written to disk. As a consequence, a query executed immediately after this will not return the entry that was just inserted.

Example Usage (for a Stream called „TestStream", with the „ActiveDirectoryDomainGroups" Datashape):

```
// tags:TAGS

var tags = new Array();


// timestamp:DATETIME

var timestamp = new Date()


// values:INFOTABLE(Datashape:ActiveDirectoryDomainGroups)

var values = Things["TestStream"].CreateValues();


values.activeDirectoryGroupName = „MyNewValue"; //STRING


// location:LOCATION

var location = new Object();

location.latitude = 0;

location.longitude = 0;

location.elevation = 0;

location.units ="WGS84";


var params = {

   tags : tags,
```

```
    timestamp : timestamp,
    source : me.name,
    values : values,
    location : location
};

// AddStreamEntry(tags:TAGS, timestamp:DATETIME, source:STRING("me.name"), values:INFOTABLE(ActiveDirectoryDomainGroups), location:LOCATION):NOTHING
Things["TestStream"].AddStreamEntry(params);
```

4.3. Add/Update Data Table

Description: adds or updates a Data Row in a Data Table that already exists in the Platform. It calls the AddOrUpdateDataTableEntry method of the DataTable, so you can use that with the same effect, but this snippet provides additional constructors for some parameters like location, timestamp, tags which makes it easier to consume that function. The snippet will generate the row fields based on the DataShape assigned to the DataTable and you need to provide values for those columns.

The method will return the entry ID from the Persistance Provider database after run (in case of PostgreSQL, this will be an incrementing Integer).

Remember that the Primary Key from this entry needs to be unique across a DataTable.

Each time when this method is called the timestamp will get updated.

When updating, if you don't specify values, as seen in Example 2, they will be left intact in the DataTable, there will be no overwriting. Also, in the second example, you must provide the ID (or whatever the Primary Key is in your case).

Example Usage 1 (this inserts a row in a DataTable called „TestDataTable", with a Data Shape called „TestDataShape", which has 3 Fields: ID (type GUID), Content (type STRING) and NumberContent (type NUMBER); use this in a service with return type STRING):

```
// tags:TAGS
var tags = new Array();

// values:INFOTABLE(Datashape: TestDataShape)
var values = Things["TestDataTable"].CreateValues();
values.Content = "AddedContent"; // STRING
values.ID =  generateGUID(); // GUID [Primary Key]
values.NumberContent = 50; // NUMBER
```

```
// location:LOCATION
var location = new Object();
location.latitude = 0;
location.longitude = 0;
location.elevation = 0;
location.units ="WGS84";

var params = {
    tags : tags,
    source : me.name,
    values : values,
    location : location
};

// AddOrUpdateDataTableEntry(tags:TAGS, source:STRING("me.name"), values:INFOTABLE(TestDataTable), location:LOCATION):STRING
var result = Things["TestDataTable"].AddOrUpdateDataTableEntry(params);
```

Example Usage 2 (this updates a row in a DataTable called „TestDataTable", with a Data Shape called „TestDataShape", which has 3 Fields: ID (type GUID), Content (type STRING) and NumberContent (type NUMBER); use this in a service with return type STRING):

```
// tags:TAGS
var tags = new Array();

// values:INFOTABLE(Datashape: TestDataShape)
var values = Things["TestDataTable"].CreateValues();
values.Content = "UpdatedContent"; // STRING
values.ID =  "d51e7163-c771-43ba-949e-a6c41396d3a7"; // GUID [Primary Key]


// location:LOCATION
```

```
var location = new Object();
location.latitude = 5;
location.longitude = 0;
location.elevation = 0;
location.units ="WGS84";

var params = {
    tags : tags,
    source : me.name,
    values : values,
    location : location
};

// AddOrUpdateDataTableEntry(tags:TAGS, source:STRING("me.name"), values:INFOTABLE(TestDataTable), location:LOCATION):STRING
var result = Things["TestDataTable"].AddOrUpdateDataTableEntry(params);
```

4.4. Create query for DataTable

Description: This snippet allows creating a Query for a DataTable. If works almost exactly the same as the „Create query for Infotable" or the „Create query for Stream", except that it will allow selecting a DataTable from the existing DataTable entities in the system.

For example, you can have the same output if you choose the „Create query for an Infotable" and you select the DataShape assigned to that DataTable.

The returned JSON can be used for any Service which requires a QUERY type parameter.

The snippet does not allow creating nesting filters with different And/Or Types or sort the infotable (which is possible by just using the query object).

In order to do that, please consult the documentation ThingWorx Model Definition and Composer > Things > Thing Services > Query Parameter for Query Services.

4.5. Create query for Stream

Description: This snippet allows creating a Query for a Stream. If works almost exactly the same as the „Create query for Infotable" or the „Create query for DataTable", except that it will allow selecting a Stream from the existing Stream entities in the system.

For example, you can have the same output if you choose the „Create query for an Infotable" and you select the DataShape assigned to that Stream.

The returned JSON can be used for any Service which requires a QUERY type parameter.

The snippet does not allow creating nesting filters with different And/Or Types or sorting the infotable (which is possible by just using the query object).

In order to do that, please consult the documentation ThingWorx Model Definition and Composer > Things > Thing Services > Query Parameter for Query Services

5. Tags

5.1. Data Tag Picker

Description: This allows inserting an existing Data Tag into server side code. Data Tags are used specifically for Stream, DataTable, Blog and Wiki Entries. They are not used to tag entities.

Note: if the data Tag Vocabulary does not exist, then the service will fail. If the Data Tag Term does not exist, and the Vocabulary is Dynamic, the service will succeed, and the Tag will be added to the Vocabulary.

Example usage:

```
// tags:TAGS

//the entry in yellow was inserted by the Data Tag Picker

var tags = "TestDataTag:TestDataTag";


// timestamp:DATETIME

var timestamp = new Date();


// location:LOCATION

var location = new Object();

location.latitude = 0;

location.longitude = 0;

location.elevation = 0;

location.units ="WGS84";


var params = {

  tags : tags,

  title : "Enter Title Here",

  content : "Enter Content Here",

  timestamp : timestamp,

  source : me.name,

  location : location

};
```

```
// AddBlogEntry(tags:TAGS, title:STRING("Enter Title Here"), content:STRING(""), timestamp:DATETIME, source:STRING("me.name"), location:LOCATION):STRING

var result = Things["TestBlog"].AddBlogEntry(params);
```

5.2. Model Tag Picker

Description: This allows inserting an existing Model Tag in server side code. Model Tags are used specifically for tagging any Entity. They are not used to tag entries from DataTables, Streams, Blogs or Wikis.

Note: if the Model Tag Vocabulary does not exist, the service will fail. If the Model Tag Term does not exist, and the Vocabulary is Dynamic, the service will succeed, and the Tag will be added to the Vocabulary.

Example usage:

```
me.AddTags({
    tags: "Applications:Test" /* TAGS */
});
```

6. DateFunctions

These snippets are extremely valuable when working with DateTime values in ThingWorx.

6.1. dateAddDays

Description: Adds or substracts a specified number of days to a datetime value. If the added value is a non-natural number, like 1.7, the snippet will disregard the decimal part.

Example usage (we get the current date and we add 2 days to it; use this in a service with return type DATETIME):

```
// dateValue:DATETIME

var dateValue = new Date();


// dateAddDays(dateValue:DATETIME, amount:NUMBER):STRING

var result = dateAddDays(dateValue, 2);
```

6.2. dateAddHours

Description: Adds or substracts a specified number of hours to a datetime value. If the added value is a non-natural number, like 1.7, the snippet will disregard the decimal part

Example usage (we get the current date and we add 2 hours to it; use this in a service with return type DATETIME):

```
// dateValue:DATETIME

var dateValue = new Date();


// dateAddHours(dateValue:DATETIME, amount:NUMBER):STRING

var result = dateAddHours(dateValue, 2);
```

6.3. dateAddMilliseconds

Description: Adds or substracts a specified number of milliseconds to a datetime value. If the added value is a non-natural number, like 1.7, the snippet will disregard the decimal part.

Example usage (we get the current date and we add 2 milliseconds to it; use this in a service with return type DATETIME):

```
// dateValue:DATETIME

var dateValue = new Date();
```

```
// dateAddMilliseconds(dateValue:DATETIME, amount:NUMBER):STRING

var result = dateAddMilliseconds(dateValue, 2);
```

6.4. dateAddMinutes

Description: Adds or substracts a specified number of minutes to a datetime value. If the added value is a non-natural number, like 1.7, the snippet will disregard the decimal part.

Example usage (we get the current date and we add 2 minutes to it; use this in a service with return type DATETIME):

```
// dateValue:DATETIME

var dateValue = new Date();


// dateAddMinutes(dateValue:DATETIME, amount:NUMBER):STRING

var result = dateAddMinutes(dateValue, 2);
```

6.5. dateAddMonths

Description: Adds or substracts a specified number of months to a datetime value. If the added value is a non-natural number, like 1.7, the snippet will disregard the decimal part.

Example usage (we get the current date and we add 2 months to it; use this in a service with return type DATETIME):

```
// dateValue:DATETIME

var dateValue = new Date();


// dateAddMonths(dateValue:DATETIME, amount:NUMBER):STRING

var resulted = dateAddMonths(dateValue, 2);
```

6.6. dateAddSeconds

Description: Adds or substracts a specified number of seconds to a datetime value. If the added value is a non-natural number, like 1.7, the snippet will disregard the decimal part.

Example usage (we get the current date and we add 2 seconds to it; use this in a service with return type DATETIME):

```
// dateValue:DATETIME
```

```
var dateValue = new Date();


// dateAddSeconds(dateValue:DATETIME, amount:NUMBER):STRING
var result = dateAddSeconds(dateValue, 2);
```

6.7. dateAddYears

Description: Adds or substracts a specified number of seconds to a datetime value. If the added value is a non-natural number, like 1.7, the snippet will disregard the decimal part.

Example usage (we get the current date and we add 2 seconds to it; use this in a service with return type DATETIME):

```
// dateValue:DATETIME
var dateValue = new Date();


// dateAddYears(dateValue:DATETIME, amount:NUMBER):STRING
var result = dateAddYears(dateValue, 2);
```

6.8. dateDayOfWeek

Description: this function returns a number representing the day of the week. Minimum is 1, representing Monday, and Maximum is 7, representing Sunday.

Example usage (this service uses the current date as an input parameter; use this in a service with return type NUMBER):

```
// dateValue:DATETIME
var dateValue = new Date();


// dateDayOfWeek(dateValue:DATETIME):NUMBER
var result = dateDayOfWeek(dateValue);
```

6.9. dateDayOfYear

Description: this function returns a number representing the day of the year, with a minimum of 0 and maximum of 365

Example usage: (this service uses the current date as an input parameter; use this in a service with return type NUMBER):

```
// dateValue:DATETIME
var dateValue = new Date();

// dateDayOfYear(dateValue:DATETIME):NUMBER
var result = dateDayOfYear(dateValue);
```

6.10. dateDifference

Description: returns the difference in milliseconds between 2 dates

Example Usage (we calculate the difference between 2 dates: date1 is the current date to which we add 1000 milliseconds, and date2 is the current date; use this in a service with result type NUMBER; result should be 1000):

```
var inputDate = new Date();

// date1:DATETIME
var date1 = dateAddMilliseconds(inputDate, 1000);

// date2:DATETIME
var date2 = inputDate;

// dateDifference(date1:DATETIME, date2:DATETIME):NUMBER
var result = dateDifference(date1, date2);
```

6.11. dateFormat

Description: returns a datetime as a string, formatted in a specific way

The format used is the one from the joda library, and specific examples can be seen at http://www.joda.org/joda-time/apidocs/org/joda/time/format/DateTimeFormat.html

Example Usage (we get the current date and we output as a string, formatted in a specific format):

```
// dateValue:DATETIME
var dateValue = new Date();
```

```
// dateFormat(dateValue:DATETIME, dateFormat:STRING):STRING

var result = dateFormat(dateValue, "yyyy-MM-dd HH:mm:ss");
```

6.12 dateFormatISO

Description: returns a Datetime as a string, formatted in the https://en.wikipedia.org/wiki/ISO_8601 format.

It does not have any input parameter. The output format is locked as „Combined date and time in UTC": 2017-09-11T13:49:34+00:00

Example usage: (we get the current datetime and we display it as a string, formatted in the ISO8601 format;use this in a service with result type DATETIME):

```
// dateValue:DATETIME

var dateValue = new Date();


// dateFormatISO(dateValue:DATETIME):STRING

var result = dateFormatISO(dateValue);
```

6.13 dateMax

Description: the dateMax snippet returns the maximum of 2 dates, in a number format. Thingworx will automatically convert it to Date if you set the result type as DateTime

Example usage (we create 2 Date objects, the first being a Date one hour in advance, and the second being the current date; use this in a service with result type DATETIME):

```
var date1 = dateAddHours(new Date(), 1);

var date2 = new Date();

// dateMax(date1:DATETIME, date2:DATETIME):NUMBER

var result = dateMax(date1, date2);
```

6.14 dateMin

Description: the dateMin snippet returns the minimum of 2 dates, in a number format. Thingworx will automatically convert it to Date if you set the result type as DateTime

Example usage (we create 2 Date objects, the first being a Date one hour in advance, and the second being the current date; use this in a service with result type DATETIME):

```
var date1 = dateAddHours(new Date(), 1);

var date2 = new Date();

// dateMin(date1:DATETIME, date2:DATETIME):NUMBER

var result =dateMin(date1, date2);
```

6.15 dateOffset

Description: the dateOffset snippet returns the offset since the UTC time for a date.

Example usage 1 (we create a new Date object and we get the offset in miliseconds since the UTC time. As the server timezone is UTC+3 and the new Date will be in that timezone, the snippet will return 10800000 = 3 hrs):

```
// dateValue:DATETIME

var dateValue = new Date();


// dateOffset(dateValue:DATETIME):NUMBER

var result = dateOffset(dateValue);
```

6.16 parseDate

Description: parses a string and outputs a datetime value, based on a specific format.

The format used is the one from the joda library, and specific examples can be seen at http://www.joda.org/joda-time/apidocs/org/joda/time/format/DateTimeFormat.html

Example Usage 1 (we parse an input string based on a specific dateFormat):

```
var result = parseDate("2017-09-22 13:00:15", "yyyy-MM-dd HH:mm:ss");
```

6.17 parseDateISO

Description: parses a string and outputs a datetime value, trying to match the ISO8601 format.

Example usage 1 (we parse an input string and we return a datetime value)

```
var result = parseDateISO("2017-09-11T13:49:34+00:00");
```

7. DiagnosticFunctions

7.1 disableDiagnosticTrace

7.2 enableDiagnosticTrace

Description: this snippet disables/enables verbose script tracing in the context of the thread which executes the service. This will not effect other scripts verbose level.

The information will be written into the Script Log only if the Script Log has SCRIPT, TRACE or LOG level It will detail all the script parameters. The information is **not** the same as the information offered by setting ApplicationLog level to TRACE, DEBUG or ALL.

Timestamp	Level	Content
2017-10-15 17:14:07.709	**DEBUG**	**Service QueryLogEntries on ApplicationLog has a return type of INFOTABLE**
2017-10-15 17:14:07.709	DEBUG	Calling base function Filter with 1 parameters com.thingworx.types.InfoTable
2017-10-15 17:14:07.709	DEBUG	BaseFunction argument was a Native Object in Filter
2017-10-15 17:14:07.709	DEBUG	Calling BaseFunction Filter with JSONObject {"level":"WARN"}
2017-10-15 17:14:07.708	DEBUG	Service QueryLogEntries on ApplicationLog had a raw result value of {"rows":[{"instance":"","level":"DEBUG","session":"","ori
2017-10-15 17:14:07.700	DEBUG	When calling service QueryLogEntries on ApplicationLog parameter oldestFirst had no value assigned
2017-10-15 17:14:07.696	DEBUG	Calling service QueryLogEntries on ApplicationLog
2017-10-15 17:14:07.696	DEBUG	Calling service QueryLogEntries on ApplicationLog with data {}
2017-10-15 17:14:07.696	DEBUG	When calling service QueryLogEntries on ApplicationLog parameter maxItems had no value assigned
2017-10-15 17:14:07.696	DEBUG	When calling service QueryLogEntries on ApplicationLog parameter startDate had no value assigned
2017-10-15 17:14:07.696	DEBUG	When calling service QueryLogEntries on ApplicationLog parameter endDate had no value assigned
2017-10-15 17:14:07.696	DEBUG	When calling service QueryLogEntries on ApplicationLog parameter fromLogLevel had no value assigned
2017-10-15 17:14:07.696	DEBUG	When calling service QueryLogEntries on ApplicationLog parameter toLogLevel had no value assigned
2017-10-15 17:14:07.696	DEBUG	When calling service QueryLogEntries on ApplicationLog parameter user had no value assigned
2017-10-15 17:14:07.696	DEBUG	When calling service QueryLogEntries on ApplicationLog parameter thread had no value assigned
2017-10-15 17:14:07.696	DEBUG	When calling service QueryLogEntries on ApplicationLog parameter origin had no value assigned
2017-10-15 17:14:07.696	DEBUG	When calling service QueryLogEntries on ApplicationLog parameter instance had no value assigned
2017-10-15 17:14:07.696	DEBUG	When calling service QueryLogEntries on ApplicationLog parameter searchExpression had no value assigned
2017-10-15 17:14:07.696	DEBUG	When calling service QueryLogEntries on ApplicationLog parameter sortFieldName had no value assigned

Example of the Diagnostic trace output from the Script Log.

It returns the previous state of the DiagnosticTrace status. For example, if you called first the enableDiagnosticFunction and then the disableDiagnosticTrace, then this function will return true.

Calling the disableDiagnosticTrace() function without calling the enableDiagnosticTracing first will have no effect from the logging point of view.

Example usage (we call first the enableDiagnosticTracing, execute a infotable operation, then we call disableDiagnosticTracing and we display the output):

```
// enableDiagnosticTrace():BOOLEAN
var state = enableDiagnosticTrace();

// result: INFOTABLE dataShape: "LogEntry"
var result =  Logs["ApplicationLog"].QueryLogEntries({
        maxItems: undefined /* NUMBER */,
        searchExpression: undefined /* STRING */,
        fromLogLevel: undefined /* STRING */,
        instance: undefined /* STRING */,
        endDate: undefined /* DATETIME */,
        origin: undefined /* STRING */,
        thread: undefined /* STRING */,
        ascendingSearch: undefined /* BOOLEAN */,
        oldestFirst: undefined /* BOOLEAN */,
        toLogLevel: undefined /* STRING */,
        user: undefined /* USERNAME */,
        sortFieldName: undefined /* STRING */,
        startDate: undefined /* DATETIME */
});
// LogEntry entry object
var newEntry = new Object();
newEntry.level="WARN";

result.Filter(newEntry);



// disableDiagnosticTrace():BOOLEAN
var state = disableDiagnosticTrace();
```

8. EncryptionFunctions

8.1 base64DecodeBytes

Description: this function takes as a parameter

Example usage:

8.2 base64DecodeString

Description: this function takes a Base64 encoded string and decodes it in order to get the original string.

Example usage (we input a base64 encoded string and we return the original string; the input parameter is the base64 encoded string calculated with the function 8.4. base64EncodeString; use this example in a service with STRING return type):

```
// base64DecodeString(value:STRING):STRING

var result = base64DecodeString("ODg4Ly9AQA==");
```

8.3 base64EncodeBytes

Description: this function.....

Example usage:

8.4 base64EncodeString

Description: this function takes a String parameter and encodes it in Base64 format . It returns it as a Base64 String representation.

Example usage (we input a random string and we return the Base64 representatation; the output of this function is used as an input for the 8.2 base64DecodeString); use this example in a service with STRING return type:

```
// base64EncodeString(value:STRING):STRING

var result = base64EncodeString("888//@@");
```

8.5. decryptStringWithkey

Description: this function decrypts an encrypted string into the original value using the specified key. The encrypted value must be the encrypted string obtained with the function 8.8 encryptStringWithKey.

The same key used for encryption must be used (56 characters length, hex characters only).

Example usage (we decrypt an encrypted string value; the encrypted value is the one obtained by using the example from function 8.8. encryptStringWithKey; use this in a service with return type STRING)

```
// decryptStringWithkey(key:STRING, value:STRING):STRING

var result = decryptStringWithkey("99f9f09df09f09ae0fb094f4ceaaaAbcffeddffffeaafaaaeaafaaaa", "0Vxg3+mfMy2k0Igcx959sA==");
```

8.6 encryptPropertyValue

Description: this function....

Example usage:

8.7 encryptString

Description: this function returns the encrypted String value for a String input parameter.

8.8. encryptStringWithKey

Description: this function encrypts a string with specified key. The key must be 56 characters length, all hex.

Example Usage: (we encrypt a string value using a key; the resulting value is the one used as an input parameter for function 8.5 decryptStringWithKey; use this in a service with return type STRING):

```
// encryptStringWithKey(key:STRING, value:STRING):STRING

var result = encryptStringWithKey("99f9f09df09f09ae0fb094f4ceaaaAbcffeddffffeaafaaaeaafaaaa", "ajhkjh");
```

9. LocationFunctions

9.1 containsLocation

Description: this function checks if a location is found in a specified area (called geofence). The location is passed as a LOCATION type parameter. The geofence is specified as an INFOTABLE and you must specify the column name containing the Geofence perimeter point from the infotable.

Example usage (we pass a location with coordinates 35,35 and we check if it's in a rectangle with coordinates 0,0 / 0,40 / 40,40 / 40,0; in order to do this we manually construct the geofence in the service, but in normal use you have the geofence defined as a property; use this in a service with return type BOOLEAN):

```
// location:LOCATION
var location = new Object();
location.latitude = 35;
location.longitude = 35;
location.elevation = 0;
location.units = "WGS84";

// geoFence:INFOTABLE
var geoFence = { dataShape: { fieldDefinitions : {} }, rows: [] };

var params = {
        json: geoFence /* JSON */
};

// result: INFOTABLE
var geoFence = Resources["InfoTableFunctions"].FromJSON(params);



var newField = new Object();
newField.name = "Location";
newField.baseType = 'LOCATION';
geoFence.AddField(newField);
```

```
var row1 = new Object(); row1.latitude=0;row1.longitude=0;row1.elevation=0;
row1.units="WGS84";var loc1 = new Object();loc1.Location=row1;

var row2 = new Object(); row2.latitude=0;row2.longitude=40;row2.elevation=0;
row2.units="WGS84";var loc2 = new Object();loc2.Location=row2;

var row3 = new Object(); row3.latitude=40;row3.longitude=40;row3.elevation=0;
row3.units="WGS84";var loc3 = new Object();loc3.Location=row3;

var row4 = new Object(); row4.latitude=40;row4.longitude=0;row4.elevation=0;
row4.units="WGS84";var loc4 = new Object();loc4.Location=row4;

geoFence.AddRow(loc1);

geoFence.AddRow(loc2);

geoFence.AddRow(loc3);

geoFence.AddRow(loc4);



// containsLocation(location:LOCATION, geoFence:INFOTABLE, locationField:STRING):BOOLEAN

var result = containsLocation(location, geoFence, "Location");
```

8.2. createCircularGeoFence

Description: this snippet creates a circular geofence formatted as an infotable, with the center in a specified location. You can specify the radius, the radius units (can be one of the following:) and the resolution. The resolution is the number of generated points -1. For example, a resolution of 4 will generate 5 rows. The output infotable has a single field, named „location”.

Example usage (we create a circular geofence with the center point 35,35, a radius of 10km and a resolution of 10; we then check if the location 35,35 is found within this geofence; use this in a service with return type BOOLEAN):

```
// location:LOCATION

var location = new Object();

location.latitude = 35;

location.longitude = 35;

location.elevation = 0;

location.units = "WGS84";



// createCircularGeoFence(location:LOCATION, radius:NUMBER, uni:STRING,
resolution:INTEGER):INFOTABLE

var geofence = createCircularGeoFence(location, 1, "km", 10);
```

```
// location:LOCATION
var location = new Object();
location.latitude = 35;
location.longitude = 35;
location.elevation = 0;
location.units = "WGS84";


// containsLocation(location:LOCATION, geoFence:INFOTABLE, locationField:STRING):BOOLEAN
var result = containsLocation(location, geofence, "location");
```

8.3 distanceBetween

Description: this snippet calculates the distance between 2 location type input parameters

Example usage: (we provide 2 locations and we calculate the distance between them; use this in a service with return type NUMBER)

```
// loc1:LOCATION
var loc1 = new Object();
loc1.latitude = 0;
loc1.longitude = 0;
loc1.elevation = 0;
loc1.units = "WGS84";

// loc2:LOCATION
var loc2 = new Object();
loc2.latitude = 0;
loc2.longitude = 30;
loc2.elevation = 0;
loc2.units = "WGS84";
```

```
// distanceBetween(loc1:LOCATION, loc2:LOCATION, units:STRING):NUMBER
var result = distanceBetween(loc1, loc2, "n");
```