Ubisense

SmartSpace®

External Data Connector

From version 3.8.1

Part Number: SS_EDC_3.8.1_EN

Ubisense Limited, St Andrew's House, St Andrew's Road, Cambridge CB4 1DL, United Kingdom.

Telephone: +44 (0)1223 535170. Website: https://www.ubisense.com

Contents

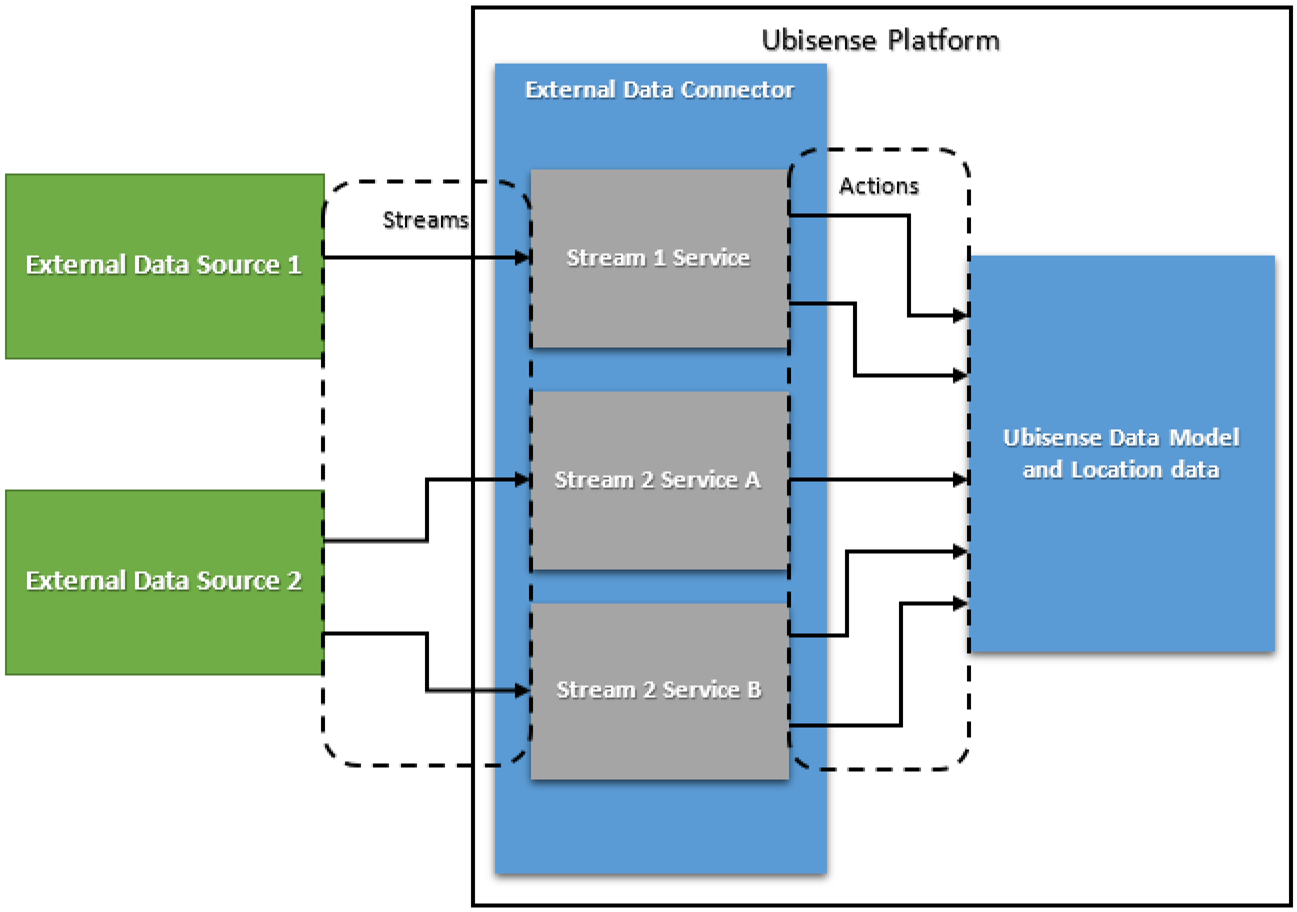
Overview of the External data connector

The External data connector (EDC) is a service for collecting location and/or property data from an external system and injecting it into SmartSpace. It supports a range of protocols for connecting to external customer systems, including HTTP(S) queries (server and client), web sockets (client), TCP (client), SQL (client) and file and allows imports in various formats, including XML, JSON, CSV, and single-field, unformatted data.

Configuration is carried out entirely within the SmartSpace Config application and this guide takes you through the configuration process step by step.

Anatomy of the service

Overview



The External data connector has two main elements, *Streams* and *Actions*:

• A *Stream* is the stream of data from an external data source to the External data connector within the Ubisense platform. A single external source may provide one or more streams.

• An *Action* is an operation that operates on a stream. This operation takes data from the stream as input and sends information to the Ubisense data model or Ubisense Location services as its output. Multiple actions can operate on a single stream.

These elements are configured in SmartSpace Config using object types of the same name. Streams and actions objects can be defined in the TYPES / OBJECTS task in SmartSpace Config.

You can use these objects to define the behavior of your streams and actions in the SERVICE PARAMETERS task.

When you have finished configuring a stream service and are ready to deploy it, you set its enabled parameter to true in the SERVICE PARAMETERS tab. This creates the Ubisense service that manages that stream and its actions. The service is deployed, but you need to start it manually using the Service Manager application.

Glossary

Term	Definition	Example
External data source	A non-Ubisense service or similar source that provides data	HTTP server serving location information in JSON format
Data message	A complete message of one or more data fields	A JSON message, received from a server via a GET request, containing location information
Data field	A datum with some information useful to Actions. Consists of an identity (name) and value in most data formats	A name-value pair within a JSON message, e.g. "name":"object1"
Identity	The name or label of a data field, used to identify it in source data	Fully qualified path name to a JSON key-value pair.
Stream	A stream of data messages from an external data source, parsed to a format usable by the EDC services. Encapsulates a connection to an external data source	JSON messages received by periodically querying an HTTP server with GET requests
Action	An operation on stream data that interacts with the wider Ubisense platform, often setting a UDM value or a location	A property action that takes a JSON value and sets a UDM property to that value

Configuration workflow

Configuration is performed in SmartSpace Config and Service Manager. The configuration process involves creating objects for stream and action types and setting service parameters for them as follows:

1. Create stream object(s).
2. Create action object(s).
3. Configure stream object parameters.
4. Configure action parameters.
5. Start stream services.

These steps are described in the sections that follow, using an example that involves setting up a service to retrieve locations from an HTTP URL.

Requirements

SmartSpace

The External data connector requires a license for RTLS integration version 3.5 or higher.

Microsoft .NET Core

On Windows, the External data connector requires Microsoft windows-core-req. For Linux servers, you may need to install .NET Core: follow the instructions for Linux at *https://learn.microsoft.com/en-gb/dotnet/core/install/linux*.

The External data connector requires Microsoft .NET Core 3.1.x. For Linux servers, you may need to install .NET Core: follow the instructions for Linux at https://dotnet.microsoft.com/.

External Systems

External systems must use a supported format and protocol. Format specific requirements are outlined below.

General format:

- Timestamps can be in an ISO 8601-compliant format or a UNIX epoch time
- Where the format supports it, values of null will be treated as the stream "null". They will otherwise be ignored.

JSON

All relevant data fields must be key-value pairs or values within an array at a fixed index. If the data is nested, the same must be true for all parent objects of the data fields.

XML

Valid XML where data fields can be either attributes or elements.

CSV

CSV data should consist of one or more rows separated by line breaks, optionally starting with a line/row of column headings. Columns are separated by commas by default but other character (s) can be configured.

CSV data should consist of two or more rows separated by line breaks. The first row must start with a line/row of column headings. Columns are separated by commas by default but other character(s) can be configured.

Single-field Data

For unformatted/unlabeled source data. Each message should be a single string line in most cases. Each line will be treated as a data field with the identity “field” (all lines will have the same identity).

Installing the External data connector

To install the External data connector feature:

- Make sure that the SmartSpace platform includes a license for RTLS integration version 3.5 or higher.
- Install the External data connector feature using Service Manager.

For further information on installing SmartSpace features see SmartSpace Installation on the Ubisense Documentation Portal.

Configuring the External data connector

The following sections take you through setting up a connection to an existing HTTP server, serving location data in JSON format, and using the EDC to set locations based on this data. The process will involve creating a stream object *ExampleHttpRequester stream*, and the single action associated with it. You will then see how to define the parameters for these types and how to deploy the service.

For more details on the other stream and object options available to you, see the *Types and parameters*.

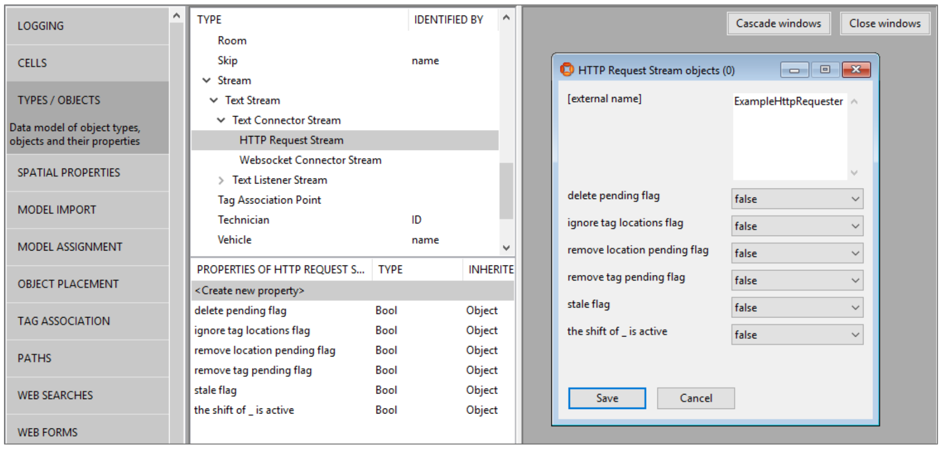
Creating the stream object

You create stream objects using the Ubisense-supplied types in the TYPES / OBJECTS tab of SmartSpace Config. Each external system connection requires its own stream object.

In this example we will only need one stream object, for our one HTTP server stream.

To create a stream object:

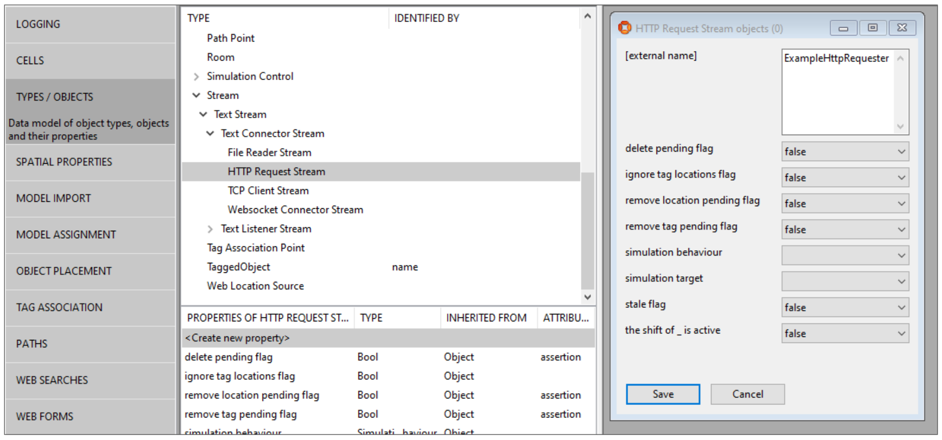
1. In SmartSpace Config, choose the TYPES / OBJECTS task.
2. Drag the required Stream type, **HTTP Request Stream** in this example, into the object browser and double-click **<Create new object>**.



TYPE	IDENTIFIED BY
Room	
Skip	name
Stream	
Text Stream	
Text Connector Stream	
HTTP Request Stream	
Websocket Connector Stream	
Text Listener Stream	
Tag Association Point	
Technician	ID
Vehicle	name

PROPERTIES OF HTTP REQUEST S...	TYPE	INHERITE
<Create new property>		
delete pending flag	Bool	Object
ignore tag locations flag	Bool	Object
remove location pending flag	Bool	Object
remove tag pending flag	Bool	Object
stale flag	Bool	Object
the shift of _ is active	Bool	Object

[external name]	ExampleHttpRequester
delete pending flag	false
ignore tag locations flag	false
remove location pending flag	false
remove tag pending flag	false
stale flag	false
the shift of _ is active	false



TYPE	IDENTIFIED BY
Path Point	
Room	
Simulation Control	
Stream	
Text Stream	
Text Connector Stream	
File Reader Stream	
HTTP Request Stream	
TCP Client Stream	
Websocket Connector Stream	
Text Listener Stream	
Tag Association Point	
TaggedObject	name
Web Location Source	

PROPERTIES OF HTTP REQUEST ST...	TYPE	INHERITED FROM	ATTRIBU...
<Create new property>			
delete pending flag	Bool	Object	assertion
ignore tag locations flag	Bool	Object	
remove location pending flag	Bool	Object	assertion
remove tag pending flag	Bool	Object	assertion

[external name]	ExampleHttpRequester
delete pending flag	false
ignore tag locations flag	false
remove location pending flag	false
remove tag pending flag	false
simulation behaviour	
simulation target	
stale flag	false
the shift of _ is active	false

3. In the dialog, enter the object's name, here *ExampleHttpRequester*, and click **Save**.

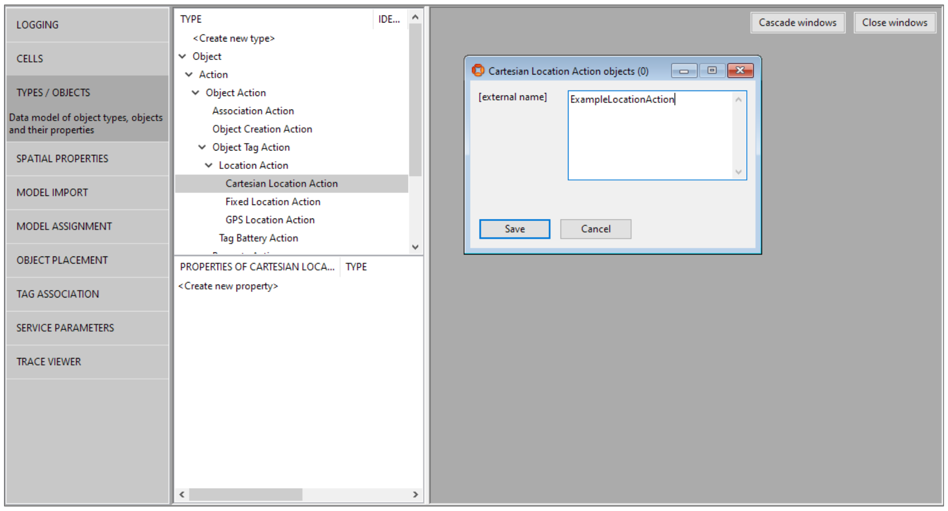
Creating Action Objects

You create actions in the same way you create stream objects.

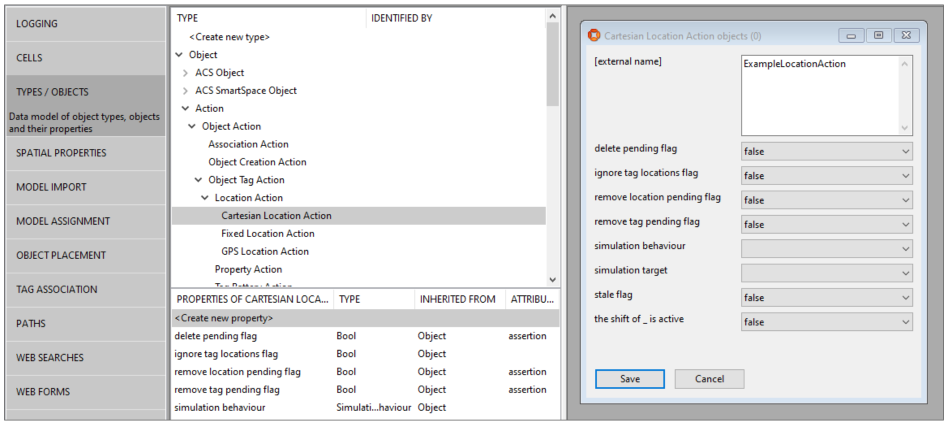
We will need only one action object in this example, a Cartesian Location Action, which will set object locations for us.

To create an action object:

1. In SmartSpace Config, choose the **TYPES / OBJECTS** task.
2. Drag an Action type, **Cartesian Location Action** in this example, into the object browser and double-click **<Create new object>**.



[external name] ExampleLocationAction



PROPERTIES OF CARTESIAN LOCA...	TYPE	INHERITED FROM	ATTRIBU...
<Create new property>			
delete pending flag	Bool	Object	assertion
ignore tag locations flag	Bool	Object	
remove location pending flag	Bool	Object	assertion
remove tag pending flag	Bool	Object	assertion
simulation behaviour	Simulati...haviour	Object	

[external name]	ExampleLocationAction
delete pending flag	false
ignore tag locations flag	false
remove location pending flag	false
remove tag pending flag	false
simulation behaviour	
simulation target	
stale flag	false
the shift of _ is active	false

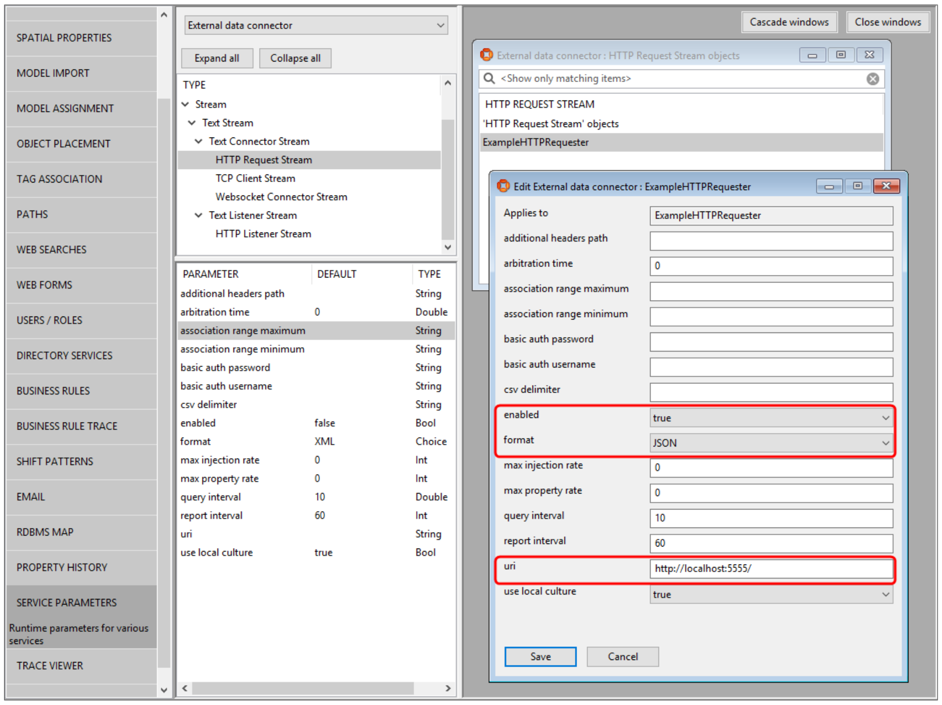
3. In the dialog enter the object's name, here *ExampleLocationAction*, and click **Save**.

Parameters for Stream Objects

Configuration of streams is performed in the SERVICE PARAMETERS task. Configuration involves the following steps:

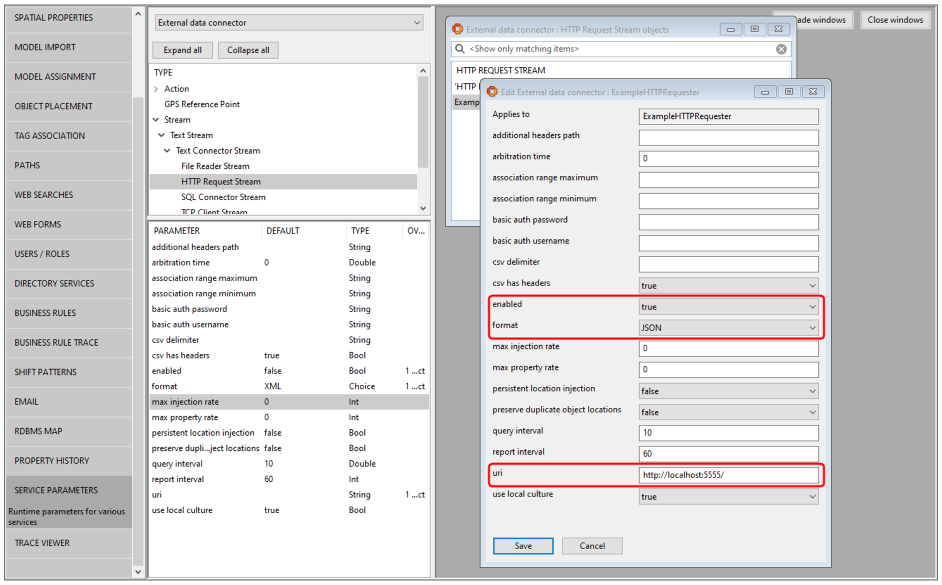
1. In SmartSpace Config, choose the **SERVICE PARAMETERS** task.
2. Choose the **External data connector** configuration, and then in the expandable type list find the type of the stream object you just created, **Http Request Stream** in our case. (Use the **Expand All** button to display the object hierarchy, if necessary.)

3. Drag this type into the object browser to display a window with all available objects of this type. Double-click the stream object created above, **ExampleHttpRequester**, and click **Edit** to edit its parameters.



PARAMETER	DEFAULT	TYPE
additional headers path		String
arbitration time	0	Double
association range maximum		String
association range minimum		String
basic auth password		String
basic auth username		String
csv delimiter		String
enabled	false	Bool
format	XML	Choice
max injection rate	0	Int
max property rate	0	Int
query interval	10	Double
report interval	60	Int
uri		String
use local culture	true	Bool

Applies to	ExampleHTTPRequester
additional headers path	
arbitration time	0
association range maximum	
association range minimum	
basic auth password	
basic auth username	
csv delimiter	
enabled	true
format	JSON
max injection rate	0
max property rate	0
query interval	10
report interval	60
uri	http://localhost:5555/
use local culture	true



PARAMETER	DEFAULT	TYPE	OV...
additional headers path		String	
arbitration time	0	Double	
association range maximum		String	
association range minimum		String	
basic auth password		String	
basic auth username		String	
csv delimiter		String	
csv has headers	true	Bool	
enabled	false	Bool	1 ...ct
format	XML	Choice	1 ...ct
max injection rate	0	Int	
max property rate	0	Int	
persistent location injection	false	Bool	
preserve dupli...ject locations	false	Bool	
query interval	10	Double	
report interval	60	Int	
uri		String	1 ...ct
use local culture	true	Bool	

Applies to	ExampleHTTPRequester
additional headers path	
arbitration time	0
association range maximum	
association range minimum	
basic auth password	
basic auth username	
csv delimiter	
csv has headers	true
enabled	true
format	JSON
max injection rate	0
max property rate	0
persistent location injection	false
preserve duplicate object locations	false
query interval	10
report interval	60
uri	http://localhost:5555/
use local culture	true

4. Edit the parameters needed for this stream, setting the enabled flag to true when done.

The parameters offered depend on the kind of stream you have created. The complete list of stream parameters for all stream types is given in *Stream Parameters*.

In our example, we have set the URI, format and enabled parameters, leaving the other parameters with their default values.

5. Click **Save**.

Note: Stream services do not react to changes to stateful configuration parameters, for example changes of address for TCP streams. We recommend that you always restart a stream service in Service Manager after changing its parameters. See *Starting Stream Services*.

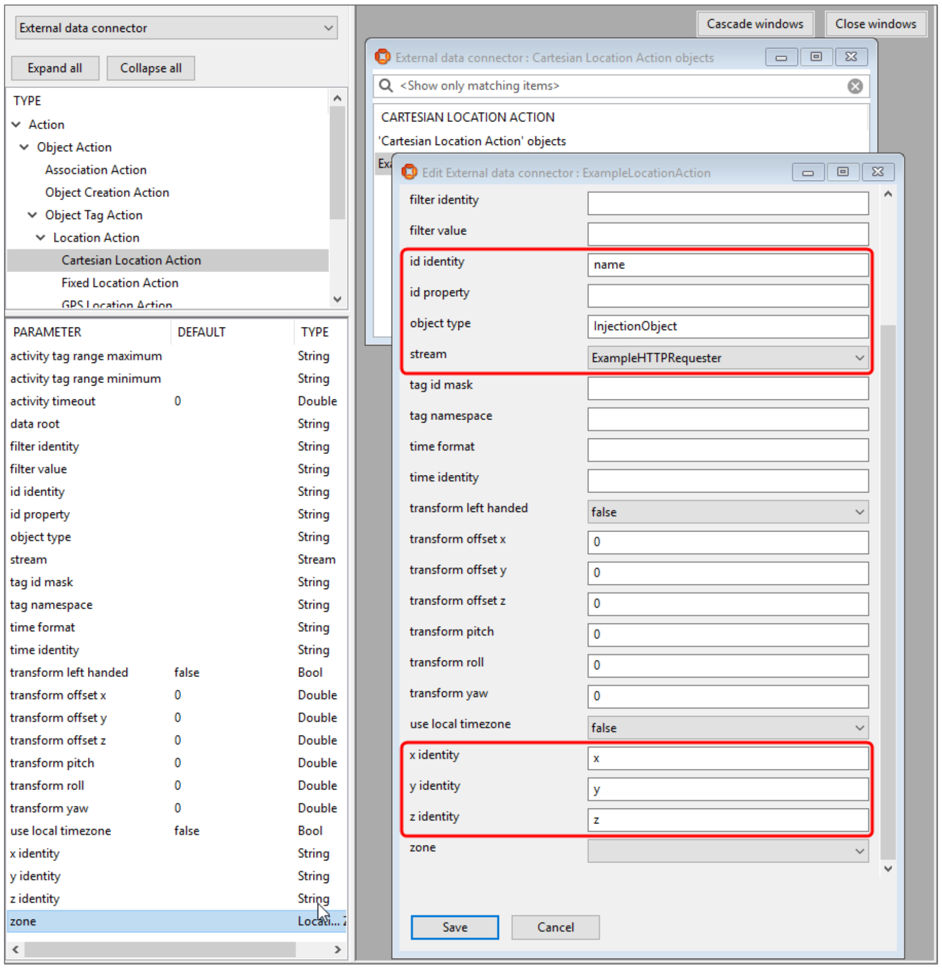
Parameters for Actions

Configuration of actions is performed in the SERVICE PARAMETERS task, similar to stream configuration above.

To configure service parameters for an action:

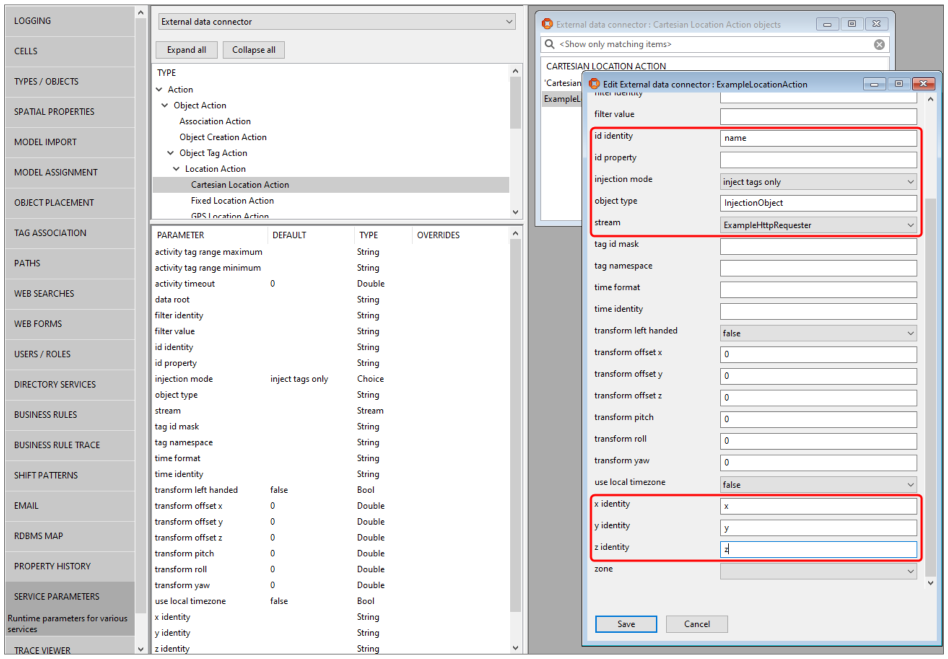
1. In SmartSpace Config, choose the **SERVICE PARAMETERS** task.
2. Choose the **External data connector** configuration, and then in the expandable type list find the type of the action object you just created, **LocationAction** in this example. (Use the **Expand All** button to display the object hierarchy, if necessary.)

3. Drag this type into the object browser to display a window with all available objects of this type. Double-click the action object created above, **ExampleLocationAction** in our case, to display the available objects, and click **Edit** in the newly-opened window to edit its parameters.


External data connector
Expand all
Collapse all
TYPE
Action
Object Action
Association Action
Object Creation Action
Object Tag Action
Location Action
Cartesian Location Action
Fixed Location Action
GPS Location Action
Cascade windows
Close windows
External data connector : Cartesian Location Action objects
<Show only matching items>
CARTESIAN LOCATION ACTION
'Cartesian Location Action' objects
Edit External data connector : ExampleLocationAction
Save
Cancel

PARAMETER	DEFAULT	TYPE
activity tag range maximum		String
activity tag range minimum		String
activity timeout	0	Double
data root		String
filter identity		String
filter value		String
id identity		String
id property		String
object type		String
stream		Stream
tag id mask		String
tag namespace		String
time format		String
time identity		String
transform left handed	false	Bool
transform offset x	0	Double
transform offset y	0	Double
transform offset z	0	Double
transform pitch	0	Double
transform roll	0	Double
transform yaw	0	Double
use local timezone	false	Bool
x identity		String
y identity		String
z identity		String
zone		Locati...

filter identity	
filter value	
id identity	name
id property	
object type	InjectionObject
stream	ExampleHTTPRequester
tag id mask	
tag namespace	
time format	
time identity	
transform left handed	false
transform offset x	0
transform offset y	0
transform offset z	0
transform pitch	0
transform roll	0
transform yaw	0
use local timezone	false
x identity	x
y identity	y
z identity	z
zone	



PARAMETER	DEFAULT	TYPE	OVERRIDES
activity tag range maximum		String	
activity tag range minimum		String	
activity timeout	0	Double	
data root		String	
filter identity		String	
filter value		String	
id identity		String	
id property		String	
injection mode	inject tags only	Choice	
object type		String	
stream		Stream	
tag id mask		String	
tag namespace		String	
time format		String	
time identity		String	
transform left handed	false	Bool	
transform offset x	0	Double	
transform offset y	0	Double	
transform offset z	0	Double	
transform pitch	0	Double	
transform roll	0	Double	
transform yaw	0	Double	
use local timezone	false	Bool	
x identity		String	
y identity		String	
z identity		String	

filter value	
id identity	name
id property	
injection mode	inject tags only
object type	InjectionObject
stream	ExampleHttpRequester
tag id mask	
tag namespace	
time format	
time identity	
transform left handed	false
transform offset x	0
transform offset y	0
transform offset z	0
transform pitch	0
transform roll	0
transform yaw	0
use local timezone	false
x identity	x
y identity	y
z identity	z
zone	

4. Edit the relevant parameters in this action. The parameters offered depend on the kind of action you have created. The complete list of action parameters for all types of action is given in *Action Parameters*.

   Typically you will need to set several identity parameters with the names of data fields in your source data so the action knows what data fields are relevant and what their value represents.

   In our example, we have configured the x, y, z identity parameters, telling our action that the x Cartesian coordinate will be in a data field with name/identity of “x”. Because this data is object data (not tag data), we have also asserted it is for the object type InjectionObject with the object name in the data field named “name”. Lastly, we set the stream this action should operate on, the ExampleHttpRequester stream we created earlier.

   Note: by default for object data, the objects will need to have tags associated as the service will only set the location of tags. If you wish to set object locations directly you will need to change the *injection mode* parameter

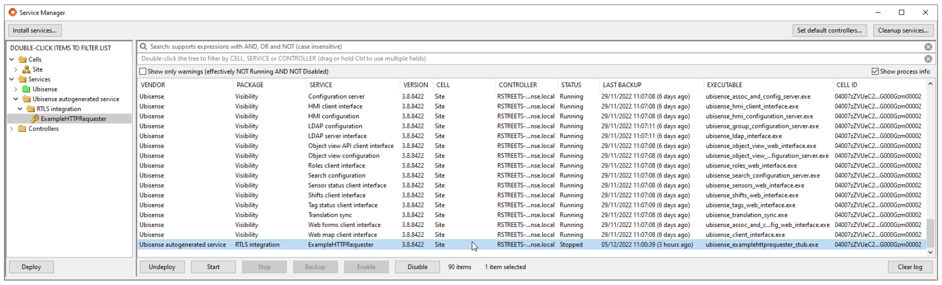
5. Click **Save**.

Starting Stream Services

When a stream is enabled (by setting the **enabled** parameter to **true**, described above), a service named after the stream object is created to manage that stream and its actions. This service is deployed but not started: you must start it manually after configuration is complete.

To start the service for a stream:

1. In Service Manager, open the MANAGE SERVICES tab.
2. Run the Service Manager application.
3. Navigate to the service by opening Services > Ubisense autogenerated service > RTLS integration and any enabled services are listed, identified by the name given to the stream objects.
4. Navigate to the service by opening **Services > Ubisense autogenerated service > RTLS integration** and any enabled services are listed, identified by the name given to the stream objects. You can also type all or part of a steam name into the filter to navigate to it directly.


Service Manager
Install services...
Set default controllers...
Cleanup services...
DOUBLE-CLICK ITEMS TO FILTER LIST
Cells
Site
Services
Ubisense
Ubisense autogenerated service
RTLS integration
ExampleHTTPRequester
Controllers
Search: supports expressions with AND, OR and NOT (case insensitive)
Double-click the tree to filter by CELL, SERVICE or CONTROLLER (drag or hold Ctrl to use multiple fields)
Show only warnings (effectively NOT Running AND NOT Disabled)
Show process info
Deploy
Undeploy
Start
Stop
Backup
Enable
Disable
90 items
1 item selected
Clear log

VENDOR	PACKAGE	SERVICE	VERSION	CELL	CONTROLLER	STATUS	LAST BACKUP	EXECUTABLE	CELL ID
Ubisense	Visibility	Configuration server	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:08 (6 days ago)	ubisense_assoc_and_config_server.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	HMI client interface	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:08 (6 days ago)	ubisense_hmi_client_interface.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	HMI configuration	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:08 (6 days ago)	ubisense_hmi_configuration_server.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	LDAP configuration	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:11 (6 days ago)	ubisense_group_configuration_server.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	LDAP server interface	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:11 (6 days ago)	ubisense_ldap_interface.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	Object view API client interface	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:08 (6 days ago)	ubisense_object_view_web_interface.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	Object view configuration	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:08 (6 days ago)	ubisense_object_view_...figuration_server.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	Roles client interface	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:08 (6 days ago)	ubisense_roles_web_interface.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	Search configuration	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:08 (6 days ago)	ubisense_search_configuration_server.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	Sensor status client interface	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:08 (6 days ago)	ubisense_sensors_web_interface.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	Shifts client interface	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:08 (6 days ago)	ubisense_shifts_web_interface.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	Tag status client interface	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:09 (6 days ago)	ubisense_tags_web_interface.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	Translation sync	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:08 (6 days ago)	ubisense_translation_sync.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	Web forms client interface	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:08 (6 days ago)	ubisense_assoc_and_c...fig_web_interface.exe	04007zZVUeC2...G000Gzm00002
Ubisense	Visibility	Web map client interface	3.8.8422	Site	RSTREETS-...nse.local	Running	29/11/2022 11:07:08 (6 days ago)	ubisense_client_interface.exe	04007zZVUeC2...G000Gzm00002
Ubisense autogenerated service	RTLS integration	ExampleHTTPRequester	3.8.8422	Site	RSTREETS-...nse.local	Stopped	05/12/2022 11:00:39 (3 hours ago)	ubisense_examplehttprequester_stub.exe	04007zZVUeC2...G000Gzm00002

5. Select the service and click **Start**.

   In our example, we start the **Ubisense autogenerated service::RTLS integration::ExampleHttpRequester** service.

   If the service does not work, you can use the messages generated by the **data_connector** and **data_connector_debug** trace streams to identify problems with the stream configuration. See *Trace Messages*.

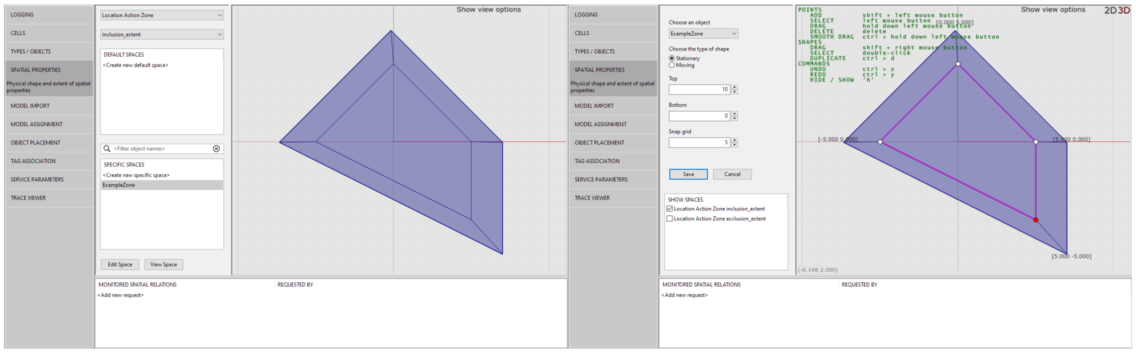
Updating the configuration

Stream services do not react to changes to stateful configuration parameters, for example changes of address for TCP streams. We recommend that you always restart a stream service after changing its parameters (by locating it, as described above, and clicking **Restart**).

Changes to actions should not require a service restart.

Configuring Location Action Zones

Inclusion/exclusion zones can be used with location actions to control what locations are injected. After creating a Location Action Zone object in the Types and objects workspace, the inclusion/exclusion shapes can be configured in the Spatial properties workspace. Locations inside an exclusion shape will be ignored. When an inclusion shape is defined, locations outside the shape will be ignored. A zone can have both an exclusion shape and inclusion shape with exclusion shapes superseding inclusion shapes.



```
POINTS
   ADD          shift + left mouse button
   SELECT       left mouse button
   DRAG         hold down left mouse button
   DELETE       delete
   SMOOTH DRAG  ctrl + hold down left mouse button
SHAPES
   DRAG         shift + right mouse button
   SELECT       double-click
   DUPLICATE    ctrl + d
COMMANDS
   UNDO         ctrl + z
   REDO         ctrl + y
   HIDE / SHOW  'h'
```

Zones should be stationary shapes. A stream can use one zone at a time.

Types and parameters

This section lists the types available for use with the External data connector and the parameters to configure them. The lists include the parent types on which the Ubisense types for the different connection types are based. These are shown for information only: you should base your streams and actions on the Ubisense types derived from them.

Types

Stream Types

Type	Purpose
Stream	Abstract base type for streams
Text Stream	Abstract base class for text based streams
Text Listener Stream	Abstract base class for text listener streams
HTTP Listener Stream	Listens for POST/PUT HTTP(S) requests See also *Ensuring EDC HTTP Listener Streams are Implemented Securely* for a discussion of securing HTTP listener streams.
Text Connector Stream	Abstract base class for text connector stream
HTTP Request Stream	Retrieves data via periodic HTTP(S) GET requests
SQL Connector Stream	Connects to a SQL database and queries for data
TCP Client Stream	Connects to a TCP server socket and listens for data
Websocket Connector Stream	Connects to a server via websockets and listens for data
File Reader Stream	Reads data from a text file

Action Types

Type	Purpose
Action	Abstract base type for all actions
Object Action	Abstract base class for actions on object data
Object Tag Action	Abstract base class for actions on object/tag data
Location Action	Abstract base class for actions on object/tag location data
Cartesian Location Action	Injects object/tag locations from Cartesian (x/y/z) data
GPS Location Action	Injects object/tag locations from GPS data
Fixed Location Action	Injects object/tag locations at a fixed position
Tag Battery Action	Asserts tag battery status from parsed data
Property Action	Sets UDM property values for objects based on parsed data
Association Action	Associates unassociated parsed objects with free tags from a given range
Object Creation Action	Creates missing SmartSpace objects to match objects from parsed data. It is recommended that you only have one Object Creation Action per External system

Zones

Type	Purpose
Location Action Zone	Used to define inclusion/exclusion extents for location actions
GPS Reference Point	Used to configure GPS coordinate conversion

GPS Reference Points

At least two GPS reference points are needed to convert GPS locations to the Cartesian coordinates used by the platform. These coordinates are defined by creating GPS Reference Point objects in the Types and objects workspace and then setting their x, y, latitude and longitude values in SERVICE PARAMETERS.

When more than two GPS reference points are defined, only the two points closest to a parsed location are used in the conversion. This should allow the use of GPS reference points when multiple areas that are not geographically adjacent are placed adjacent on the map, so long as there are at least two reference points for each area.

Parameters

Stream Parameters

Parameter	Type	Purpose
arbitration time	Stream	Arbitration time in seconds. When set to a positive value, tag/object locations must be newer by this amount than the most recent location seen by the platform for that tag/object, else they will be ignored. **How arbitration time is used:** In the External data connector *Arbitration time* is used as follows. Where: *A* is arbitration time *X* is the time of a new location seen by the service for a tag *T* *Y* is the last time the platform saw the tag *T* The location will only be injected if X >= Y + A Significant values for arbitration time are: • **0** (or any negative number) = arbitration disabled • **0.1** (or any small, positive number) = avoid injecting repeated locations • **10** (some larger number) = give priority to another system, i.e. if you are using both Ubisense tags and an external GPS system, give Ubisense tags priority
association range mimimum	Stream	For use with the association action. Mimimum tag id of the tag pool available to the association action
association range maximum	Stream	For use with the association action. Maximum tag id of the tag pool available to the association action
enabled	Stream	Enable/disable the stream service
max injection rate	Stream	Maximum rate (Hz) at which locations will be sent to Ubisense cells for the service

Parameter	Type	Purpose
max property rate	Stream	Maximum rate (Hz) at which properties will be set
persistent location injection	Stream	Whether to ensure injected location events are persistent. Non-persistent injection does not wait for confirmation from the location cell but events may be silently dropped with large batches of locations. The max injection rate parameter can help reduce the number of dropped locations.
preserve duplicate object locations	Stream	Enable when data contains duplicate objects/tags in a single message and all historical locations must be preserved/injected
report interval	Stream	Interval between monitor of the service in seconds
use local culture	Stream	If **true** allows for parsing of data using local conventions; false uses EN-gb conventions
csv delimiter	Text Stream	Delimiter used to separate values in CSV data. The service will use the default for the current culture when unset
csv has headers	Text Stream	Set to false when the csv has no header row. [x] notation should be used to specify column numbers for identities in this case.
format	Text Stream	Format of received data
query interval	Text Connector Stream	Interval between HTTP queries in seconds
additional headers path	HTTP Request Stream	Path to a file containing one or more HTTP headers to include in requests
basic auth password	HTTP Request Stream	Basic authentication password to use with HTTPS
basic auth username	HTTP Request Stream	Basic authentication username to use with HTTPS

Parameter	Type	Purpose
uri	HTTP Request Stream	URI to query
sql connection string	SQL Connector Stream	Connection string for the SQL server/database
sql query string	SQL Connector Stream	The SQL query string for retrieving data
endpoint address	TCP Client Stream	Address of the TCP server socket to connect to
endpoint port	TCP Client Stream	Port of the TCP server socket to connect to
message end regex	TCP Client Stream	Regex used to identify end of a complete message from the server. New line is used by default
initialisation message path	Websocket Connector Stream	Stream Path to text file containing message(s) to send to the server when the service connects. For use when the external system requires an initialisation/subscription message before receiving data
password	Websocket Connector Stream	Basic authentication password to use with the websocket
server url	Websocket Connector Stream	URL to connect to
username	Websocket Connector Stream	Basic authentication username to use with the websocket

Parameter	Type	Purpose
listening url	HTTP Listener Stream	One or more URLs to listen for clients on listening url can be one URL, or a space-separated list of URLs. For example the listener could be configured to accept requests sent to localhost:4444 and proxy.external:8080 by specifying: `http://localhost:4444/ http://proxy.external:8080/` A trailing / is automatically added for each listening URL if it has been omitted. The listener will only accept a request if it is sent to an address matching one of the URLs specified. For example "localhost" is not the same as "127.0.0.1", and "hostname.domain" is not the same as "hostname". The hostname can use "*" or "*.mydomain.com", but this is less secure than specifying an exact hostname in the URL. A reverse proxy or firewall could be supported by specifying the hostname of the proxy/firewall. So for the proxy.external example above, a reverse proxy on server proxy.external might pass the request on its port 8080 to hostname.internal:8080. On Linux, to support an https primary end point set the listening url something like "http://*:8008/endpoint", configure an Apache or NGINX reverse proxy to forward the https request to "http://127.0.0.1:8008/", and then use the firewall on Linux to block anything sent to port 8008 from any other host. Requests to "https://proxyhostname.domain/endpoint" will then be handled by the External data connector listening at "http://127.0.0.1:8008/endpoint".
basic authentication username	HTTP Listener Stream	Username to use for basic authentication. Only requests using basic authentication and matching this username will be accepted
basic authentication password	HTTP Listener Stream	Password to use for basic authentication. Only requests using basic authentication and matching this password will be accepted

Parameter	Type	Purpose
file path	File Reader Stream	File path to read from, for example "C:\Ubisense\file_reader_ source.txt".

Show table sorted by parameter

Parameter	Type	Purpose
additional headers path	HTTP Request Stream	Path to a file containing one or more HTTP headers to include in requests
arbitration time	Stream	Arbitration time in seconds. When set to a positive value, tag/object locations must be newer by this amount than the most recent location seen by the platform for that tag/object, else they will be ignored. **How arbitration time is used:** In the External data connector *Arbitration time* is used as follows. Where: *A* is arbitration time *X* is the time of a new location seen by the service for a tag *T* *Y* is the last time the platform saw the tag *T* The location will only be injected if X >= Y + A Significant values for arbitration time are: • **0** (or any negative number) = arbitration disabled • **0.1** (or any small, positive number) = avoid injecting repeated locations • **10** (some larger number) = give priority to another system, i.e. if you are using both Ubisense tags and an external GPS system, give Ubisense tags priority
association range maximum	Stream	For use with the association action. Maximum tag id of the tag pool available to the association action
association range mimimum	Stream	For use with the association action. Mimimum tag id of the tag pool available to the association action
basic auth password	HTTP Request Stream	Basic authentication password to use with HTTPS

Parameter	Type	Purpose
basic auth username	HTTP Request Stream	Basic authentication username to use with HTTPS
basic authentication username	HTTP Listener Stream	Username to use for basic authentication. Only requests using basic authentication and matching this username will be accepted
csv delimiter	Text Stream	Delimiter used to separate values in CSV data. The service will use the default for the current culture when unset
csv has headers	Text Stream	Set to false when the csv has no header row. [x] notation should be used to to specify column numbers for identities in this case.
enabled	Stream	Enable/disable the stream service
endpoint address	TCP Client Stream	Address of the TCP server socket to connect to
endpoint port	TCP Client Stream	Port of the TCP server socket to connect to
format	Text Stream	Format of received data
initialisation message path	Websocket Connector Stream	Stream Path to text file containing message(s) to send to the server when the service connects. For use when the external system requires an initialisation/subscription message before receiving data

Parameter	Type	Purpose
listening url	HTTP Listener Stream	One or more URLs to listen for clients on listening url can be one URL, or a space-separated list of URLs. For example the listener could be configured to accept requests sent to localhost:4444 and proxy.external:8080 by specifying: `http://localhost:4444/ http://proxy.external:8080/` A trailing / is automatically added for each listening URL if it has been omitted. The listener will only accept a request if it is sent to an address matching one of the URLs specified. For example "localhost" is not the same as "127.0.0.1", and "hostname.domain" is not the same as "hostname". The hostname can use "*" or "*.mydomain.com", but this is less secure than specifying an exact hostname in the URL. A reverse proxy or firewall could be supported by specifying the hostname of the proxy/firewall. So for the proxy.external example above, a reverse proxy on server proxy.external might pass the request on its port 8080 to hostname.internal:8080.
max injection rate	Stream	Maximum rate (Hz) at which locations will be sent to Ubisense cells for the service
max property rate	Stream	Maximum rate (Hz) at which properties will be set
message end regex	TCP Client Stream	Regex used to identify end of a complete message from the server. New line is used by default
password	Websocket Connector Stream	Basic authentication password to use with the websocket
persistent location injection	Stream	Whether to ensure injected location events are persistent. Non-persistent injection does not wait for confirmation from the location cell but events may be silently dropped with large batches of locations. The max injection rate parameter can help reduce the number of dropped locations.

Parameter	Type	Purpose
preserve duplicate object locations	Stream	Enable when data contains duplicate objects/tags in a single message and all historical locations must be preserved/injected
query interval	Text Connector Stream	Interval between HTTP queries in seconds
report interval	Stream	Interval between monitor of the service in seconds
server url	Websocket Connector Stream	URL to connect to
sql connection string	SQL Connector Stream	Connection string for the SQL server/database
sql query string	SQL Connector Stream	The SQL query string for retrieving data
uri	HTTP Request Stream	URI to query
use local culture	Stream	If **true** allows for parsing of data using local conventions; false uses EN-gb conventions
username	Websocket Connector Stream	Basic authentication username to use with the websocket

Action Parameters

Parameter	Type	Purpose
stream	Action	The stream to act on
data root	Object Action	Identity of the element containing the relevant data in source data. For nested data, e.g. complex JSON/XML
filter identity	Object Action	Identity of the filter field in source data. When set, data for objects not matching the filter value will be ignored
filter value	Object Action	Accepted value for the filter
id identity	Object Action	Identity of the id field (the tag id or object name) in source data
id property	Object Action	Name of the SmartSpace property to match the ID identity value against in object lookup. For when the id identity matches a property of the object other than the unique name. The property must be unique.
object type	Object Action	When set, the ID identity is assumed to be an object name, otherwise ID identity is assumed to be a tag. The type to use, together with the ID identity, to determine which object from source data
tag id mask	Object Tag Action	Bitmask to apply (as a bitwise OR) to parsed tag IDs. Cannot be used with tag id namespaces.
tag namespace	Object Tag Action	The namespace to use for non-Ubisense tag IDs. See *Tag namespaces* for details
activity tag range maximum	Location Action	Maximum tag boundary of the monitored tag range. Tags in this range will have their activity set to inactive when unseen by the action for activity timeout seconds
activity tag range minimum	Location Action	Minimum tag boundary of the monitored tag range. Tags in this range will have their activity set to inactive when unseen by the action for activity timeout seconds

Parameter	Type	Purpose
activity timeout	Location Action	Interval, in seconds, without the action seeing an object/tag before its activity is set to inactive. The tag must be in the monitored range. When this parameter is not manually set (or is set to a value <= 0 with 0 being the default), the location action will not set the tag's activity to active.
injection mode	Location Action	For object/non-tag source data. Determines how an action will set object locations: • inject tags only: the action will only inject tag locations. Objects referred to by the data must have a tag associated. This is the default setting. • inject objects and tags: the action can inject object locations directly when the object does not have a tags associated. Injection of tag locations is still the preference and will be used instead when an associated tag is available.
time format	Location Action	Custom format specifier for non-ISO 8601, non-unix timestamp date/times. Details of suitable values can be found here *https://docs.microsoft.com/en-us/dotnet/standard/base-types/custom-date-and-time-format-strings?view=netframework-4.8*
time identity	Location Action	Identity of time values in source data. Current time is assumed when this is not set time identity is the name of the field containing the time at which you wish to inject the tag or object location (which depends on the *injection mode*). Tags cannot meaningfully be injected more than 60 seconds in the past (strictly they can be, but any associated object is not updated). From version 3.7 SP1, retrospective injection is supported on *objects*.
transform left handed	Location Action	Set to true when the source system uses a left handed coordinate system, i.e. the y coordinate needs to be negated to match the Ubisense coordinate system
transform offset x	Location Action	Offset to add to the x coordinate of parsed locations (after applying transform rotation)

Parameter	Type	Purpose
transform offset y	Location Action	Offset to add to the y coordinate of parsed locations
transform offset z	Location Action	Offset to add to the z coordinate of parsed locations
transform pitch	Location Action	Pitch rotation in degrees to apply to parsed locations
transform roll	Location Action	Roll rotation in degrees to apply to parsed locations
transform yaw	Location Action	Yaw rotation in degrees to apply to parsed locations
use local timezone	Location Action	Whether parsed times are assumed to be local or UTC times
zone	Location Action	Inclusion/exclusion zones to use
x identity	Cartesian Location Action	Identity of the x coordinate field in source data
y identity	Cartesian Location Action	Identity of the y coordinate field in source data
z identity	Cartesian Location Action	Identity of the z coordinate field in source data
fixed x	Fixed Location Action	Fixed x co-ordinate to use for injected locations

Parameter	Type	Purpose
fixed y	Fixed Location Action	Fixed y co-ordinate to use for injected locations
fixed z	Fixed Location Action	Fixed z co-ordinate to use for injected locations
latitude identity	GPS Location Action	Identity of the latitude field in source data
longitude identity	GPS Location Action	Identity of the longitude field in source data
SmartSpace property	Property Action	Name of the SmartSpace property to set
property identity	Property Action	Identity of the field containing the property value
battery identity	Tag Battery Action	Identity of the battery status field in source data
failing values	Tag Battery Action	Comma-separated values to parse as failing
ok values	Tag Battery Action	Comma-separated values to parse as ok
unknown values	Tag Battery Action	Comma-separated values to parse as unknown
warning values	Tag Battery Action	Comma-separated values to parse as warning

Show table sorted by parameter

Parameter	Type	Purpose
activity tag range maximum	Location Action	Maximum tag boundary of the monitored tag range. Tags in this range will have their activity set to inactive when unseen by the action for activity timeout seconds
activity tag range minimum	Location Action	Minimum tag boundary of the monitored tag range. Tags in this range will have their activity set to inactive when unseen by the action for activity timeout seconds
activity timeout	Location Action	Interval, in seconds, without the action seeing an object/tag before its activity is set to inactive. The tag must be in the monitored range. When this parameter is not manually set (or is set to a value <= 0 with 0 being the default), the location action will not set the tag's activity to active.
battery identity	Tag Battery Action	Identity of the battery status field in source data
data root	Object Action	Identity of the element containing the relevant data in source data. For nested data, e.g. complex JSON/XML
failing values	Tag Battery Action	Comma-separated values to parse as failing
filter identity	Object Action	Identity of the filter field in source data. When set, data for objects not matching the filter value will be ignored
filter value	Object Action	Accepted value for the filter
fixed x	Fixed Location Action	Fixed x co-ordinate to use for injected locations
fixed y	Fixed Location Action	Fixed y co-ordinate to use for injected locations
fixed z	Fixed Location Action	Fixed z co-ordinate to use for injected locations

Parameter	Type	Purpose
id identity	Object Action	Identity of the id field (the tag id or object name) in source data
id property	Object Action	Name of the SmartSpace property to match the ID identity value against in object lookup. For when the id identity matches a property of the object other than the unique name. The property must be unique.
injection mode	Location Action	For object/non-tag source data. Determines how an action will set object locations: • inject tags only: the action will only inject tag locations. Objects referred to by the data must have a tag associated. This is the default setting. • inject objects and tags: the action can inject object locations directly when the object does not have a tag associated. Injection of tag locations is still the preference and will be used instead when an associated tag is available.
latitude identity	GPS Location Action	Identity of the latitude field in source data
longitude identity	GPS Location Action	Identity of the longitude field in source data
object type	Object Action	When set, the ID identity is assumed to be an object name, otherwise id identity is assumed to be a tag. The type to use, together with the ID identity, to determine which object from source data
ok values	Tag Battery Action	Comma-separated values to parse as ok
property identity	Property Action	Identity of the field containing the property value
SmartSpace property	Property Action	Name of the SmartSpace property to set
stream	Action	The stream to act on

Parameter	Type	Purpose
tag id mask	Object Tag Action	Bitmask to apply (as a bitwise OR) to parsed tag IDs. Cannot be used with tag id namespaces.
tag namespace	Object Tag Action	The namespace to use for non-Ubisense tag IDs. See *Tag namespaces* for details
time format	Location Action	Custom format specifier for non-ISO 8601, non-unix timestamp date/times. Details of suitable values can be found here *https://docs.microsoft.com/en-us/dotnet/standard/base-types/custom-date-and-time-format-strings?view=netframework-4.8*
time identity	Location Action	Identity of time values in source data. Current time is assumed when this is not set time identity is the name of the field containing the time at which you wish to inject the tag or object location (which depends on the *injection mode*). Tags cannot meaningfully be injected more than 60 seconds in the past (strictly they can be, but any associated object is not updated). From version 3.7 SP1, retrospective injection is supported on *objects*.
transform left handed	Location Action	Set to true when the source system uses a left handed coordinate system, i.e. the y coordinate needs to be negated to match the Ubisense coordinate system
transform offset x	Location Action	Offset to add to the x coordinate of parsed locations (after applying transform rotation)
transform offset y	Location Action	Offset to add to the y coordinate of parsed locations
transform offset z	Location Action	Offset to add to the z coordinate of parsed locations
transform pitch	Location Action	Pitch rotation in degrees to apply to parsed locations
transform roll	Location Action	Roll rotation in degrees to apply to parsed locations

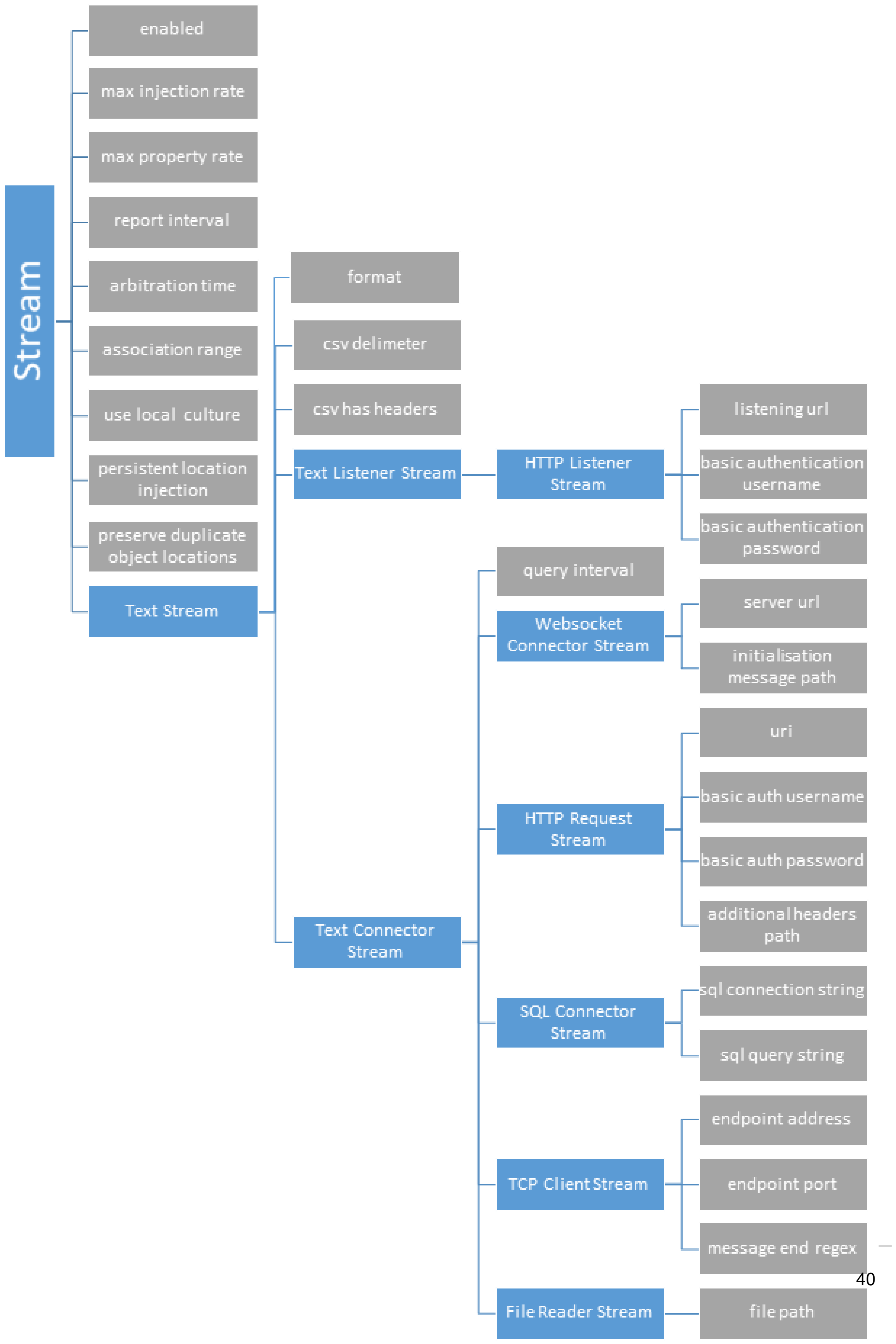
Parameter	Type	Purpose
transform yaw	Location Action	Yaw rotation in degrees to apply to parsed locations
unknown values	Tag Battery Action	Comma-separated values to parse as unknown
use local timezone	Location Action	Whether parsed times are assumed to be local or UTC times
warning values	Tag Battery Action	Comma-separated values to parse as warning
x identity	Cartesian Location Action	Identity of the x coordinate field in source data
y identity	Cartesian Location Action	Identity of the y coordinate field in source data
z identity	Cartesian Location Action	Identity of the z coordinate field in source data
zone	Location Action	Inclusion/exclusion zones to use

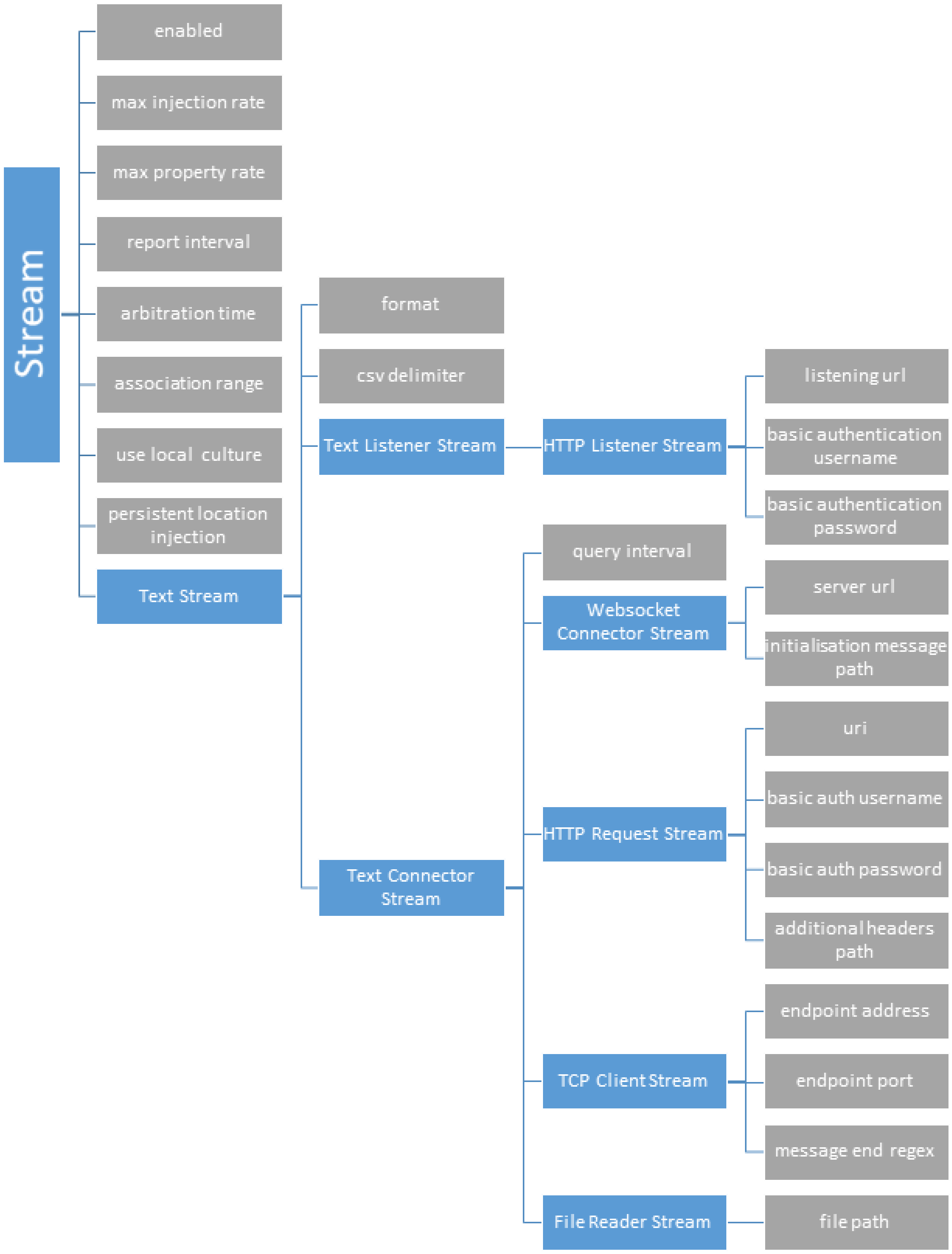
Miscellaneous Parameters

Parameter	Type	Purpose
latitude	GPS Reference Point	Latitude of the reference point
longitude	GPS Reference Point	Longitude of the reference point
x	GPS Reference Point	Platform x coordinate of the reference point
y	GPS Reference Point	Platform y coordinate of the reference point

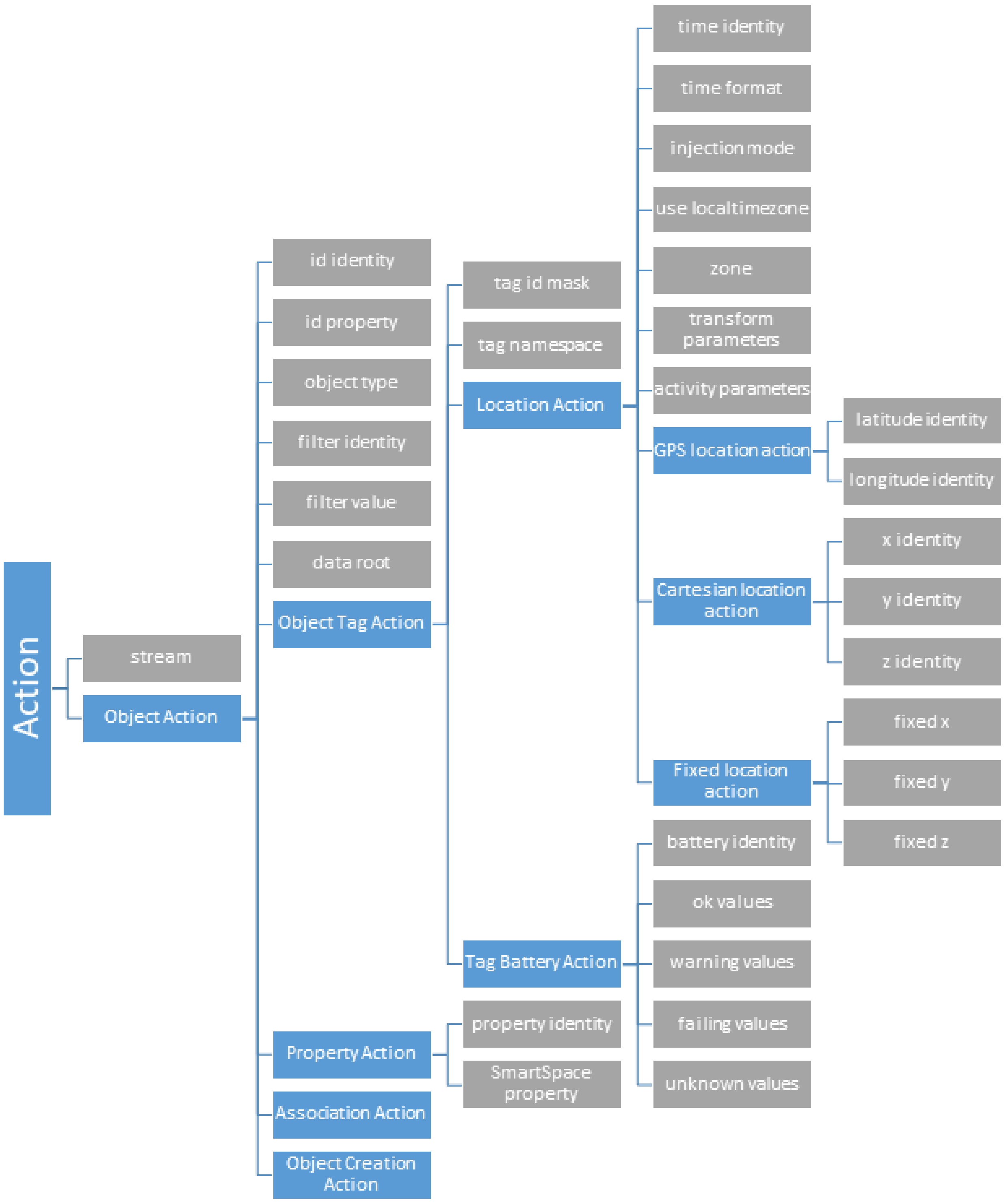
Types and parameters tree view

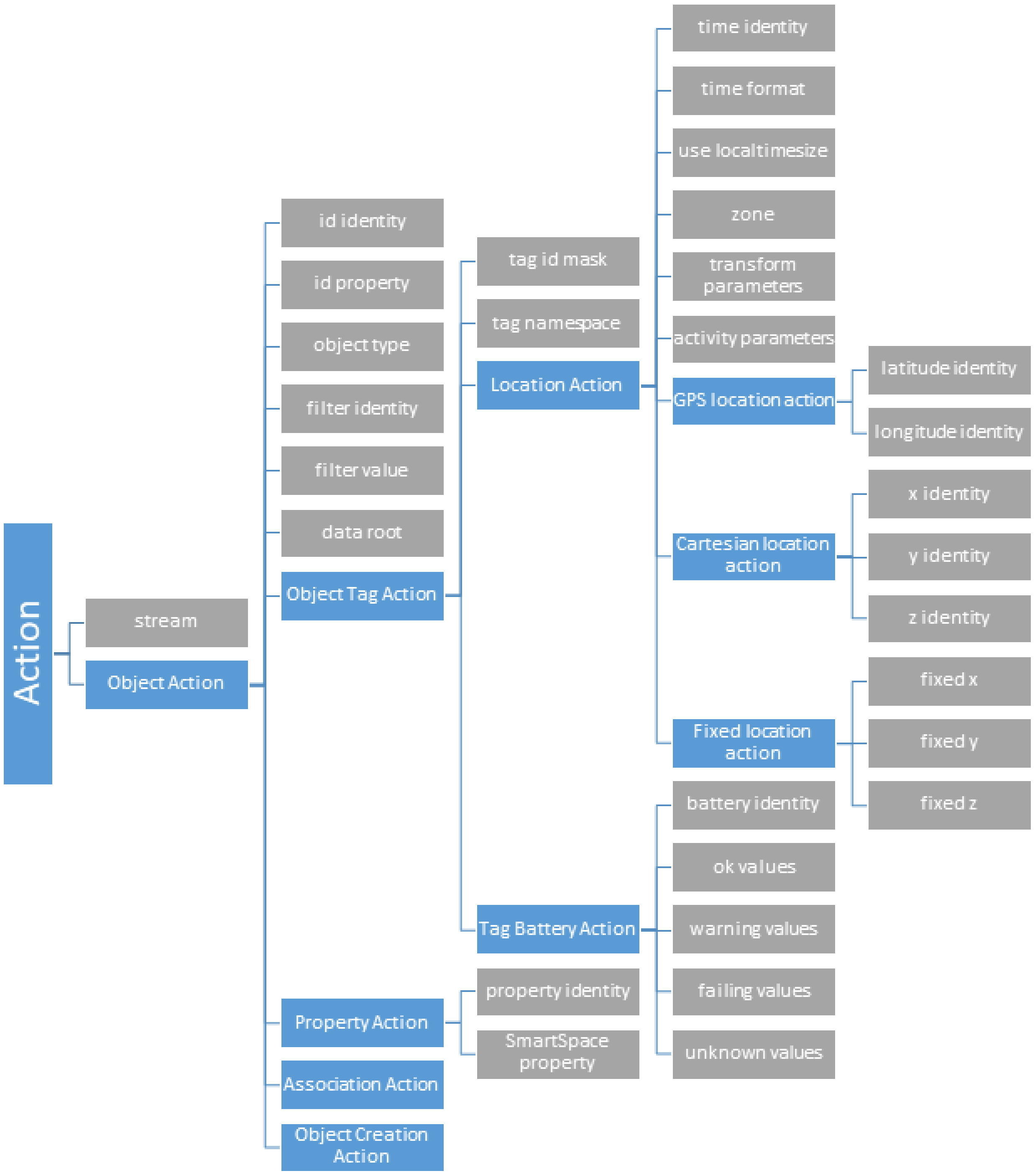
Streams


Stream
enabled
max injection rate
max property rate
report interval
arbitration time
association range
use local culture
persistent location injection
preserve duplicate object locations
Text Stream
format
csv delimeter
csv has headers
Text Listener Stream
HTTP Listener Stream
listening url
basic authentication username
basic authentication password
Text Connector Stream
query interval
Websocket Connector Stream
server url
initialisation message path
HTTP Request Stream
uri
basic auth username
basic auth password
additional headers path
SQL Connector Stream
sql connection string
sql query string
TCP Client Stream
endpoint address
endpoint port
message end regex
File Reader Stream
file path

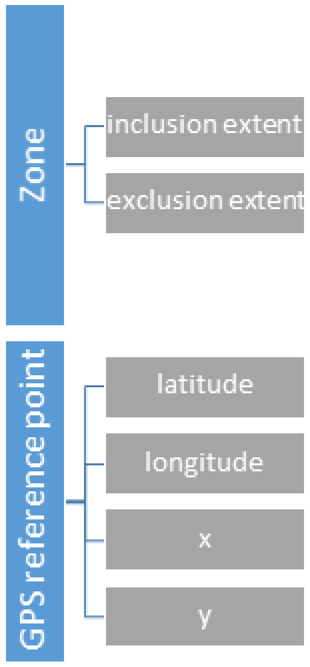

Stream
enabled
max injection rate
max property rate
report interval
arbitration time
association range
use local culture
persistent location injection
Text Stream
format
csv delimiter
Text Listener Stream
HTTP Listener Stream
listening url
basic authentication username
basic authentication password
Text Connector Stream
query interval
Websocket Connector Stream
server url
initialisation message path
HTTP Request Stream
uri
basic auth username
basic auth password
additional headers path
TCP Client Stream
endpoint address
endpoint port
message end regex
File Reader Stream
file path

Actions


Action
stream
Object Action
id identity
id property
object type
filter identity
filter value
data root
Object Tag Action
Property Action
Association Action
Object Creation Action
tag id mask
tag namespace
Location Action
Tag Battery Action
property identity
SmartSpace property
time identity
time format
injection mode
use localtimezone
zone
transform parameters
activity parameters
GPS location action
Cartesian location action
Fixed location action
battery identity
ok values
warning values
failing values
unknown values
latitude identity
longitude identity
x identity
y identity
z identity
fixed x
fixed y
fixed z


Action
stream
Object Action
id identity
id property
object type
filter identity
filter value
data root
Object Tag Action
tag id mask
tag namespace
Location Action
time identity
time format
use localtimesize
zone
transform parameters
activity parameters
GPS location action
latitude identity
longitude identity
Cartesian location action
x identity
y identity
z identity
Fixed location action
fixed x
fixed y
fixed z
Tag Battery Action
battery identity
ok values
warning values
failing values
unknown values
Property Action
property identity
SmartSpace property
Association Action
Object Creation Action

Miscellaneous Parameters



Trace Messages

The External data connector has several trace streams to help monitor performance and spot issues. These can be enabled using the **platform_monitor** configuration parameter.

The trace streams available are as follows:

- **data_connector**

  Periodic messages giving an overview of the number/rate of events and errors occurring. The rate of these messages is controlled with the **report interval** parameter.

- **data_connector_debug**

  Verbose messages giving real-time information on received data and errors encountered.

  Not recommended for regular use.

Understanding reports

data_connector stream

When enabled, the data_connector stream will output periodic reports summarizing the number of events and errors that occurred over the report period. These reports are grouped by action/component and look like the following:

```
[01/08/2019 18:01:58] data_connector: HttpRequester: Reports for last 10
seconds:
[01/08/2019 18:01:58] data_connector: HttpRequester: 1 HTTP(S) requests
completed, 0 requests failed.
[01/08/2019 18:01:58] data_connector: HttpRequester: 1 JSON objects/arrays
deserialized, 0 deserialization errors, 0 errors parsing action root elements,
1 JSON objects passed to actions
[01/08/2019 18:01:58] data_connector: HttpRequester: 1 locations for
injection, 0 discarded as outside cells, 0 discarded due to arbitration
[01/08/2019 18:01:58] data_connector: HttpRequester: 1 properties for
settings, 1 discarded as value unchanged, 0 errors setting value
[01/08/2019 18:01:59] data_connector: HttpRequester: CartesianAction: 1
objects for parsing, with 0 object/tag retrieval errors, 0 removed by data
fields filter, 0 removed by location filter, 0 location/time parsing errors
[01/08/2019 18:01:59] data_connector: HttpRequester: PropertyAction: 1 objects
for parsing, with 0 object/tag retrieval errors, 0 removed by data fields
filter, 0 unrecognised SmartSpace properties, 0 property value parsing errors
[01/08/2019 18:01:59] data_connector: HttpRequester: TBAction: 1 objects for
parsing, with 0 object/tag retrieval errors, 0 removed by data fields filter,
0 status parsing errors
```

Each report has a similar format with the general format as follows:

- The first item in the report is the number of operations/attempts that occurred in this report period
- Subsequent numbers are the number of these total operation/attempts that had issues/errors

data_connector_debug stream

Following is an example of support for actions under data_connector_debug:

```
data_connector_debug: Product Import: Http listener received "{
"id":"12365-567",
"project":"Broadsword",
"owner":"Andy"
}
"
data_connector_debug: Product Import: JsonParser::ParseValuesForActions
getting root ""
data_connector_debug: Product Import: JsonParser::ParseValuesForActions found
single object root ""
data_connector_debug: Product Import: JsonParser::ParseData evaluating actions
using input values: id:12365-567, project:Broadsword, owner:Andy
data_connector_debug: Product Import: ObjectCreationAction/Blade Creation:
queueing request: 12365-567
data_connector_debug: Product Import: PropertyAction/Blade-Owner: needed non-
nil but found nil object id for 12365-567
data_connector_debug: Product Import: PropertyAction/Blade-Project: needed
non-nil but found nil object id for 12365-567
data_connector_debug: Product Import: Repeating parse/actions phase to take
account of newly-created objects
data_connector_debug: Product Import: JsonParser::ParseValuesForActions
getting root ""
data_connector_debug: Product Import: JsonParser::ParseValuesForActions found
single object root ""
data_connector_debug: Product Import: JsonParser::ParseData evaluating actions
using input values: id:12365-567, project:Broadsword, owner:Andy
data_connector_debug: Product Import: ObjectCreationAction/Blade Creation:
needed nil but found non-nil object id for 12365-567
data_connector_debug: Product Import: PropertyAction/Blade-Owner: queueing
request: 12365-567 owner = Andy
data_connector_debug: Product Import: PropertyAction/Blade-Project: queueing
request: 12365-567 project = Broadsword
```

In this case we are tracing a listener stream for JSON data and what is happening is described below:

1. The stream 'Product Import' prints out the data it has received verbatim.
2. The parser will tell us whether it locates the root object. (Because this is JSON we use a JsonParser but this should be similar for other data formats.)
3. If the root object is located, then the parser will tell us the data that it is using to evaluate actions.
4. Here we have three actions: 'Blade Creation', which is a ObjectCreationAction; Blade-Owner, which is a PropertyAction; Blade-Project, which is a PropertyAction. Each action will

print data that includes its type/name at the start and provides s simple commentary on what it does. In this case:

a. Blade Creation finds the ID value 12365-567 and discovers that this value isn't already created and queues a request to create it.
b. Blade-Owner finds the same ID value but is unable to set a property for it because it hasn't yet been created (it was just queued for creation).
c. Blade-Project does the same.
d. The actions are then performed (just the single creation action this time).
e. In this case, because an object has just been created, the actions are done again to see if any more of them can now be achieved.
f. Now Blade-Owner finds the ID value which has been created and queues a request to set the property.
g. Blade-Project does the same.
h. The actions are then performed (two property set actions this time).

5. The final state has three changes to the data: an object called '12365-567' created, its 'owner' property set to "Andy"; and its 'project' property set to "Broadsword".

Ensuring EDC HTTP Listener Streams are Implemented Securely

The External data connector includes an *HTTP listener* option, which allows remote systems to push data to SmartSpace. It is important to understand the security implications of this feature, because the External data connector by itself is not intended to provide secure authentication or access control features. Therefore in a production system some extra steps are required to secure the interface:

1. ***Best practice – use a reverse proxy***: The listener should bind to a loopback port and accept connections from a reverse proxy such as Apache, IIS or NGINX, which will be responsible for all user authentication and control, and will only forward requests that satisfy its security requirements
2. ***Weaker alternative – use network access controls***: The listener should bind to an accessible network port, and firewall rules should restrict connection to the relevant external system(s) only. This ensures that the service is not just open to arbitrary users, but doesn't provide the full set of security features that (1) does.
3. ***Weak security – no protection***: The listener binds to an accessible network port without access restrictions. In principle on a public network this approach may open the EDC listener to arbitrary remote connectors, which would be able to change application data via the EDC. Hence this should only be used for internal experimentation, demoware and similar applications.

Identity language

To help with parsing of complex, nested data structures, the service uses a language, mirroring C# syntax, to help define the significance of data members in stream data. An identity is a name or sequence of names describing the full path to a data field in data messages. Identities are read from left to right, with the leftmost data field name being a top-level data field name and depth increasing as you move right. An empty identity represents the root element. The specific syntax of identities is specific to the format of the data.

JSON

Identities are made up of sequences of JSON object keys, starting with a key in the top level object. A '.' is used to denote a nested object and '[x]' is used to denote a fixed index in an array where x is the index, starting from 0. For example, the root of the locations in the JSON below is "Locations" and the y coordinate of tag1 is "Locations[0].Coords[1]".

```
{
   "Irrelevant": "some_data",
   "Locations": [
   {
      "name": "tag1",
      "Coords": [
         1,
         2,
         3
      ]
   },
   {
      "name": "tag2",
      "Coords": [
         1,
         2,
         3
      ]
   }],
   "More irrelevant": "some_data"
}
```

The '$' wildcard can be used as the index number to signify the last element in an array.

XML

Identities start with an element tag in the root element, each nested element or attribute is denoted with a '.'. Attributes can only be the rightmost element.

In the same way as described for *JSON*, above,'[x]' can be used to denote a fixed index in an array where x is the index, starting from 0.

CSV

There is no nesting of data in CSVs. Identities are either empty, for the "root element", or the name of a column heading. Column headings are optional, and in the case of CSV without column headings, identities should take the form '[x]' where x is the column index, starting from 0.

Single-field data

There is no nesting of data in single-field data. Identities should either be empty, for the "root element", or 'field', for the single field.

Tag namespaces

Tag namespaces are used by the External data connector to support non-Ubisense tags for ObjectTagActions. Currently only EPC tag IDs up to 128 bits in length are supported.

Tag namespaces can be specified by adding the appropriate prefix to the start of a tag id followed by "::", e.g. EPC-64::1234567890abcdef. Namespaces can be used as tag IDs in SmartSpace, in tag association or for tag parameters for External data connector service parameters. The tag namespace parameter can be used to automatically prepend the prefix to parsed tag IDs in external data (do not include the "::" in the parameter value).

Some actions or functionality may not be supported for non-Ubisense tags, e.g. battery and activity data.

Supported Namespace Prefixes

- EPC-64
- EPC-96
- EPC-128

Supported protocols

The following outlines the protocols and formats supported by the External data connector service.

Protocols

HTTP(S) connector

The EDC supports connection to external systems via both HTTP and HTTPS with basic authentication. Custom request headers are also supported. Data provided by the external system should be in a valid text format.

HTTP(S) receiver

The EDC supports running as an HTTP(S) server, receiving data messages via HTTP or HTTPS requests. Optionally supports use of basic authentication of incoming requests. Data sent to the service should be in a valid text format.

Web socket connector

The EDC can retrieve data from external systems via a web socket client, optionally supporting basic authentication. The client will connect, optionally send a configurable initialization message then wait for the server to send response(s). After a configurable period of no server communication, the client will close and attempt to reconnect again, sending the initialization message on reconnect. A compatible server should periodically send the required data to connected clients (optionally after an initialization message), or it should send the required information once to a connected client and rely on the timeout/reconnect functionality to send more data when the EDC reconnects.

TCP

The EDC supports running as a TCP client. The EDC will connect to a TCP server and listen for data messages in a valid text format. The EDC does not perform any connection management, such as sending keepalives, but will attempt to reconnect if disconnected. Received messages should have a termination point which can be determined by a regular expression, e.g. a newline signifies a complete message.

SQL

The EDC supports retrieval of table rows from a SQL server using conventional connection and query strings. Retrieved data will converted to CSV format; SQL Connector Streams should have their format parameter set to CSV.

File

The EDC can retrieve data directly from a text file.

Formats

Currently, the service only supports tag/object data. This may be location, property or battery/activity data. Formats will be described in terms of the following definitions:

- *Field* – A datum in a data message, identifiable by an identity e.g. object id, a location coordinate or a field containing other, child fields.
- *Action object* – A collection of data fields, grouped together logically in a data message. Together, these fields contain all information required by an action to operate for a single tag/object.
- *Root field* – For nested source data formats. The innermost field containing all action object (s) in a data message.

Text formats

For sources where the data retrieved is a string.

JSON

JSON string data should be a valid JSON object or array of objects. The action object(s) should be JSON object(s), either as a single JSON object or as an array. All relevant fields (and all ancestors of that field) must be a named JSON value or a value in an array at a fixed index.

Fields – All relevant fields must be a named JSON value or a value in an array at a fixed index.

Action objects – All fields for a single object/tag should be within a JSON object, either at the top level of this object or nested with objects or arrays at a fixed index.

Root field – Action objects can be defined as a JSON object/array at the top level or within a nested field.The root should be a JSON object whose value is an object or array of objects.

Valid examples

```
{
   "name":"object1",
   "location":[
      12.3,
      43.7,
      0.0
   ],
   "irrelevant_data":80946
}

{
   "system_name":"external source 1",
   "status":"good",
   "location_data":{
      "location_count":2,
      "locations":[
         {
            "name":"object1",
            "x":3.2,
            "y":4.6,
            "z":1.0
         },
         {
            "name":"object2",
            "x":2.6,
            "y":15.7,
            "z":0.8
         }
      ]
   }
}
```

XML

XML data should be a valid XML string. The action object(s) should be an element with one or more child elements (possibly nested) or attributes. Field values can be the contents of the elements or attribute values (of the parent element of child elements).

Fields – Relevant fields can be child elements with contents or attributes of the parent or child elements.

Action objects – Fields for a single object/tag should be contained in a single element. When there are multiple action objects in a single data message, they should be sibling elements with the same name.

Root field – Action objects can be in XML elements at any level in the tree.

Valid examples

```
<?xml version="1.0"?>
<ArrayOfLocationObject xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <LocationObject>
    <name>TestObject1</name>
    <x>4.4679310077186347</x>
    <y>7.6369389699012684</y>
    <z>0</z>
  </LocationObject>
  <LocationObject>
    <name>TestObject2</name>
    <x>6.4845871801881989</x>
    <y>7.4083634868303143</y>
    <z>0</z>
  </LocationObject>
</ArrayOfLocationObject>
```

CSV

CSV data must start with a row of column headings.

CSV data can optionally start with a row of column headings.

Fields – A column in a single row.

Actions Objects – Each row is treated as an action object.

Root field – CSV data is not nested.

Valid examples

- With column headings

```
id,x,y,z
TestObject1,4.4679,7.6369,0.0
TestObject2,6.4856,7.4084,0.0
```

- Without column headings

```
Tag1,data
Tag2,other_data
```

evaluates to the following id:value pairs:

```
[0]:Tag1, [1]:data
[0]:Tag2, [1]:other_data
```

Single-field data

Single-field data should be a single line of data, representing a single data field.

Fields – A single line string.

Action Objects – Single-field data should only have a single action object, each row (and therefore each complete single-field data message) is treated as an action object.

Root field – Single-field data is not nested.

Valid examples

```
StorageArea1
```

Example: Connecting a Quuppa tracking system to SmartSpace

This section gives and example of configuring the External data connector to take location information from a Quuppa Positioning Engine and inject it into SmartSpace.

Overview

Quuppa provide data through a Web Service API in JSON.

The EDC points directly to the JSON data stream using the web address of the Quuppa Positioning Engine (QPE) plus some additional parameters in the web address link. This is configured in a SmartSpaceEDC Stream Object (described in *Creating the Stream Object*).

In addition to the parameters for the stream object, an action object and its parameters are required (described in *Creating the Action Object*) to process the location information.

Quuppa Positioning Engine APIs

The *Quuppa Positioning Engine APIs* document can be downloaded from Quuppa Customer Portal on the *Quuppa website*.

Connecting the SmartSpace EDC to Quuppa using the getTagPosition method

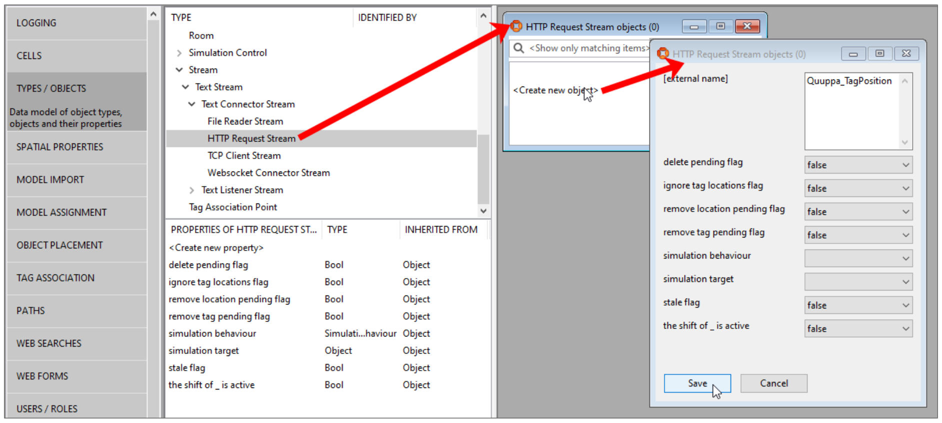
Note: This example is based on version *2.0* of the *Quuppa Positioning Engine API*.

Creating the Stream Object

We need to create a *SmartSpace EDC Stream Object* that will point to the QPE and regularly request data from it.

1. In SmartSpace Config, in the TYPES / OBJECTS task, we create an HTTP Request Stream object called "Quuppa_TagPosition".

   We drag out the HTTP Request Stream type from the type hierarchy (it's under Stream > Text Stream > Text Connector Stream), double-click **<Create new object>** and enter Quuppa_TagPosition (or whatever name is required) for the object's name.



TYPE	IDENTIFIED BY
Room	
Simulation Control	
Stream	
Text Stream	
Text Connector Stream	
File Reader Stream	
HTTP Request Stream	
TCP Client Stream	
Websocket Connector Stream	
Text Listener Stream	
Tag Association Point	

PROPERTIES OF HTTP REQUEST ST...	TYPE	INHERITED FROM
<Create new property>		
delete pending flag	Bool	Object
ignore tag locations flag	Bool	Object
remove location pending flag	Bool	Object
remove tag pending flag	Bool	Object
simulation behaviour	Simulati...haviour	Object
simulation target	Object	Object
stale flag	Bool	Object
the shift of _ is active	Bool	Object

[external name]	Quuppa_TagPosition
delete pending flag	false
ignore tag locations flag	false
remove location pending flag	false
remove tag pending flag	false
simulation behaviour	
simulation target	
stale flag	false
the shift of _ is active	false

Click **Save**.

2. We need to configure this "Quuppa_TagPosition" stream to point to the QPE by defining the HTTP request with some parameters.

   In SmartSpace Config, in the SERVICE PARAMETERS task, choose External data connector from the dropdown, drag out the HTTP Request Stream type from the type hierarchy (it's under Stream > Text Stream > Text Connector Stream), and double-click Quuppa_ TagPosition. Click **Edit**.

   In this case, set the arbitration time to "0" as the asset objects are driven only by the Quuppa tag, so there is no arbitration to do between two or more tracking systems driving each object (e.g. Quuppa indoors and GPS outdoors).

   We set the “query interval” to "1". The value we here depends on the use case, for example if the Quuppa tags were configured with an update rate of 30 seconds, there would be no need for a query interval of 1 sec, but the value would be around 10 seconds (the default).

   The URL link in this example is:
   http://192.168.28.139:8080/qpe/getTagPosition?version=2&maxAge=5000

   where :

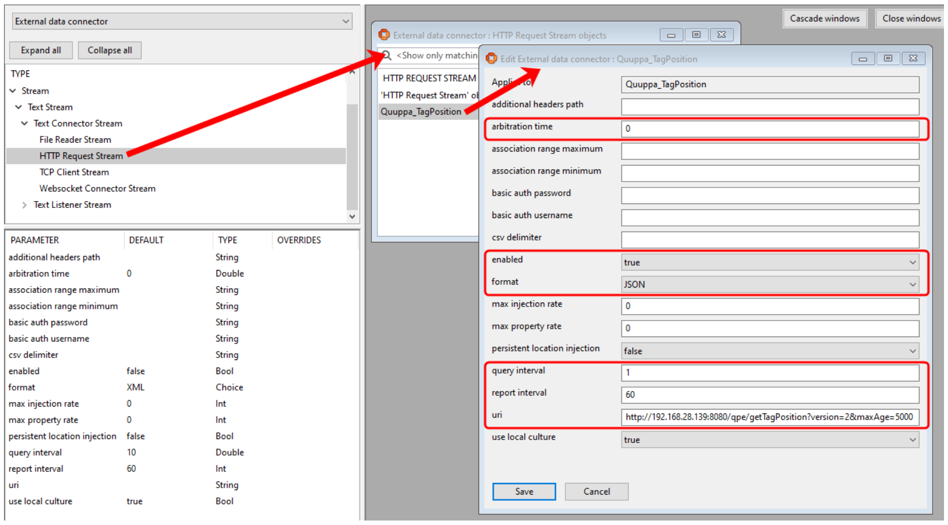
   - 192.168.28.139 is the IP of the QPE server
   - 8080 the port used by Quuppa for the request (it can be different according to the configuration of the Quuppa API on their side)
   - getTagPosition the method used to request Quuppa tag position data
   - version=2 is a mandatory parameter and is the version of the QPE API

- maxAge=XXXX is an optional parameter. It defines the maximum age in milliseconds for the information. For example, by defining '&maxAge=1000', no results are returned if the last tag position update is older than 1 second.

  In our example, no data older than 5 seconds is returned.

Other parameters for the Quuppa URL are available and are described in the *Quuppa Positioning Engine APIs* document.

We set the other parameters as shown below:



PARAMETER	DEFAULT	TYPE	OVERRIDES
additional headers path		String	
arbitration time	0	Double	
association range maximum		String	
association range minimum		String	
basic auth password		String	
basic auth username		String	
csv delimiter		String	
enabled	false	Bool	
format	XML	Choice	
max injection rate	0	Int	
max property rate	0	Int	
persistent location injection	false	Bool	
query interval	10	Double	
report interval	60	Int	
uri		String	
use local culture	true	Bool	

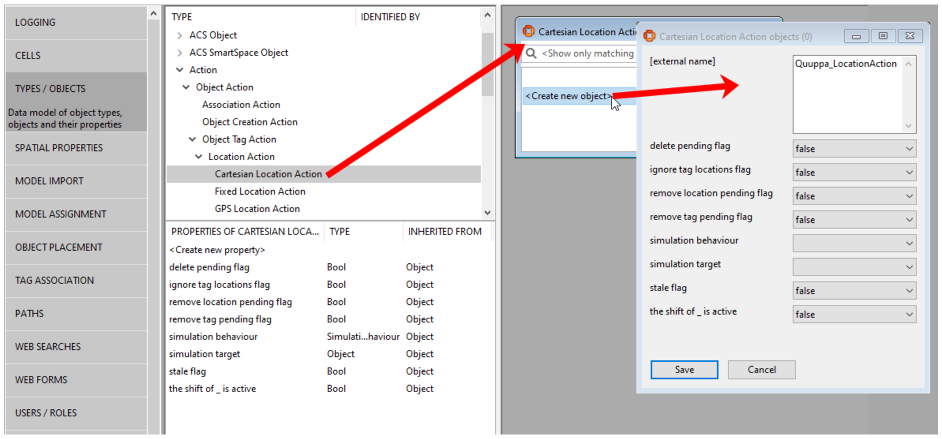
Appli to	Quuppa_TagPosition
additional headers path	
arbitration time	0
association range maximum	
association range minimum	
basic auth password	
basic auth username	
csv delimiter	
enabled	true
format	JSON
max injection rate	0
max property rate	0
persistent location injection	false
query interval	1
report interval	60
uri	http://192.168.28.139:8080/qpe/getTagPosition?version=2&maxAge=5000
use local culture	true

Click **Save**.

Creating the Action Object

We are now able to request Quuppa tag information every "query interval" seconds, here every 1 second, but we have not defined what to do with the data. So, we need to process this stream and feed the right objects into SmartSpace by creating a *SmartSpace Action object*.

1. In SmartSpace Config, in the TYPES / OBJECTS task, we create a Cartesian Location Action object called "Quuppa_LocationAction".
2. We drag out the Cartesian Location Action type from the type hierarchy (it's under Action > Object Action > Object Tag action > Location Action), double-click **<Create new object>** and enter Quuppa_LocationAction (or whatever name is required) for the object's name.



PROPERTIES OF CARTESIAN LOCA...	TYPE	INHERITED FROM
<Create new property>		
delete pending flag	Bool	Object
ignore tag locations flag	Bool	Object
remove location pending flag	Bool	Object
remove tag pending flag	Bool	Object
simulation behaviour	Simulati...haviour	Object
simulation target	Object	Object
stale flag	Bool	Object
the shift of _ is active	Bool	Object

[external name]	Quuppa_LocationAction
delete pending flag	false
ignore tag locations flag	false
remove location pending flag	false
remove tag pending flag	false
simulation behaviour	
simulation target	
stale flag	false
the shift of _ is active	false

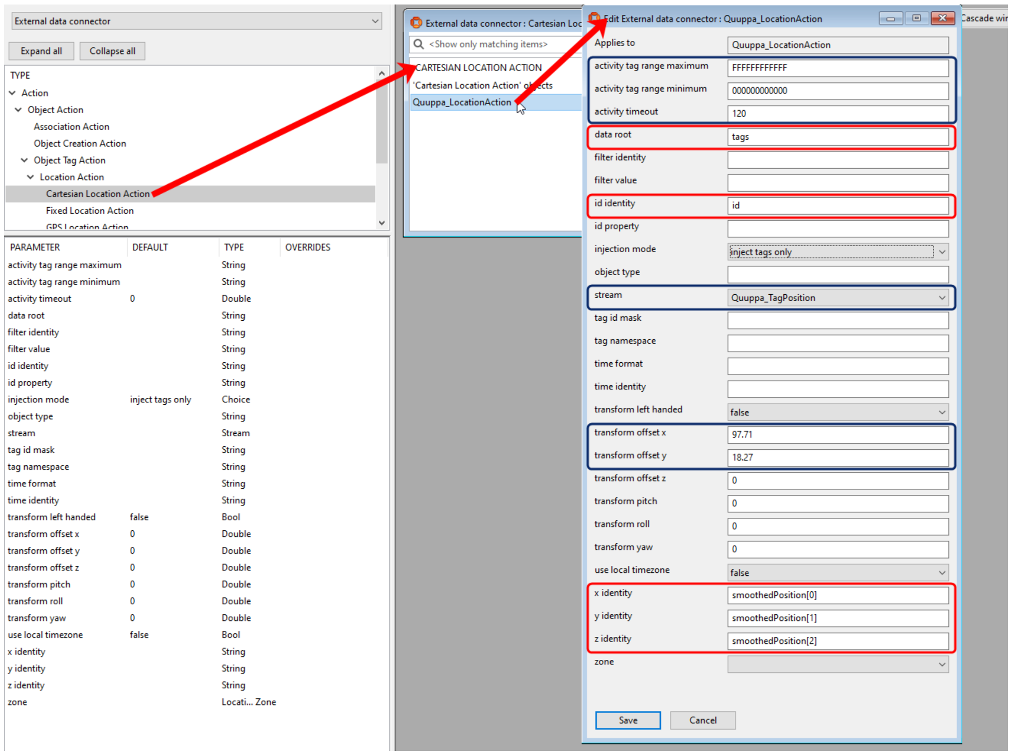
Click **Save**.

3. We need to configure this "Quuppa_LocationAction" action to drive our SmartSpace assets. Below we describe two methods to do this:
    - Method A: the Quuppa action will point directly to the tag associated with the asset object (similar to the DIMENSION4 location platform when it drives the tag associated with an asset object with the association being made in the using the TAG ASSOCIATION task in SmartSpace Config)
    - Method B: the Quuppa action will point to a property of the asset (which must have a unique name)

Method A

In SmartSpace Config, in the SERVICE PARAMETERS task, choose External data connector from the dropdown, drag out the Cartesian Location Action type from the type hierarchy (it's under Action > Object Action > Object Tag action > Location Action), and double-click Quuppa_LocationAction. Click **Edit**.

Configure the parameters as shown below.



External data connector

Expand all | Collapse all

TYPE
- Action
  - Object Action
    - Association Action
    - Object Creation Action
    - Object Tag Action
      - Location Action
        - Cartesian Location Action
        - Fixed Location Action
        - GPS Location Action

PARAMETER	DEFAULT	TYPE	OVERRIDES
activity tag range maximum		String	
activity tag range minimum		String	
activity timeout	0	Double	
data root		String	
filter identity		String	
filter value		String	
id identity		String	
id property		String	
injection mode	inject tags only	Choice	
object type		String	
stream		Stream	
tag id mask		String	
tag namespace		String	
time format		String	
time identity		String	
transform left handed	false	Bool	
transform offset x	0	Double	
transform offset y	0	Double	
transform offset z	0	Double	
transform pitch	0	Double	
transform roll	0	Double	
transform yaw	0	Double	
use local timezone	false	Bool	
x identity		String	
y identity		String	
z identity		String	
zone		Locati... Zone	

External data connector : Cartesian Loc

<Show only matching items>

CARTESIAN LOCATION ACTION
'Cartesian Location Action' objects
Quuppa_LocationAction

Edit External data connector : Quuppa_LocationAction

Applies to	Quuppa_LocationAction
activity tag range maximum	FFFFFFFFFFFF
activity tag range minimum	000000000000
activity timeout	120
data root	tags
filter identity	
filter value	
id identity	id
id property	
injection mode	inject tags only
object type	
stream	Quuppa_TagPosition
tag id mask	
tag namespace	
time format	
time identity	
transform left handed	false
transform offset x	97.71
transform offset y	18.27
transform offset z	0
transform pitch	0
transform roll	0
transform yaw	0
use local timezone	false
x identity	smoothedPosition[0]
y identity	smoothedPosition[1]
z identity	smoothedPosition[2]
zone	

Click **Save**.

The parameters shown boxed in red in the example above refer to the content of the Quuppa JSON data. An example of Quuppa JSON that can be received via an HTTP request is shown below with the corresponding parameters identified.

Parameters in the blue boxes relate to SmartSpace's handling of the received data.

```
{
  "code": 0,
  "command":
"http://localhost:8080/qpe/getTagPosition?version=2&humanReadable=true&maxAge=5000.htm
l",
  "message": "TagPosition",
  "responseTS": 1520946349517,
  "status": "Ok",
  "tags": [
    {
      "areaId": "Tracking001_2D",
      "areaName": "SAM146",
      "color": "#FF0000",
      "coordinateSystemId": "CoordSys001",
      "coordinateSystemName": "SAM146",
      "covarianceMatrix": [
        1.82,
        0.09,
        0.09,
        0.87
      ],
      "id": "0cb2b725c5f0",
      "name": "BAT35_SAM146_0135",
      "position": [
        15.2,
        7.45,
        1.2
      ],
      "positionAccuracy": 0.41,
      "positionTS": 1520928431109,

      "smoothedPosition": [
        15.2,
        7.45,
        1.2
      ],
```

"tags" is the "data root" parameter

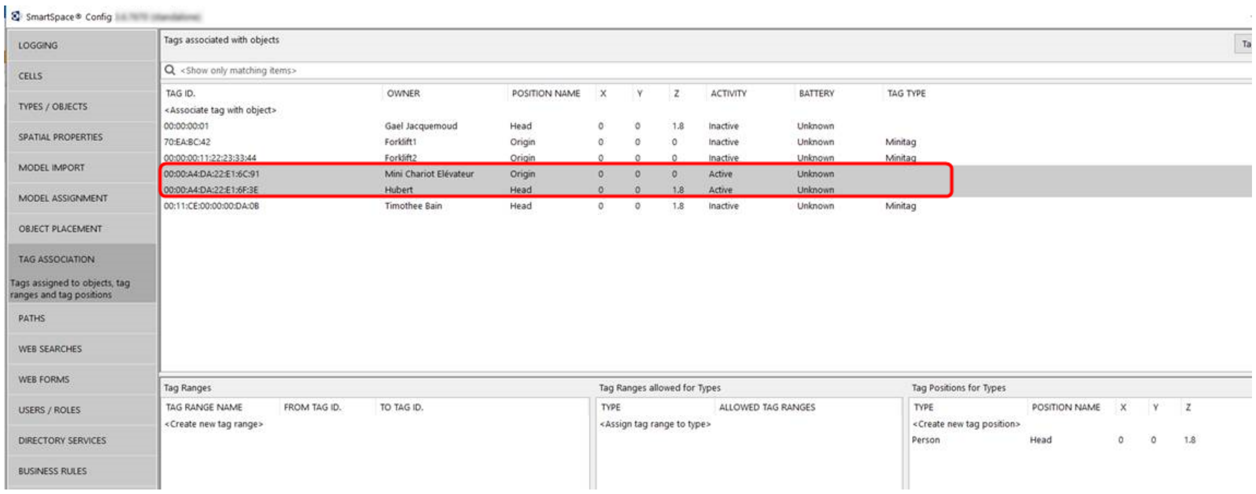
"id" is the "id identity"

The "smoothedPosition" co-ordinates correspond to the "x identity", "y identity" and "z identity" parameters

```
      "smoothedPositionAccuracy": 0.44,
      "zones": []
    },
    {
      "areaId": "Tracking001_2D",
      "areaName": "SAM146",
      "color": "#FF0000",
      "coordinateSystemId": "CoordSys001",
      "coordinateSystemName": "SAM146",
      "covarianceMatrix": [
        1.29,
        -0.1,
        -0.1,
        0.96
      ],
      "id": "0cb2b72478a7",
      "name": "BAT35_SAM146_0059",
      "position": [
        16.1,
        7.45,
        1.2
      ],
      "positionAccuracy": 0.34,
      "positionTS": 1520930414097,
      "smoothedPosition": [
        13.86,
        4.9,
        1.2
      ],
      "smoothedPositionAccuracy": 0.34,
      "zones": []
    },
       {
      "areaId": "Tracking002_2D",
      "areaName": "MagasinKME",
      "color": "#FF0000",
      "coordinateSystemId": "CoordSys002",
      "coordinateSystemName": "MagasinKME",
      "covarianceMatrix": [
        0.91,
        -0.01,
        -0.01,
        0.74
      ],
  "version": "2.1"
}
```

With the parameters defined, we need to make the association between Quuppa tags and SmartSpace objects, using a tag ID format with 16 hexadecimal digits. In SmartSpace Config, Open the TAG ASSOCIATION task and double-click **<Associate tag with object>** to enter details of the association.

Below is an example of the tag associations:



Tags associated with objects

TAG ID.	OWNER	POSITION NAME	X	Y	Z	ACTIVITY	BATTERY	TAG TYPE
<Associate tag with object>								
00:00:00:01	Gael Jacquemoud	Head	0	0	1.8	Inactive	Unknown	
70:EA:BC:42	Forklift1	Origin	0	0	0	Inactive	Unknown	Minitag
00:00:00:11:22:23:33:44	Forklift2	Origin	0	0	0	Inactive	Unknown	Minitag
00:00:A4:DA:22:E1:6C:91	Mini Chariot Elévateur	Origin	0	0	0	Active	Unknown	
00:00:A4:DA:22:E1:6F:3E	Hubert	Head	0	0	1.8	Active	Unknown	
00:11:CE:00:00:00:DA:0B	Timothee Bain	Head	0	0	1.8	Inactive	Unknown	Minitag

Tag Ranges

TAG RANGE NAME	FROM TAG ID.	TO TAG ID.
<Create new tag range>		

Tag Ranges allowed for Types

TYPE	ALLOWED TAG RANGES
<Assign tag range to type>	

Tag Positions for Types

TYPE	POSITION NAME	X	Y	Z
<Create new tag position>				
Person	Head	0	0	1.8

Method B

In this method, the Quuppa action will point to a property of the asset (*not* to its associated tag, as done in Method A).

1. After we have defined the Stream and its parameter, we need to create a "Quuppa Tag Id" property for your asset object whose value is unique. In SmartSpace Config, we use the TYPES / OBJECTS task to add a "Quuppa Tag Id" property for each object type that is tracked by a Quuppa tag. Then for each instance of the tracked object we must assign a Quuppa tag ID, using the format used in the *JSON file*.
2. In SmartSpace Config, we use the SERVICE PARAMETERS task to configure the parameters for the "Quuppa_LocationAction". For this method, we need to set the following parameters:
    - "object type" which should be the name of the SmartSpace object type tracked by Quuppa
    - "id property" which should be the name of the unique property we created for the tracked object which is set to the "Quuppa Tag Id".
3. We must then ensure SmartSpace asset objects tracked by Quuppa have a tag associated with them, either manually (with a dummy tag) or using the *Association Action*.

Connecting the SmartSpace EDC to Quuppa using the getTagData method

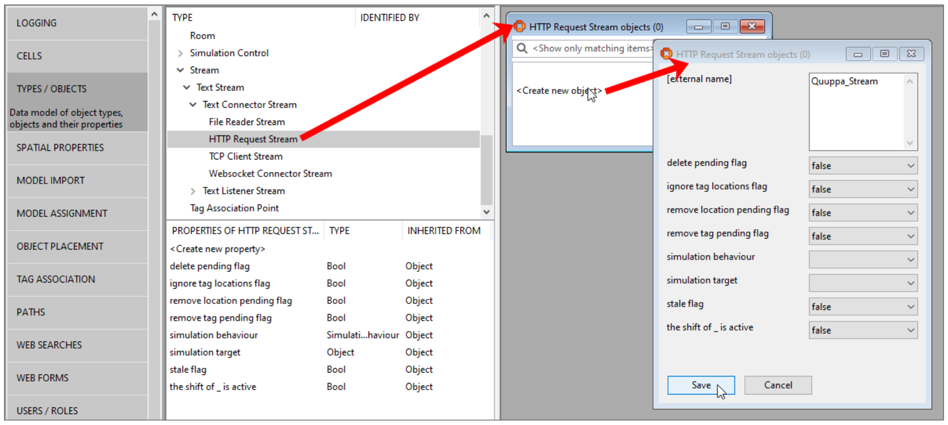
Note: This example is based on version *2.2* of the *Quuppa Positioning Engine API*.

Creating the Stream Object

We need to create a *SmartSpace EDC Stream Object* that will point to the QPE and regularly request data from it.

1. In SmartSpace Config, in the TYPES / OBJECTS task, we create an HTTP Request Stream object called "Quuppa_Stream".

   We drag out the HTTP Request Stream type from the type hierarchy (it's under Stream > Text Stream > Text Connector Stream), double-click <**Create new object**> and enter Quuppa_Stream (or whatever name is required) for the object's name.

PROPERTIES OF HTTP REQUEST ST...	TYPE	INHERITED FROM
<Create new property>		
delete pending flag	Bool	Object
ignore tag locations flag	Bool	Object
remove location pending flag	Bool	Object
remove tag pending flag	Bool	Object
simulation behaviour	Simulati...haviour	Object
simulation target	Object	Object
stale flag	Bool	Object
the shift of _ is active	Bool	Object

[external name]	Quuppa_Stream
delete pending flag	false
ignore tag locations flag	false
remove location pending flag	false
remove tag pending flag	false
simulation behaviour	
simulation target	
stale flag	false
the shift of _ is active	false

   Click **Save**.

2. We need to *configure* "Quuppa_Stream" to point to the QPE by defining the HTTP request with some parameters.

   In SmartSpace Config, in the SERVICE PARAMETERS task, choose External data connector from the dropdown, drag out the HTTP Request Stream type from the type hierarchy (it's under Stream > Text Stream > Text Connector Stream), and double-click Quuppa_Stream. Click **Edit**.

   In this case, set the arbitration time to "0" as the asset objects are driven only by the Quuppa tag, so there is no arbitration to do between two or more tracking systems driving each object (e.g. Quuppa indoors and GPS outdoors).

   We set the “query interval” to "1". The value we here depends on the use case, for example if the Quuppa tags were configured with an update rate of 30 seconds, there would be no need for a query interval of 1 sec, but the value would be around 10 seconds (the default).

   The URL link in this example is: *http://192.168.28.139:8080/qpe/getTagData&maxAge=5000*

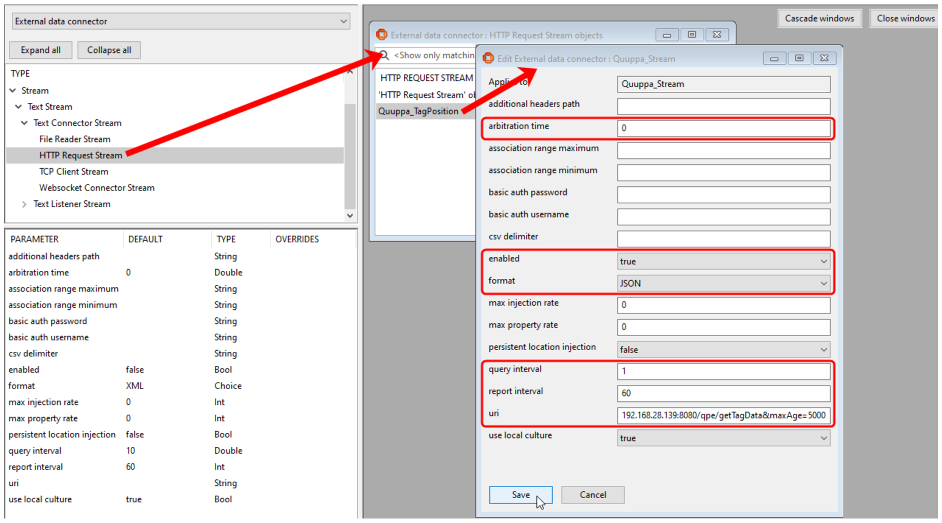
where :

- 192.168.28.139 is the IP of the QPE server
- 8080 the port used by Quuppa for the request (it can be different according to the configuration of the Quuppa API on their side)
- getTagData the method used to request Quuppa tag position data
- maxAge=XXXX is an optional parameter. It defines the maximum age in milliseconds for the information. For example, by defining '&maxAge=1000', no results are returned if the last tag position update is older than 1 second.

  In our example, no data older than 5 seconds is returned.

Other parameters for the Quuppa URL are available and are described in the *Quuppa Positioning Engine APIs* document.

We set the other parameters as shown below:



PARAMETER	DEFAULT	TYPE	OVERRIDES
additional headers path		String	
arbitration time	0	Double	
association range maximum		String	
association range minimum		String	
basic auth password		String	
basic auth username		String	
csv delimiter		String	
enabled	false	Bool	
format	XML	Choice	
max injection rate	0	Int	
max property rate	0	Int	
persistent location injection	false	Bool	
query interval	10	Double	
report interval	60	Int	
uri		String	
use local culture	true	Bool	

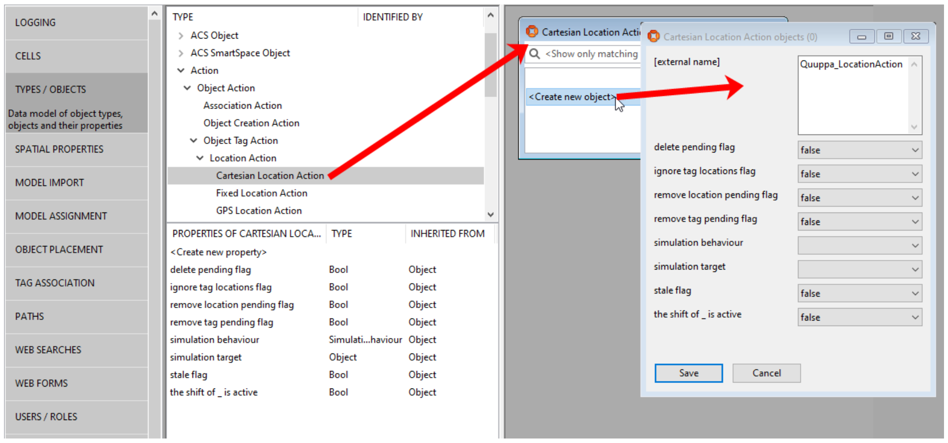
Appli to	Quuppa_Stream
additional headers path	
arbitration time	0
association range maximum	
association range minimum	
basic auth password	
basic auth username	
csv delimiter	
enabled	true
format	JSON
max injection rate	0
max property rate	0
persistent location injection	false
query interval	1
report interval	60
uri	192.168.28.139:8080/qpe/getTagData&maxAge=5000
use local culture	true

Click **Save**.

Creating the Action Object

We are now able to request Quuppa tag information every "query interval" seconds, here every 1 second, but we have not defined what to do with the data. So, we need to process this stream and feed the right objects into SmartSpace by creating a *SmartSpace Action object*.

1. In SmartSpace Config, in the TYPES / OBJECTS task, we create a Cartesian Location Action object called "Quuppa_LocationAction".
2. We drag out the Cartesian Location Action type from the type hierarchy (it's under Action > Object Action > Object Tag action > Location Action), double-click **<Create new object>** and enter Quuppa_LocationAction (or whatever name is required) for the object's name.



PROPERTIES OF CARTESIAN LOCA...	TYPE	INHERITED FROM
<Create new property>		
delete pending flag	Bool	Object
ignore tag locations flag	Bool	Object
remove location pending flag	Bool	Object
remove tag pending flag	Bool	Object
simulation behaviour	Simulati...haviour	Object
simulation target	Object	Object
stale flag	Bool	Object
the shift of _ is active	Bool	Object

[external name]	Quuppa_LocationAction
delete pending flag	false
ignore tag locations flag	false
remove location pending flag	false
remove tag pending flag	false
simulation behaviour	
simulation target	
stale flag	false
the shift of _ is active	false

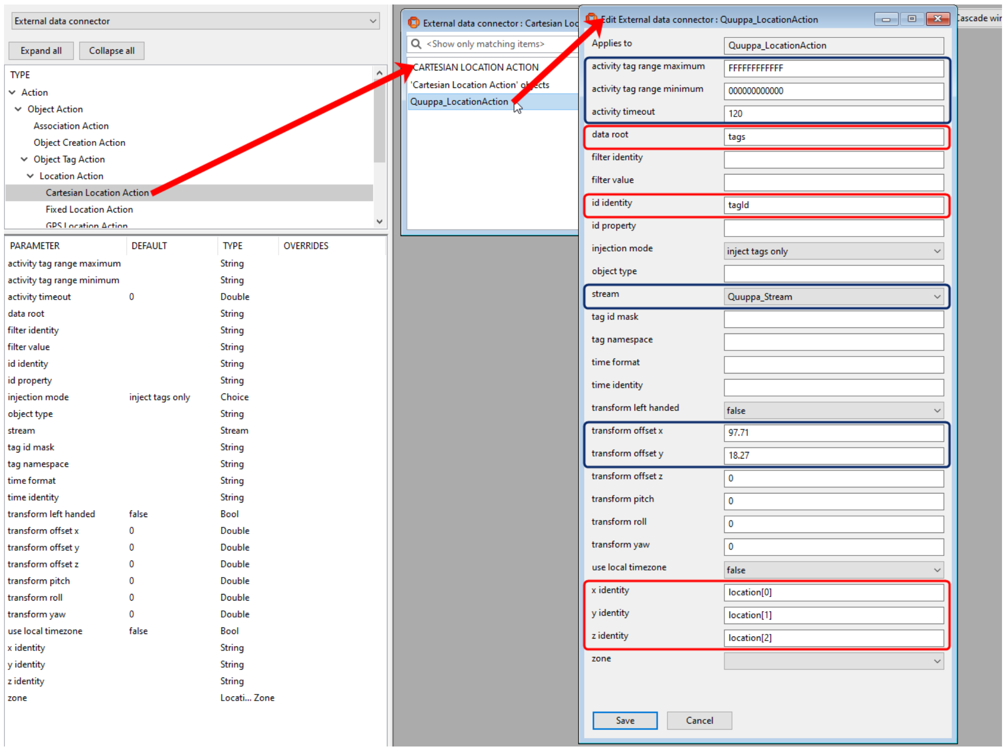
Click **Save**.

3. We need to *configure* this "Quuppa_LocationAction" action to drive our SmartSpace assets. Below we describe two methods to do this:
   - *Method A*: the Quuppa action will point directly to the tag associated with the asset object (similar to the DIMENSION4 location platform when it drives the tag associated with an asset object with the association being made in the using the TAG ASSOCIATION task in SmartSpace Config)
   - *Method B*: the Quuppa action will point to a property of the asset (which must have a unique name)

Method A

In SmartSpace Config, in the SERVICE PARAMETERS task, choose External data connector from the dropdown, drag out the Cartesian Location Action type from the type hierarchy (it's under Action > Object Action > Object Tag action > Location Action), and double-click Quuppa_ LocationAction. Click **Edit**.

Configure the parameters as shown below.



PARAMETER	DEFAULT	TYPE	OVERRIDES
activity tag range maximum		String	
activity tag range minimum		String	
activity timeout	0	Double	
data root		String	
filter identity		String	
filter value		String	
id identity		String	
id property		String	
injection mode	inject tags only	Choice	
object type		String	
stream		Stream	
tag id mask		String	
tag namespace		String	
time format		String	
time identity		String	
transform left handed	false	Bool	
transform offset x	0	Double	
transform offset y	0	Double	
transform offset z	0	Double	
transform pitch	0	Double	
transform roll	0	Double	
transform yaw	0	Double	
use local timezone	false	Bool	
x identity		String	
y identity		String	
z identity		String	
zone		Locati... Zone	

Applies to	Quuppa_LocationAction
activity tag range maximum	FFFFFFFFFFFF
activity tag range minimum	000000000000
activity timeout	120
data root	tags
filter identity	
filter value	
id identity	tagId
id property	
injection mode	inject tags only
object type	
stream	Quuppa_Stream
tag id mask	
tag namespace	
time format	
time identity	
transform left handed	false
transform offset x	97.71
transform offset y	18.27
transform offset z	0
transform pitch	0
transform roll	0
transform yaw	0
use local timezone	false
x identity	location[0]
y identity	location[1]
z identity	location[2]
zone	

Click **Save**.

The parameters shown boxed in red in the example above refer to the content of the Quuppa JSON data. An example of Quuppa JSON that can be received via an HTTP request is shown below with the corresponding parameters identified.

Parameters in the blue boxes relate to SmartSpace's handling of the received data.

```
{
   "code":"0",
   "status":"Ok",
   "command":http://192.168.28.139:8080/qpe/getTagData&maxAge=5000,
   "message":"Tag data",
   "responseTS":1657007507616,
   "version":"1.0",
   "formatId":"defaultLocationAndInfo",
   "formatName":"defaultLocationAndInfo",

   "tags":[
      {

         "tagId":"ca0a00025004",
         "tagName":null,
         "lastPacketTS":1657007497625,
         "color":"#FF0000",
         "tagGroupName":null,
         "locationType":"position",
         "locationMovementStatus":"stationary",
         "locationRadius":0.14,

         "location":[
            63.07,
            7.24,
            1.00
         ],
```

"tags" is the "data root" parameter

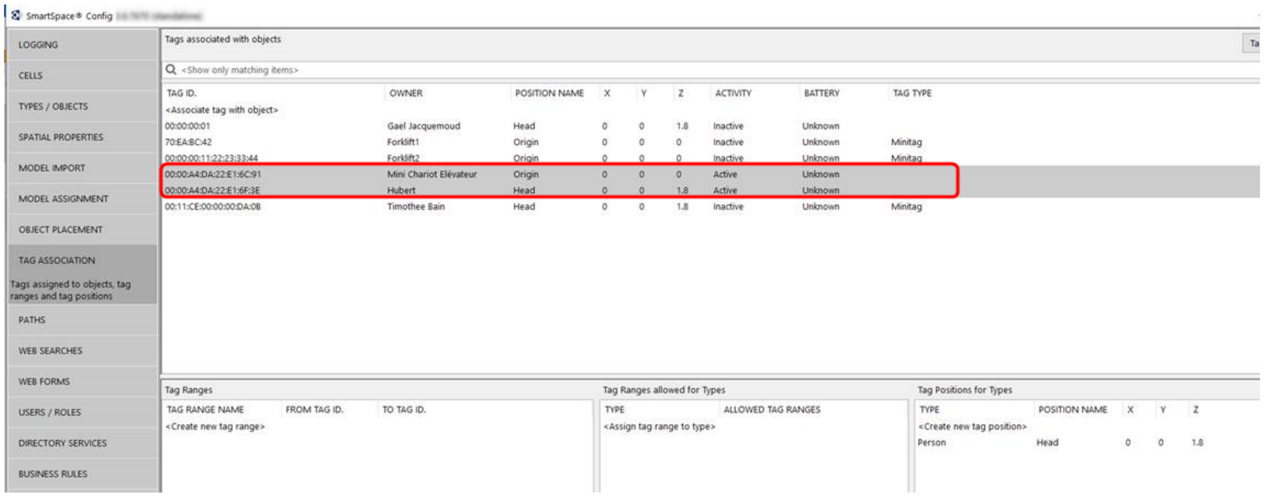
"tagId" is the "id identity"

The "location" co-ordinates correspond to the "x identity", "y identity" and "z identity" parameters

```
         "locationTS":1657007497625,
         "locationCoordSysId":"9e5a0d6c-c006-42fe-97a1-
641615ca7d05",
         "locationCoordSysName":"ecf86399-f850-093a-7410-
aebb6d3b4c16",
         "locationZoneIds":[
            "413d6c5b-a486-4ea5-b078-ef086d0594e5"
         ],
         "locationZoneNames":[
            "RED"
         ],
         "button1State":"notPushed",
         "button1StateTS":1657007477601,
         "button1LastPressTS":null,
         "batteryAlarm":"ok",
         "batteryAlarmTS":1657007477601,
         "rssi":35,
         "rssiLocatorCount":2
      },
      {
         "tagId":"ca2000006062",
         "tagName":null,
         "lastPacketTS":1657007506392,
         "color":"#FF0000",
         "tagGroupName":null,
         "locationType":"presence",
         "locationMovementStatus":"stationary",
         "locationRadius":null,
         "location":null,
         "locationTS":1657007506392,
         "locationCoordSysId":"9e5a0d6c-c006-42fe-97a1-
641615ca7d05",
         "locationCoordSysName":"ecf86399-f850-093a-7410-
aebb6d3b4c16",
         "locationZoneIds":null,
         "locationZoneNames":null,
         "button1State":"notPushed",
         "button1StateTS":1657007506392,
         "button1LastPressTS":null,
         "batteryAlarm":"ok",
         "batteryAlarmTS":1657007506392,
         "rssi":28,
         "rssiLocatorCount":1
      },
   ]
}
```

With the parameters defined, we need to make the association between Quuppa tags and SmartSpace objects, using a tag ID format with 16 hexadecimal digits. In SmartSpace Config, Open the TAG ASSOCIATION task and double-click **<Associate tag with object>** to enter details of the association.

Below is an example of the tag associations:



SmartSpace® Config

Tags associated with objects

<Show only matching items>

TAG ID.	OWNER	POSITION NAME	X	Y	Z	ACTIVITY	BATTERY	TAG TYPE
<Associate tag with object>								
00:00:00:01	Gael Jacquemoud	Head	0	0	1.8	Inactive	Unknown	
70:EA:BC:42	Forklift1	Origin	0	0	0	Inactive	Unknown	Minitag
00:00:00:11:22:23:33:44	Forklift2	Origin	0	0	0	Inactive	Unknown	Minitag
00:00:A4:DA:22:E1:6C:91	Mini Chariot Elévateur	Origin	0	0	0	Active	Unknown	
00:00:A4:DA:22:E1:6F:3E	Hubert	Head	0	0	1.8	Active	Unknown	
00:11:CE:00:00:00:DA:0B	Timothee Bain	Head	0	0	1.8	Inactive	Unknown	Minitag

Tag Ranges

TAG RANGE NAME	FROM TAG ID.	TO TAG ID.
<Create new tag range>		

Tag Ranges allowed for Types

TYPE	ALLOWED TAG RANGES
<Assign tag range to type>	

Tag Positions for Types

TYPE	POSITION NAME	X	Y	Z
<Create new tag position>				
Person	Head	0	0	1.8

Method B

In this method, the Quuppa action will point to a property of the asset (*not* to its associated tag, as done in Method A).

1. After we have defined the Stream and its parameter, we need to create a "Quuppa Tag Id" property for your asset object whose value is unique. In SmartSpace Config, we use the TYPES / OBJECTS task to add a "Quuppa Tag Id" property for each object type that is tracked by a Quuppa tag. Then for each instance of the tracked object we must assign a Quuppa tag ID, using the format used in the *JSON file*.
2. In SmartSpace Config, we use the SERVICE PARAMETERS task to configure the parameters for the "Quuppa_LocationAction". For this method, we need to set the following parameters:
    - "object type" which should be the name of the SmartSpace object type tracked by Quuppa
    - "id property" which should be the name of the unique property we created for the tracked object which is set to the "Quuppa Tag Id".
3. We must then ensure SmartSpace asset objects tracked by Quuppa have a tag associated with them, either manually (with a dummy tag) or using the *Association Action*.