SmartSpace

Location Rules Configuration Guide

For version 3.3.2

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

Telephone: +44 (0)1223 535170. Website: http://www.ubisense.net.

Ubisense Limited

St Andrew's House

St Andrew's Road

Cambridge CB4 1DL

United Kingdom

Tel: +44 (0)1223 535170

WWW: http://www.ubisense.net

Contents

Overview

This document describes the configuration of the features that make up the Location rules component of SmartSpace 3.3. In this release, the features are:

Automatic tag association

This allows a tag detected at a given location to be associated with an externally determined candidate object, subject to conditions that make this association robust in a production environment. This includes the display of the status of an association point using a web browser.

Robust assertion points

An assertion point detects when an object is located at a given place using strong evidence, including distance and speed. This allows the point to be robust to transient process errors in production, such as carrying assets close to a gate location.

Parking

Parking is an extension of robust assertion points, where the object is snapped into the given bay while it satisfies the strong evidence.

Driven objects

This supports locating an object based on the location of another *container* object. When the container moves, the contained objects are carried along. Various layout options are supported.

Stale location detection

When an object has an associated tag, and no sighting of that tag has been generated for a configured time interval, an assertion is made that the object is "stale". This assertion is retracted as soon as the tag is located again.

Location removal mechanism

The location removal feature is similar to tag removal and object deletion supported via assertions in SmartSpace core. Location removal allows an assertion to be made that causes the current location of the object to be removed, and the assertion will be reset once this has been done.

Audience

This guide should be read by those who will be setting up a SmartSpace system to use the features of Location rules, or those who are maintaining a system that uses Location rules.

Prerequisites

Installation

Location rules is a licensed component of SmartSpace 3.3. See *Installing SmartSpace* on the SmartSpace website at http://www.ubisensesmartspace.com/ for requirements and installation process. You will need a license to Location rules in order to install it.

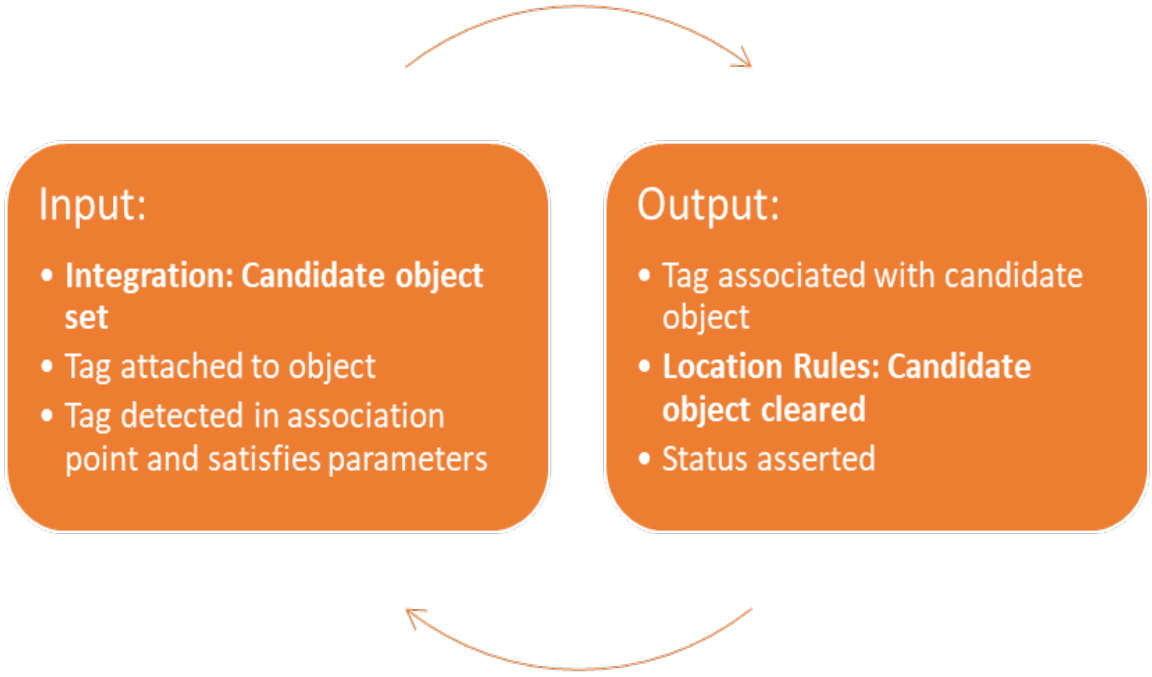
The automatic association status web page is part of the SmartSpace Web. See Installing SmartSpace Web on the SmartSpace website at http://www.ubisensesmartspace.com/ for requirements and installation process.

SmartSpace Config

The configuration of Location Rules features is done in the SmartSpace Config application. Use the Application Manager tool to download this application once the services have been installed.

Automatic Tag Association

The automatic tag association feature operates by assigning a sequence of objects created from an external source to tags detected at an association point, usually as the tag is attached to the object during production. The candidate objects are provided to each association point using an assertion which must be set in time for the tag to be attached, and is then automatically retracted when the association has been made. We will discuss options for implementing this integration process below.



The cycle of operation of an association point

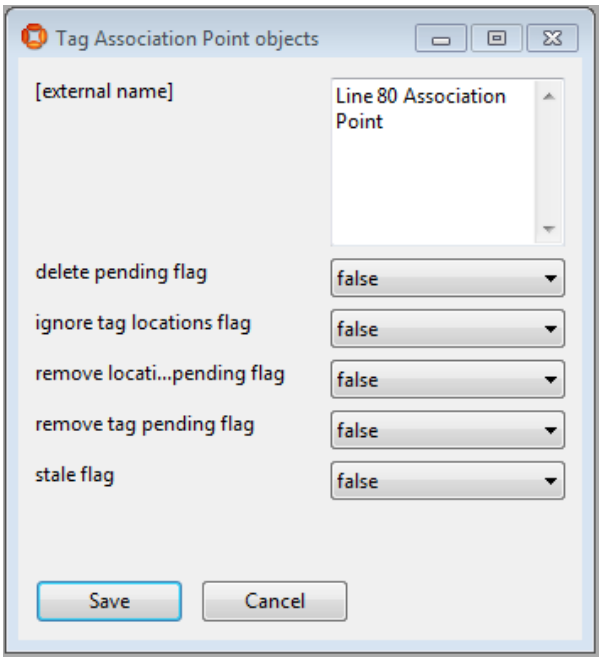
Creating a Tag Association Point

The following steps are used to create and configure a tag association point.

1. If you haven't already done so, set up a representation model for tag association points so you can place them and see where they are placed. See *Importing an object representation* on the SmartSpace website at http://www.ubisensesmartspace.com/ for how to set a representation on a type in SmartSpace Config. Make sure the representation is scaled to a suitable size, and

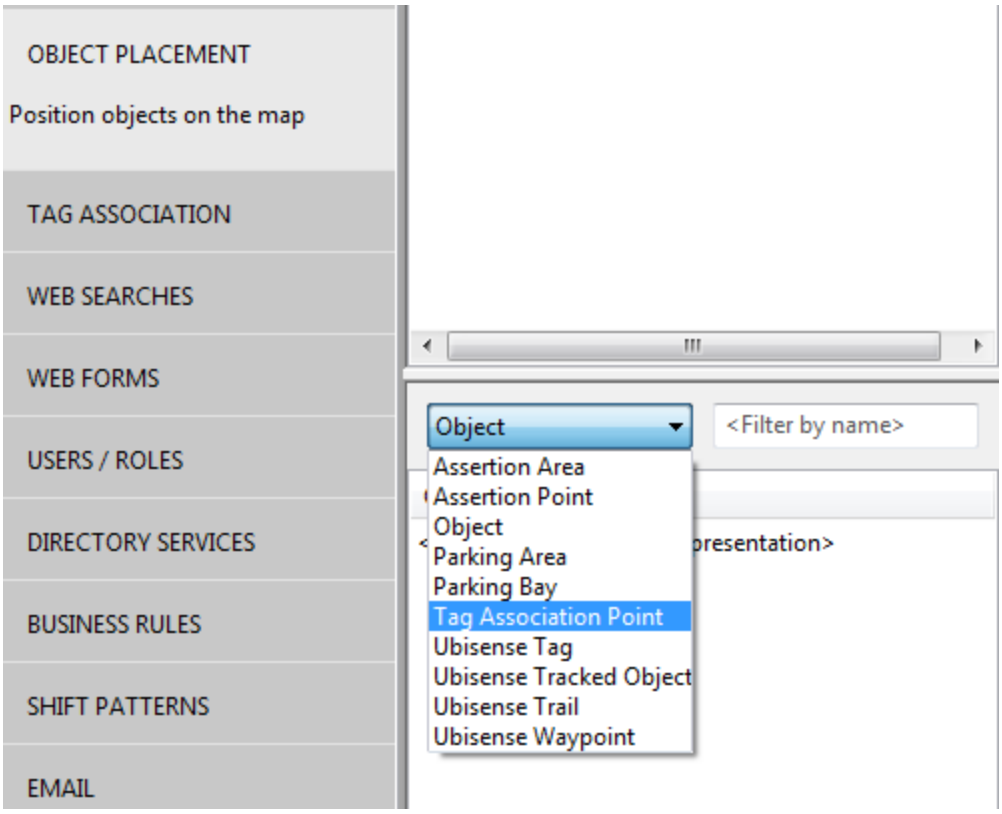
the origin set appropriately.

2. Create an object for the tag association point. Using SmartSpace Config, under **TYPES/OBJECTS** drag out the Tag Association Point type into the right-hand pane. Double-click **<Create new object>** from the Tag Association Point objects dialog. Enter the name of the association point into the name field, and click **Save.**

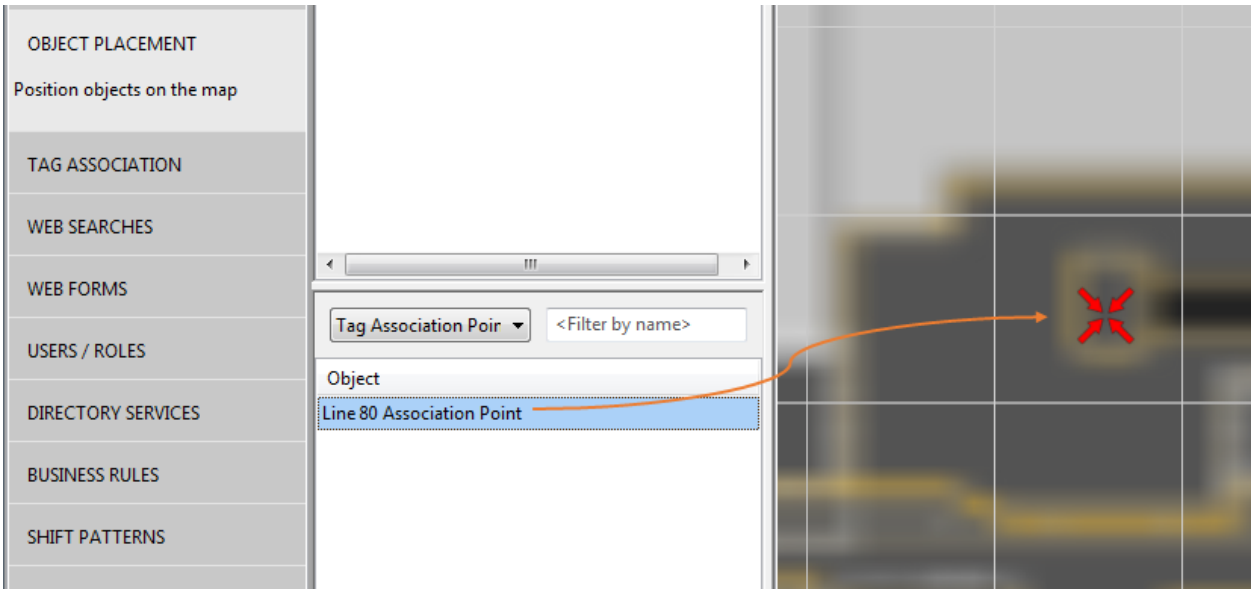


Creating a new tag association point

3. Place the tag association point in the correct location on your site map. Using **OBJECT PLACEMENT**, select the Tag Association Points type in the bottom left pane, and then drag the association point instance onto the correct location in the map.

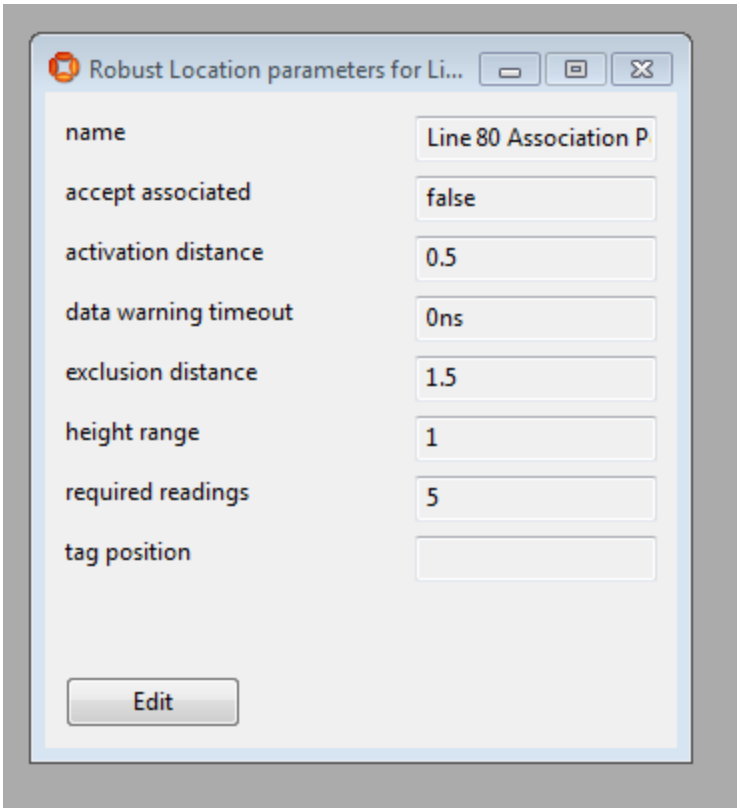


Placing the tag association point: select Tag Association Point type



Placing the tag association point: dragging onto the map

4. Configure the parameters of the tag association point. In **SERVICE PARAMETERS**, select Parking and tag auto-association then Tag Association Point. Drag the new association point into the right-hand pane. The set of parameters of the association point will be displayed. Click Edit to change them.



Note that you can set parameters for all instances of Tag Association Point by dragging the line <All objects of type Tag Association Point> instead of a single instance. This will set values for parameters which are not overridden for a specific object instance or sub-type.

Parameters for Tag Association Points

The following static parameters can be set for tag association points. They define configuration that is not frequently changed during operation. For dynamic configuration and operational data, see the assertions below.

accept associated

If false, the tag association point will ignore any tags that have already been associated with another object. If true, an associated tags will be considered for association provided it satisfies all the other constraints.

activation distance

The detected tag must be within this distance of the association point in order to be considered for association.

data warning timeout

How long to wait for a candidate object to be set for the association point, before the web page will display a warning. Once a tag is detected at the association point, the status page will wait this long for a candidate object, before it highlights the error. If this is set to 0 (the default) the web page will not display a warning after a timeout.

exclusion distance

If other tags are located within this distance of the tag association point, they will block association. Specifically, each time a different tag is seen within the exclusion distance, the tag reading count for the current tag will be reset to zero. See required readings below.

height range

The tag must be located within this vertical distance of the tag association point in order to be considered for association. Tags outside this range will be ignored.

required readings

In order to make association more robust to stray tag or tag sighting errors, the association point will wait until it has seen the tag this many times within the activation distance and height range before associating it to the candidate object.

tag position

The tag will be placed at a given attachment point on the candidate object. This parameter defines the offset of the tag from the origin of the object. It must be set to a tag position defined in the TAG ASSOCIATION workspace of SmartSpace Config. If a different tag position is used for different objects (for example different models of car) then this can be supported using the property "sets tag position" as described below.

Assertions for Tag Association Points

The operational control of tag association points is performed using assertions. These will typically be set using some integration code, or via the Business Rules component if it has been

licensed. For testing and development they can be set using **TYPES/OBJECTS** in SmartSpace Config.

'Tag Association Point' has candidate 'Object'

This assertion should be made for each candidate object in turn. It is an essential input to the association point. The association will not happened until there is both a candidate object and a tag that has satisfied the parameters of the association point. Once a tag has been associated, the candidate will be cleared, indicating that the association point is ready for the next candidate to be set.

'Tag Association Point' sets tag position 'String'

This assertion can be used if different objects have tags attached at different positions. Before setting the candidate object, set the tag position to use for the next association. Unlike the candidate object, this assertion is not cleared when the association has been made, so it is up to the integrator to ensure that at most one assertion has been made for each tag association point.

external system message of <Tag Association Point> is <String>

This assertion can be set to deliver a system message to the status web page for a tag association point. It should be combined with the next assertion.

external system status of <Tag Association Point> is <String>

This assertion can be set to indicate a status of the external system integration on the status web page for a tag association point. Valid strings are (without the quotes):

String	Effect
ok	green status
warn	yellow status
error	red status

status of <Tag Association Point> is <String> for object <String> and tag <String> at <Time>

This is generated as output of the association point, and indicates when a successful association has been made, or the cause of a failure to associate. It is displayed in the status web page.

current tag id of <Tag Association Point> is <String>

The currently detected tag id is set as output during operation of the association point. This is cleared once an association succeeds. It is displayed in the status web page.

Support for Multiple Tag Positions

Different tag positions for different object types can be supported:

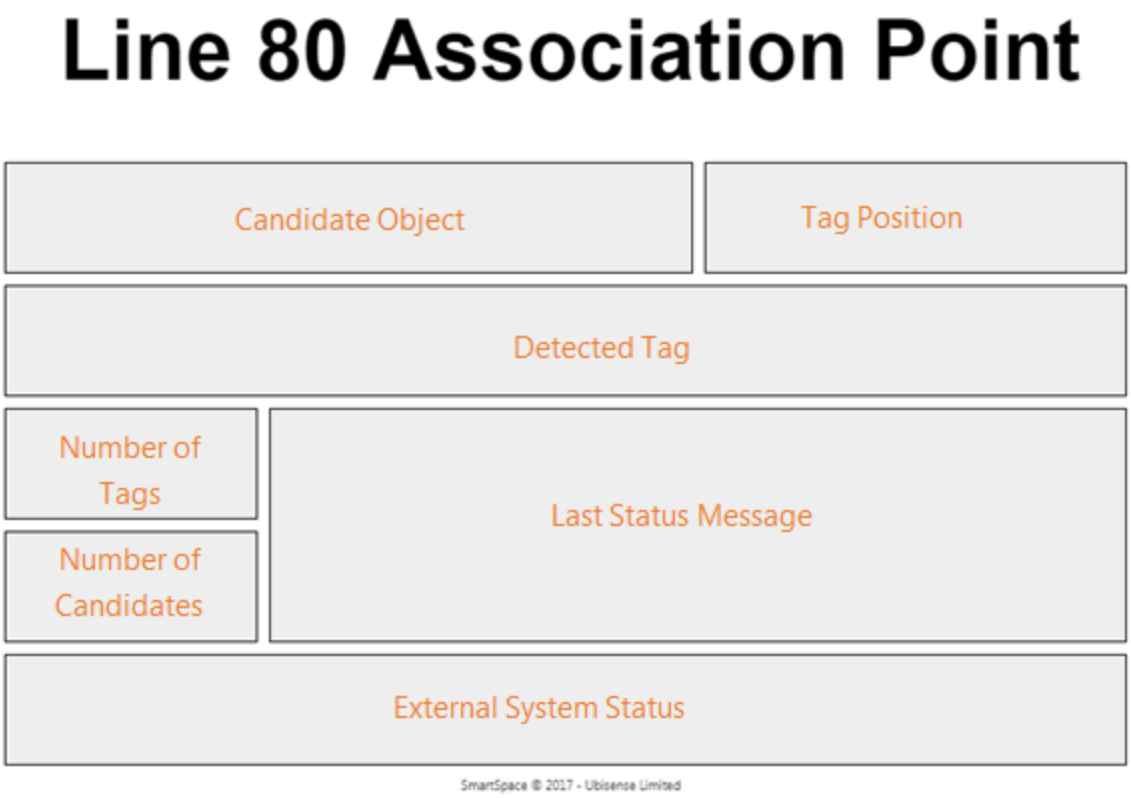
- by using the same name for the tag position, but defining the tag position differently for each candidate object type; or
- by defining external integration logic or rules to set the "'Tag Association Point' sets tag position 'String'" assertion before the candidate object is set.

For example, if Business Rules was licensed, the product could have a "model" property, and the model could have a "tag position" property, and this position could be set for the association point before the candidate object is set. Similar behavior could be achieved using an integration API.

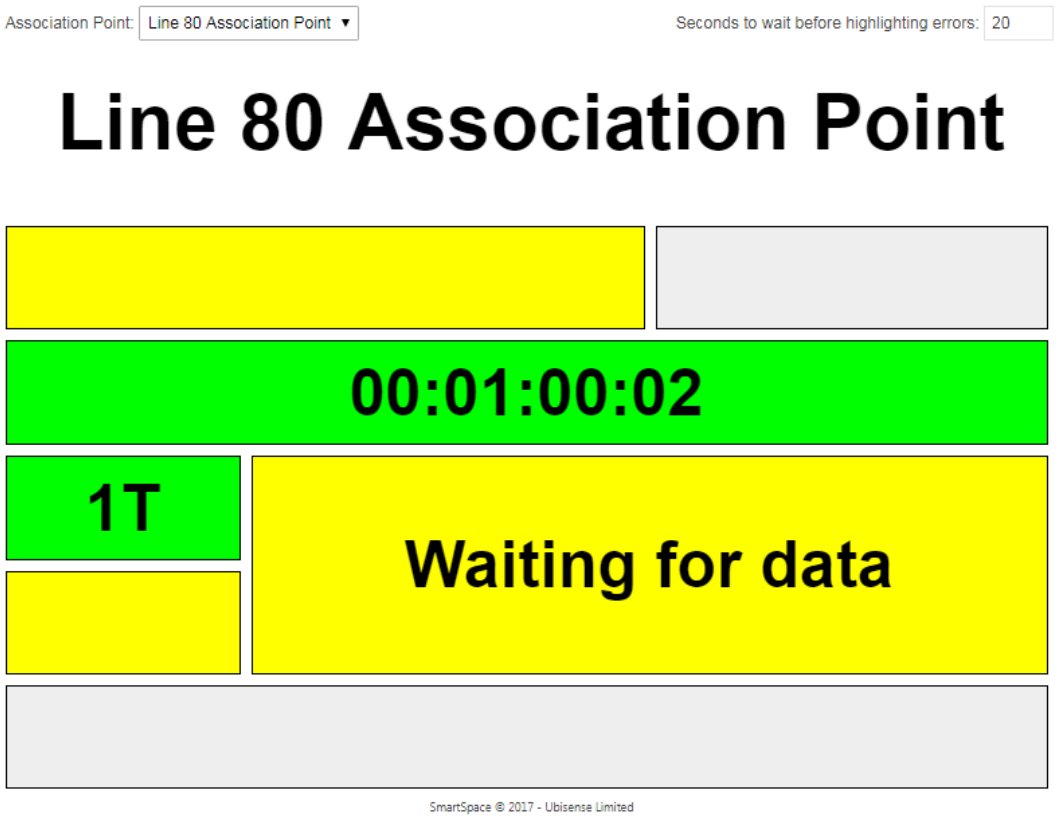
Web Site Operation

The automatic association web page is reached by clicking on the Association menu item. It allows the association point to be selected from a dropdown, or by URL anchor. The first association point will be selected by default.

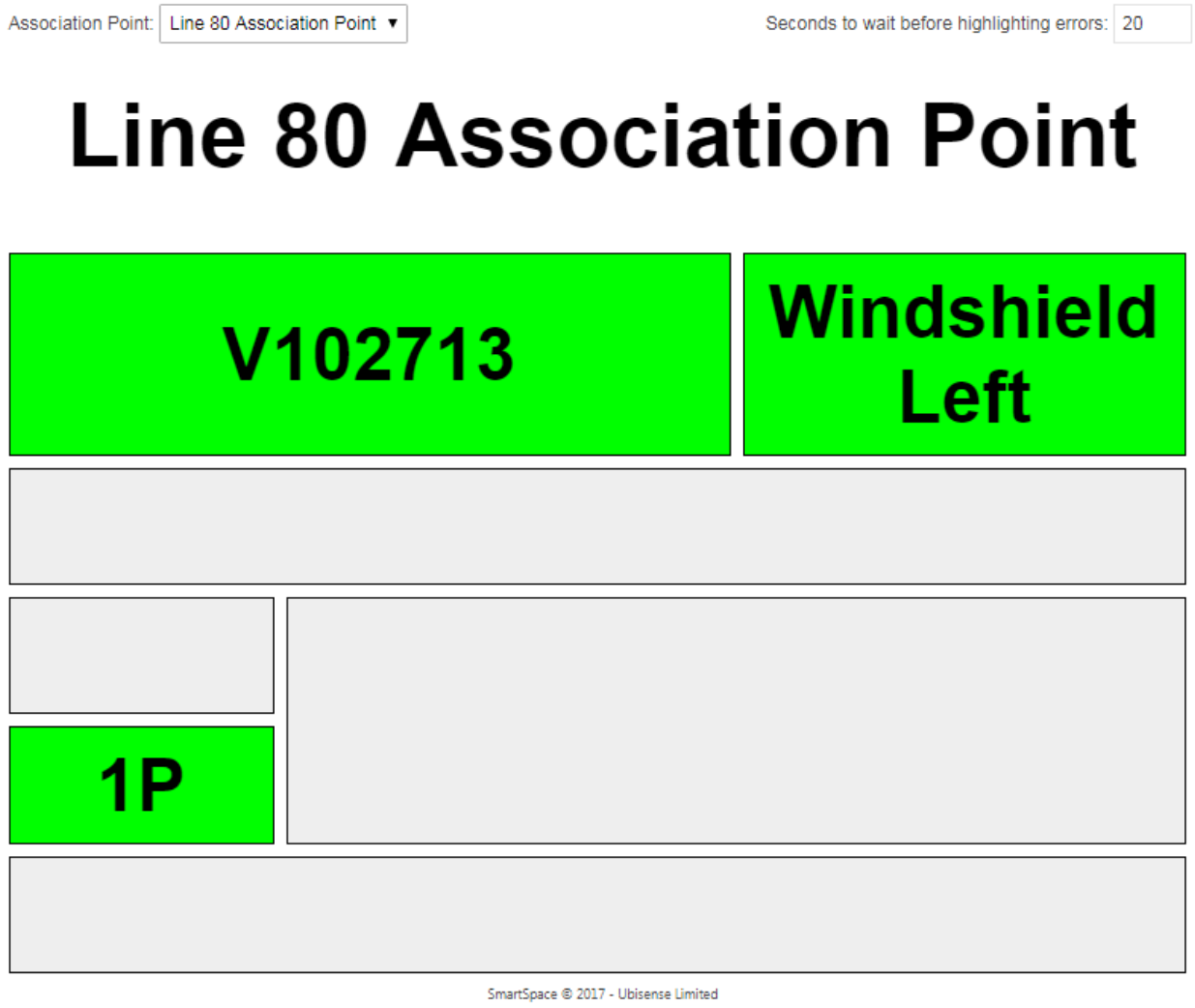
- The web site shows all the relevant state about the association point:
- the candidate object and the position at which the tag should be attached
- the tag id detected
- the most recent status of association, including reasons for not associating
- any external system message or status



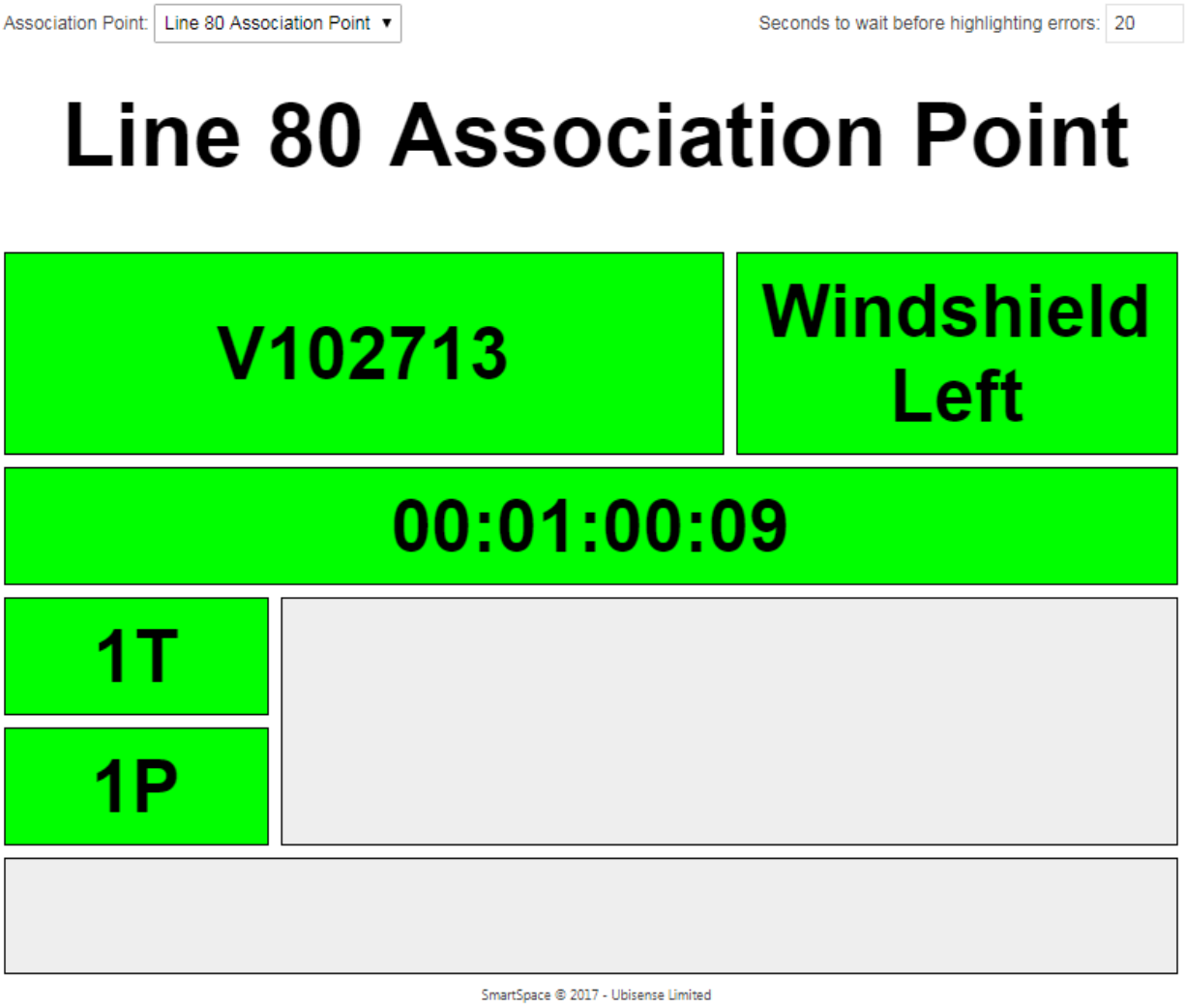
The parts of the association status web page



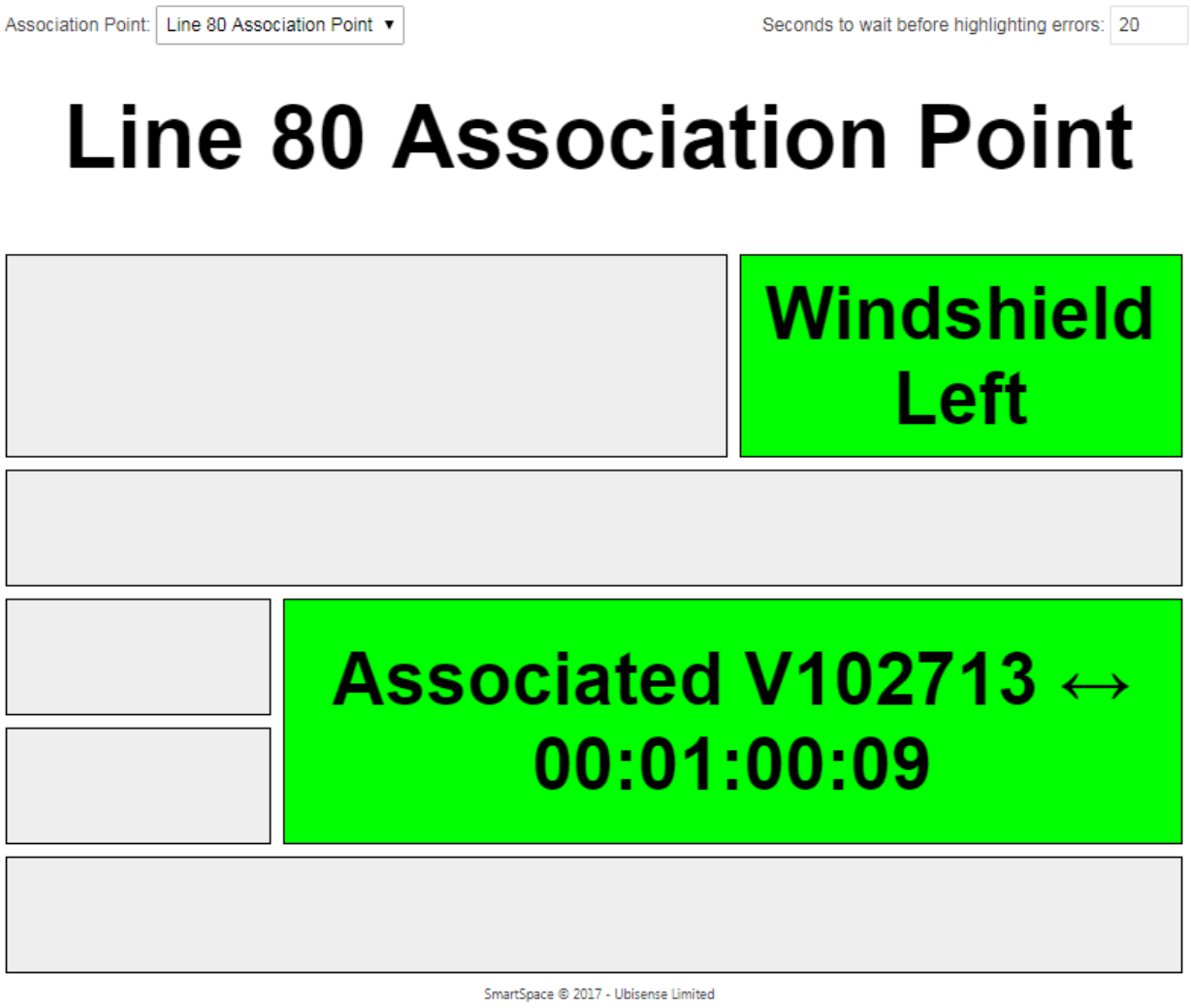
The association status web site showing a tag detected in the station with the required number of sightings, but no candidate object or position.



An association point for which the MES integration has set the next candidate, but no tag has yet been detected in the association point.



The association point now has a detected tag, and when the number of sightings has been satisfied the tag will be associated.



The association station has successfully associated the tag, and the message is displayed.

Association Status Messages

The following status messages can be displayed:

Association Status	Condition
multiple tags in exclusion zone	other tags have been seen within the exclusion distance and height range around the association point
tag battery is not ok	the battery status of the current tag is not OK
object already associated	if "accept associated" is not set, the candidate must not be associated already
tag already associated	if "accept associated" is not set, the tag must not be associated already
no tag position	a tag position has not been defined in either the assertion or configuration parameter for this association point
tag position not defined	the tag position set for the association point has not been defined for the candidate object type
waiting for data	a tag is ready to be associated, but no candidate has yet been set for the association point
associated	the association was successful

Data Warning Timeout

If "data warning timeout" is not 0, the web page displays the missing fields in red after the tag has been detected for the given time but no candidate has been set.

If the user logged on to the web site is a member of the role System.Operator, then the page also shows a form to set and save the "data warning timeout" parameter.

URL to a Specific Association Point

To create a shareable URL for a specific tag association point, append the name of the association point as the query string at the end of the URL. For example, if the default status page URL is:

```
http://smartspace.int/SmartSpace/AA/Association/Public
```

then a URL that goes directly to an association point “AP ITL” will be:

```
http://smartspace.int/SmartSpace/AA/Association/Public?AP%20ITL
```

Note that any special characters in the association point must be URL encoded, as has been done for the space in the name above.

Robust Assertion Points

Assertion points report when an object is at a given place, using strong evidence, including distance from the point and the speed at which the object is moving. This evidence allows the assertion point to be robust to transient errors in production, such as assets moving past a process step or gate location but not actually stopping there. The output of assertion points is the assertion that an object is located at the point, which can then be used by integration code, or the Business Rules if licensed, to trigger other actions or track process.

> Robust Assertion Points are closely related to Parking: the difference between them is that Robust Assertion Points provide the ability to assert an object is located at a point, whilst Parking provides the additional capability of snapping an object into a given bay. If you need to snap objects to a location, you must use the *Parking Area* and *Parking Types* described in Parking.

There are two object types introduced by Robust Assertion Points: *Assertion Area* and *Assertion Point*. The assertion area has two main functions:

1. It groups together assertion points with the same configuration parameters; and
2. It controls a set of objects, which can be located at one of its assertion points.

In a normal configuration, spatial containment is used to define both the points within each area, and the objects controlled by the area.

> Assertion points only work for objects that have associated tags, because it is the tag location events that are processed to decide when the object is located at an assertion point.

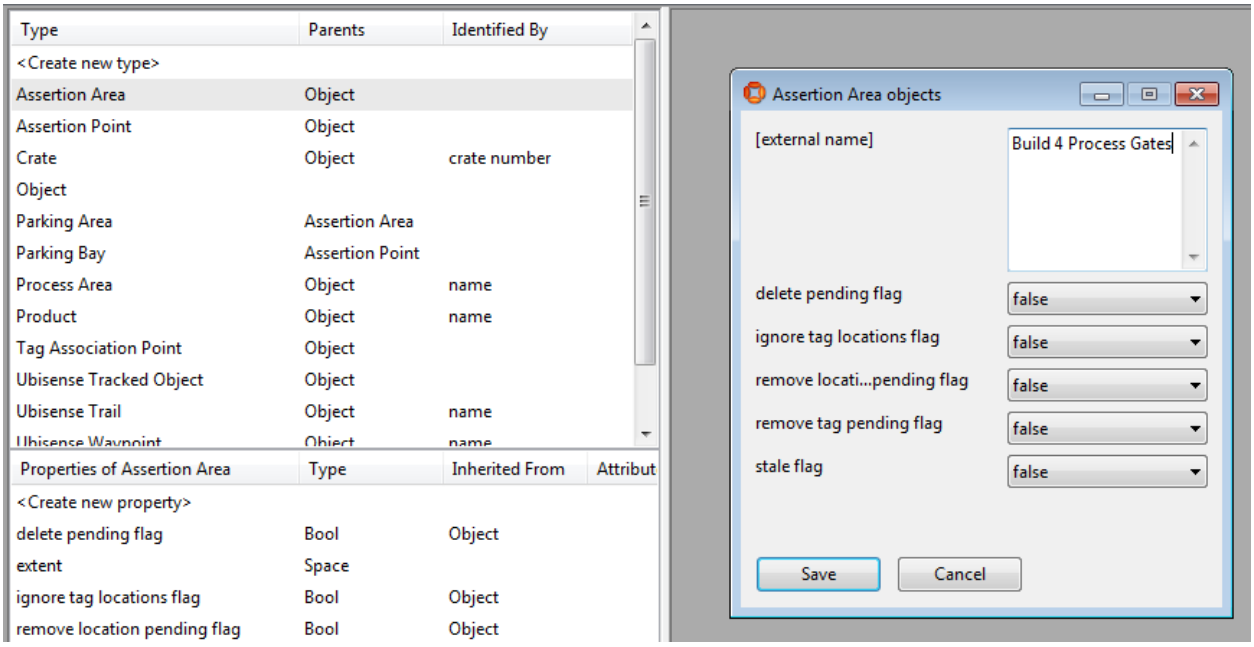
Configuring Assertion Points

This section will walk through creating and configuring robust assertion points. This guide assumes that you have already created a type for the assets to be tracked, and have some instances associated with tags. See Creating a custom type and Tag association on the SmartSpace website at http://www.ubisensesmartspace.com/ for how to do this. We assume the asset type is called *Product*.

1. If you haven't already done so, set up a representation model for Assertion Point so you can place them and see where they are placed. See Importing an object representation

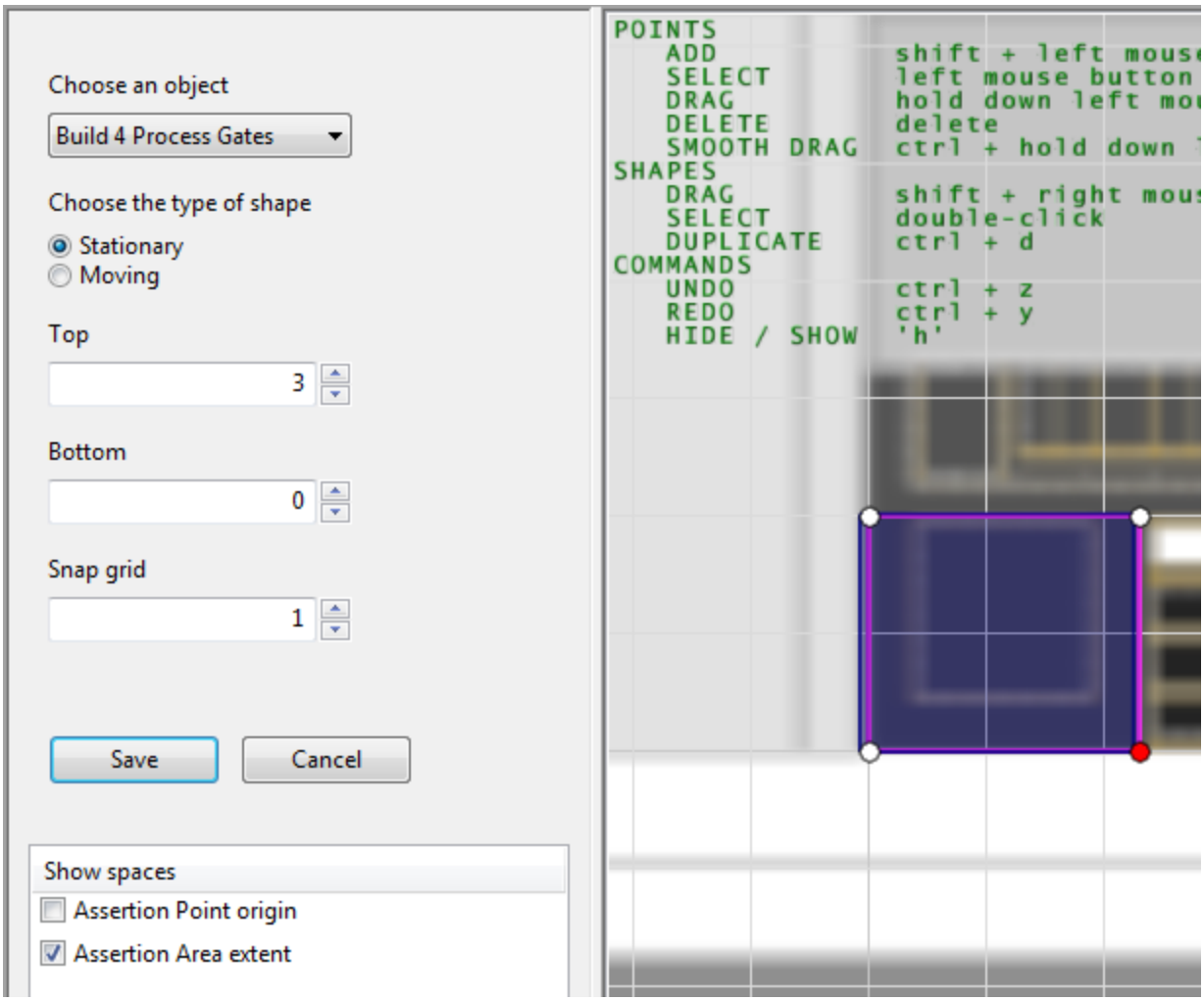
on the SmartSpace website at http://www.ubisensesmartspace.com/ for how to set a representation on a type in SmartSpace Config. Make sure the representation is scaled to a suitable size, and the origin set appropriately.

2. Create an object for the assertion area. Using **SmartSpace Config**, under **TYPES/OBJECTS** drag out the *Assertion Area* type into the right-hand window. Double-click **<Create new object>** from the **Assertion Area objects** window. Enter the name of the assertion area into the name field, and click Save.

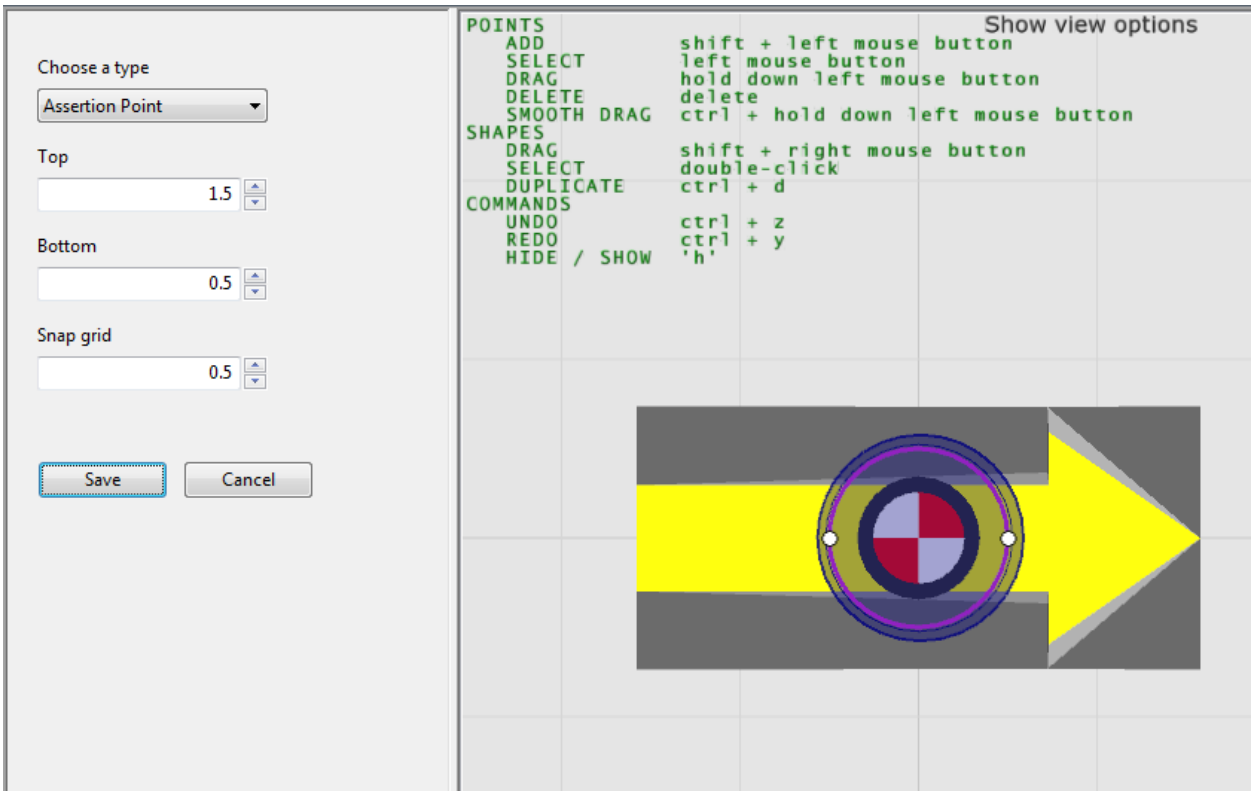


3. Similarly create an instance of type *Assertion Point* for each assertion point you want in the assertion area. In our example, we will create assertion points called "B4 Gate 1" and "B4 Gate 2". Drag the *Assertion Point* type into the right hand pane, double click **<Create new object>** and then enter the two names on separate lines of the name box. Click Save.

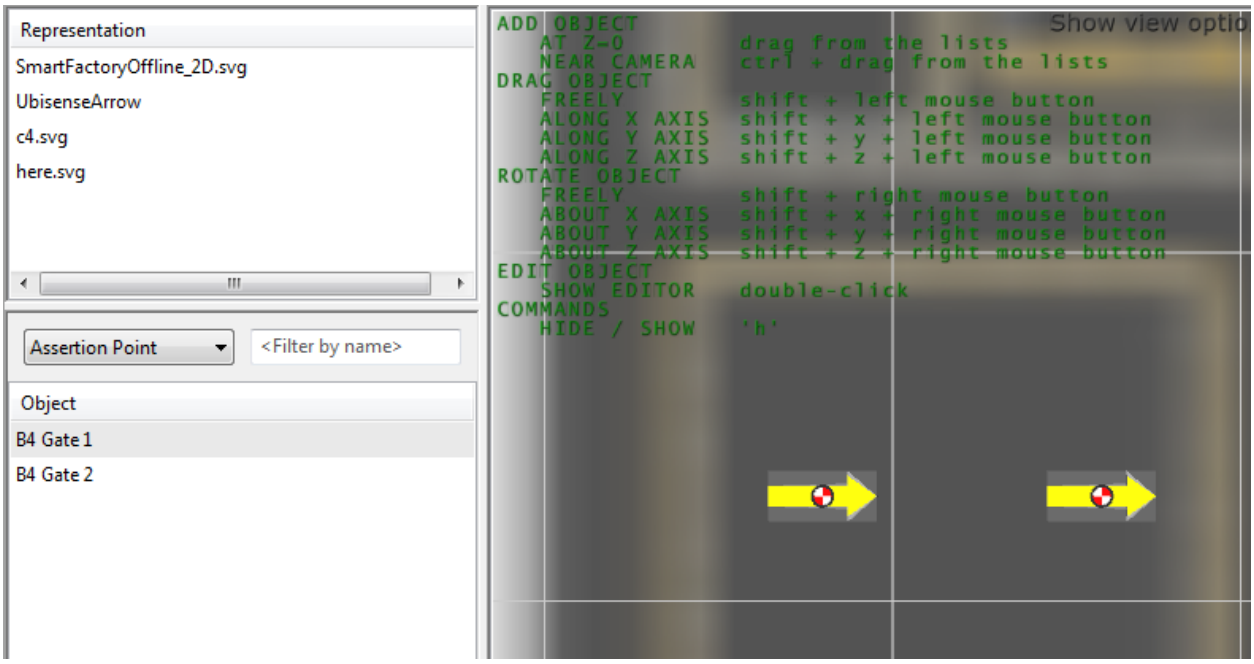
4. Now set up a spatial extent for the assertion area, covering the region containing all the assertion points that will be in this area. In **SPATIAL PROPERTIES** select the *Assertion Area* type, and then the "extent" spatial property. Under **Specific Spaces** double click **<Create new specific space>** and select the area you are creating in **Choose an object**, then use shift + left mouse to place points for the boundary of the extent. Click Save.



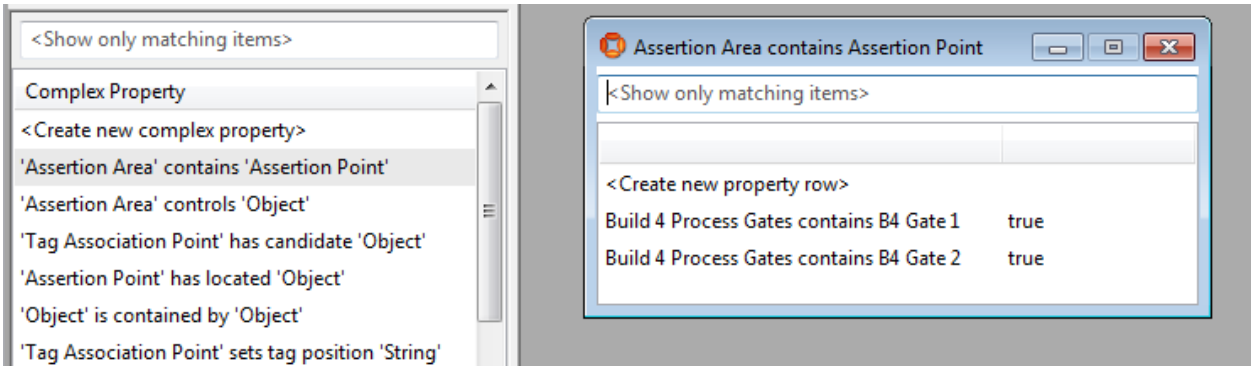
5. Also create a default space for the origin of the association point type – in this case a small space with a limited height, so that it can be contained within the extent of the association area. Select the *Assertion Point* type, then the "origin" spatial property. Under **Default Spaces**, double click **<Create a new default space>**, select Assertion Point type again, and hit the space bar to reset the view. Now create a 1m diameter cylinder around the point with vertical extent from top 1.0 m to bottom 0.5 m. Click Save.



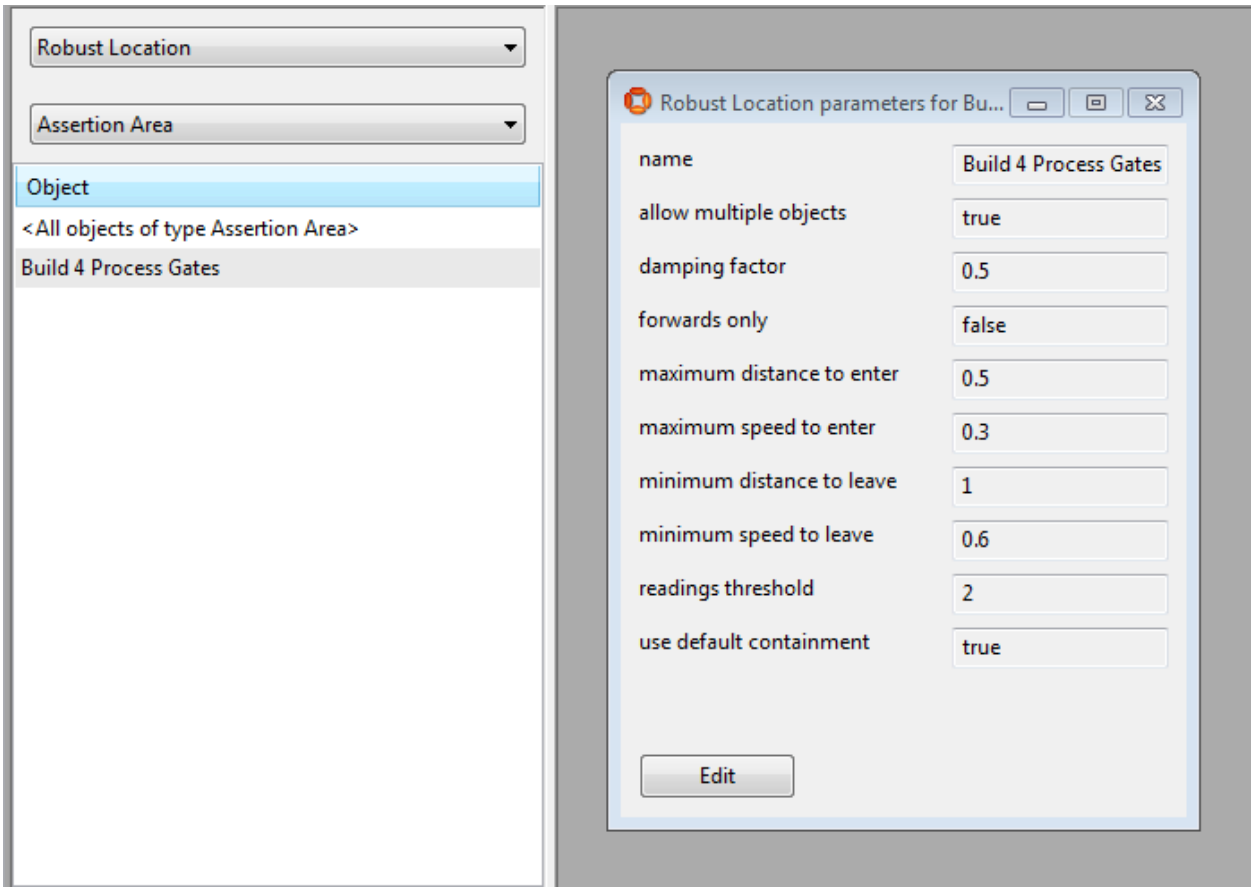
6. Place each of the assertion points that were created. In the **OBJECT PLACEMENT** task, select the *Assertion Point* type from the filter dropdown, and then drag each assertion point instance into the location on the map. If Assertion Point does not appear as a choice, you have not set a representation model – see step 1.



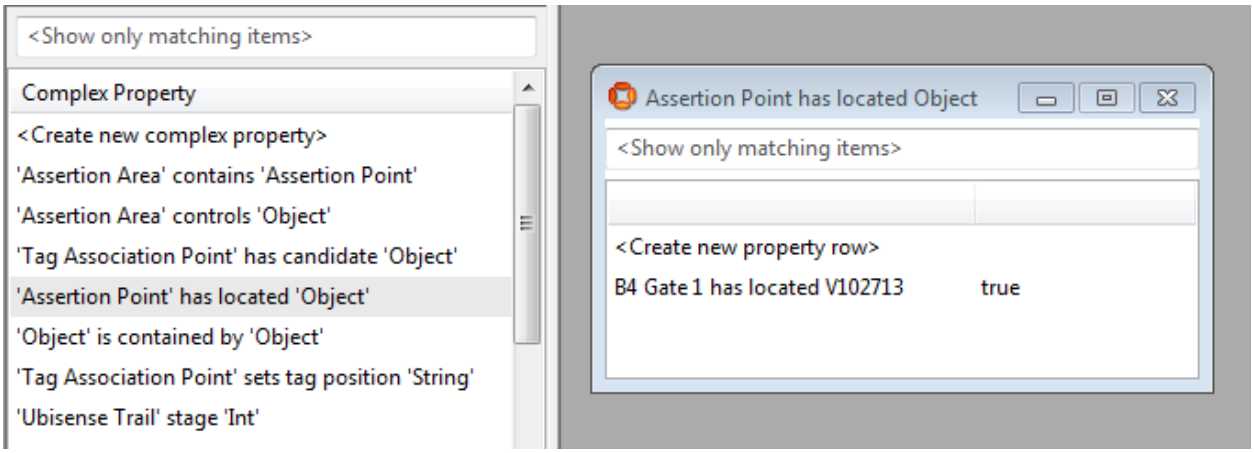
7. At this point, the spatial properties you have just set up should automatically populate the Assertion Area contains Assertion Point property. To check this, go to **TYPES/OBJECTS** and in the Complex Property list find 'Assertion Area' contains 'Assertion Point'. Drag this into the right hand pane. There should be two rows.



8. Configure the parameters of the assertion area. Go to the SERVICE PARAMETERS task, select the Parking and tag auto-association configuration, and the Assertion Area type. Drag the area you created into the right hand pane. You can set up the evidence required for the area. The parameter meanings are described below. For the moment we will leave them with their default values.



9. Now we define a spatial property of the Product type to automatically make them controlled by the assertion area when they enter its extent. In **TYPES/OBJECTS**, select the Product type, and under **Properties of Product**, click **<Create new property>**. Create a property called "origin" whose value has type Space. Now go to the **SPATIAL PROPERTIES** task, and follow a similar process to step 5 to create a default space for this origin property. Then in **Monitor spatial relations**, double click **<Add new request>** and select Container *extent of 'Assertion Area'* and Contained *origin of 'Product'*. Click Save.
10. Now move one of your products so that it is at the location of one of the association points. The tag should get sightings at the corresponding offset, and once the evidence is satisfied, the assertion should be made that the assertion point has located the object. In **TYPES/OBJECTS**, under the **Complex Property** list, drag the property *'Assertion Point' has located 'Object'* into the right hand pane. There should be a row for the detected object.

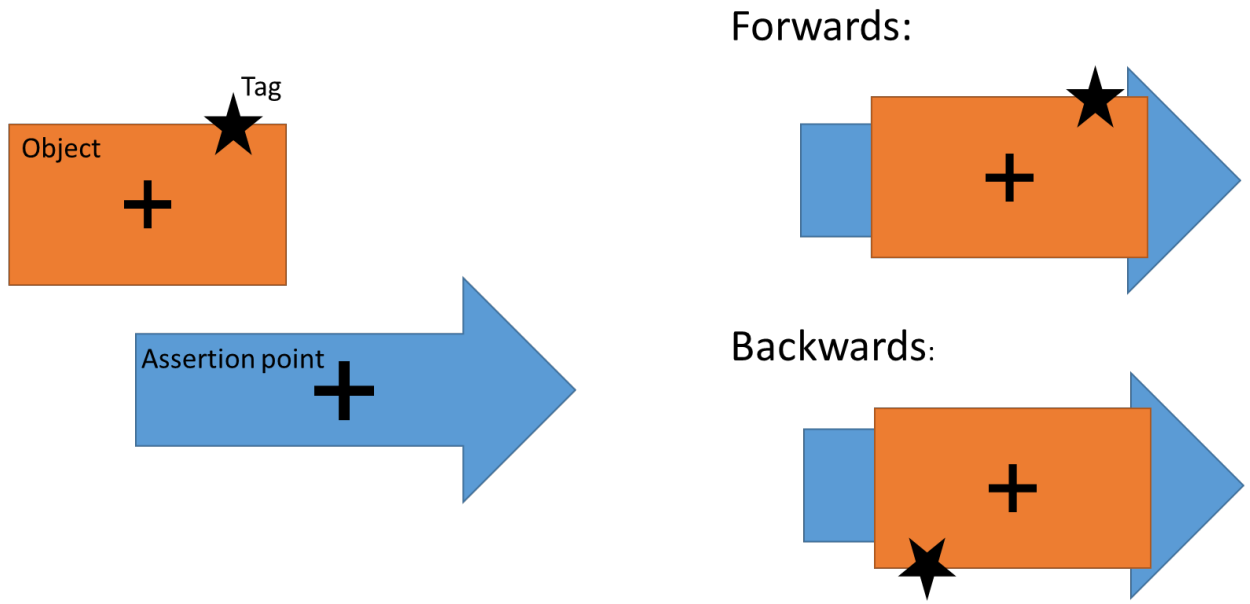


Parameters for Assertion Areas

The following parameters are defined for assertion areas, and apply to the assertion points they contain.

forwards only

When set to false (the default), the object will be detected whether it is at the assertion point forwards or backwards. When true, the object must be pointing only in the same direction as the assertion point. The two cases, forwards and backwards, are illustrated below.



The two possible orientations of the object when located at an assertion point, and their relationship to the tag position.

readings threshold

The minimum number of tag events required to have been seen for an object that is controlled by the area, before it is located at an assertion point.

damping factor

The sightings of the tag are passed through a low-pass filter, before computing the distance of the object from the assertion point, and the speed of the object (see the parameters below). This filter effectively smooths out the tag position by blending each sighting with the previous position. The damping factor controls the strength of this smoothing, and should be in the range 0 <= damping factor < 1. At 0, there is no damping, so the raw tag sightings are used. At 0.5, the current position of the tag is blended equally with the previous position of the tag. A value of 1 should not be used, as the current position of the tag would be entirely discarded.

If the tag sightings are noisy, such as locations produced by GPS, then a higher damping factor can be used to smooth the data out before attempting to satisfy the constraints below. The trade-off will be how many sightings it takes before "located at" is asserted.

maximum distance to enter

The damped position of the tag must be below this distance from where it would be if the object was located at an assertion point (forwards or backwards) to consider it as "located at" the point.

maximum speed to enter

The damped speed of the tag must be below this threshold to consider the object as "located at" a point.

minimum distance to leave

The damped position of the tag must be above this distance from where it is would be if the object was located at an assertion point (forwards or backwards) to consider it as no longer "located at" the point.

minimum speed to leave

The damped speed of the tag must be above this threshold to consider it as no longer "located at" a point.

use default containment

If true (the default), then use the default spatial property interactions to populate *the 'Assertion Area' contains 'Assertion Point'* and *'Assertion Area' controls 'Object'* assertions. See the section on Assertions for Assertion Areas.

allow multiple objects

Whether multiple objects can be asserted as located at a single assertion point at the same time. If true, then a single assertion point can have multiple rows at one time in the *'Assertion Point' has located 'Object'* assertion. If false, only a single object can be detected at a time, and a second detection will be ignored.

Assertions for Assertion Areas

<Assertion Area> contains <Assertion Point>

An input assertion, which should contain one row for each assertion area that contains a given assertion point. In most cases this will remain relatively constant in operation, unless the assertion points themselves are mobile. If "use default containment" is true, the contents of this assertion will match the spatial relation "Assertion Area extent contains Assertion Point origin".

<Assertion Area> controls <Object>

An input assertion, which contains the set of objects that can be located at an assertion point within each assertion area. If "use default containment" is true, then contents of this assertion will match all spatial relations "Assertion Area extent contains <any other monitored type and role>".

<Assertion Area> has located <Object>

The output of assertion points, indicating which objects have been located at each point.

Advanced use

While the default behavior when "use default containment" is true covers many useful cases, it is possible that a process requires more sophisticated logic to decide which assertion point is in a given assertion area, and which object can be controlled by a given assertion area. For example, it may be that only objects which a specific other property (such as "ready to ship") are allowed to activate a process gate.

If this is the case, set "use default containment" to false, and then implement your own method of maintaining the two input assertions, for example use a Business Rule that combines spatial containments with other logical predicates.

Details of Assertion Operation

The technical details of how assertion areas operate are as follows.

- For each object controlled by an area, the service maintains a damped estimate of the position and velocity of the object's associated tag.
- In order to be detected at a point contained in the area,
    - has tag must be seen at least *readingsthreshold* times while controlled by the area; and
    - the tag position estimate must be closer than *maximum distancetoenter* from where it would be if the object were at the point, and its speed estimate must be less than *maximum speed to enter*.
- If the area is *forwards only* then the object is only considered for detection in the same direction as the point. Otherwise it is tested for detection in both the direction of the point and also rotated 180 degrees.
- Once an object is detected at a point, the *haslocated* assertion is made, and it remains until either:
    - the damped tag position is more than *minimum distance to leave* from the position when located at the point, and moving faster than the *minimum speed to leave*; or
    - the area no longer "controls" the object.

Once conditions are met the assertion is retracted.

Parking

Parking is an extension of robust assertion points, where the object is snapped into the given bay while it satisfies the strong evidence. Configuring parking is almost exactly the same as for robust assertion points, so you should first read the section on Robust Assertion Points.

Whilst the terminology used is parking-based, this functionality can be applied more generally where the location of objects should be snapped to a set of allowed locations and orientations, such as presenting a cleaned-up view of noisy location data.

There are four major differences between parking and assertion points.

1. The types used are different:

   Assertion Area → *Parking Area*

   Assertion Point → *Parking Bay*

   Parking Area is derived from *Assertion Area*, and *Parking Bay* is derived from *Assertion Point*. This means that they inherit all spatial properties, parameters and assertions, and these inherited configurations have the same function.

2. There are two additional parameters of Parking Area: "default orientation" and "default orientation used". These will be discussed below.

3. The configuration parameter "allow multiple objects" is false for all objects of type Parking Area. You can change this parameter as required, to allow multiple objects to park at each bay if this makes sense, but it usually does not.

4. When the assertion "has located" is made for a parking bay and object, the object is also snapped to the location of the parking bay.

Location Snapping

Parking, as with assertion points, is only applied for objects that have an associated tag, because the tag location, along with the attachment position on the object, is used to decide whether the object is at the parking bay.

While an object is parked at a bay ("has located" has been asserted), each time a tag location is seen for the object, it will be placed in the bay if it is not currently there. The orientation used will be that of the parking bay, possibly rotated by 180 degrees if "forwards only" is false. The tag locations will be used to decide whether the object is still in the bay.

As with assertion points, if the tag satisfies the parameters to leave the parking bay, then the "has located" assertion will be retracted, and the object will move again according to the tag position. If the default orientation parameters are set, and the object is still "controlled" by the parking area, then that orientation will be applied as the tag moves.

Also, as with assertion points, if the "controls" assertion is retracted for a parking area and object, the object will move again according to the tag position.

Parameters for Parking Areas

The parameters of parking areas are the same as assertion areas, but with the following additional parameters.

default orientation

When "default orientation used" is set to true, and an object is "controlled" by a *Parking Area*, but not currently parked at a bay, then the orientation of the object (its "yaw", or rotation about the vertical axis) will be snapped to this angle, in degrees. The tag position offset will be applied assuming that the object orientation is this default orientation.

This is only used if the object is not currently parked in a bay. When parked at a bay, the orientation of the bay will be used to instead, either forwards or backwards if the parking area is configured to allow this ("forwards only" set to false).

default orientation used

Set to true to enable the default orientation within the area as described above.

Multiple Parking Areas

Only one parking area should control a given object at a time. If multiple areas control an object at the same time, the resulting parking behavior is undefined. Normally this means that parking area extents should not overlap.

Technically, the intersection of any two parking area extents should not be able to contain the origin space of the parked objects – spatial hysteresis within the SmartSpace spatial monitoring then prevents the object from being controlled by both areas at the same time. See the section on *Spatial monitoring* for more details.

Layout of Parking Bays

It is worth noting that the driven objects feature, described in the section on Driven Objects can be used to layout large numbers of regularly positioned parking bays, which may simplify the configuration of parking for a large site. Essentially the process is to create a container with the correct layout parameters, and then add each bay to the container in the correct order so that they get placed in a grid that matches the real bay layout. This technique only works where the parking area layout is exactly regular over extended areas of the site.

When dealing with many parking bays it is probably most convenient to prepare a set of commands for the **ubisense_udm_admin** command-line tool, rather than manipulating the "is contained by" relation using **SmartSpace Config**.

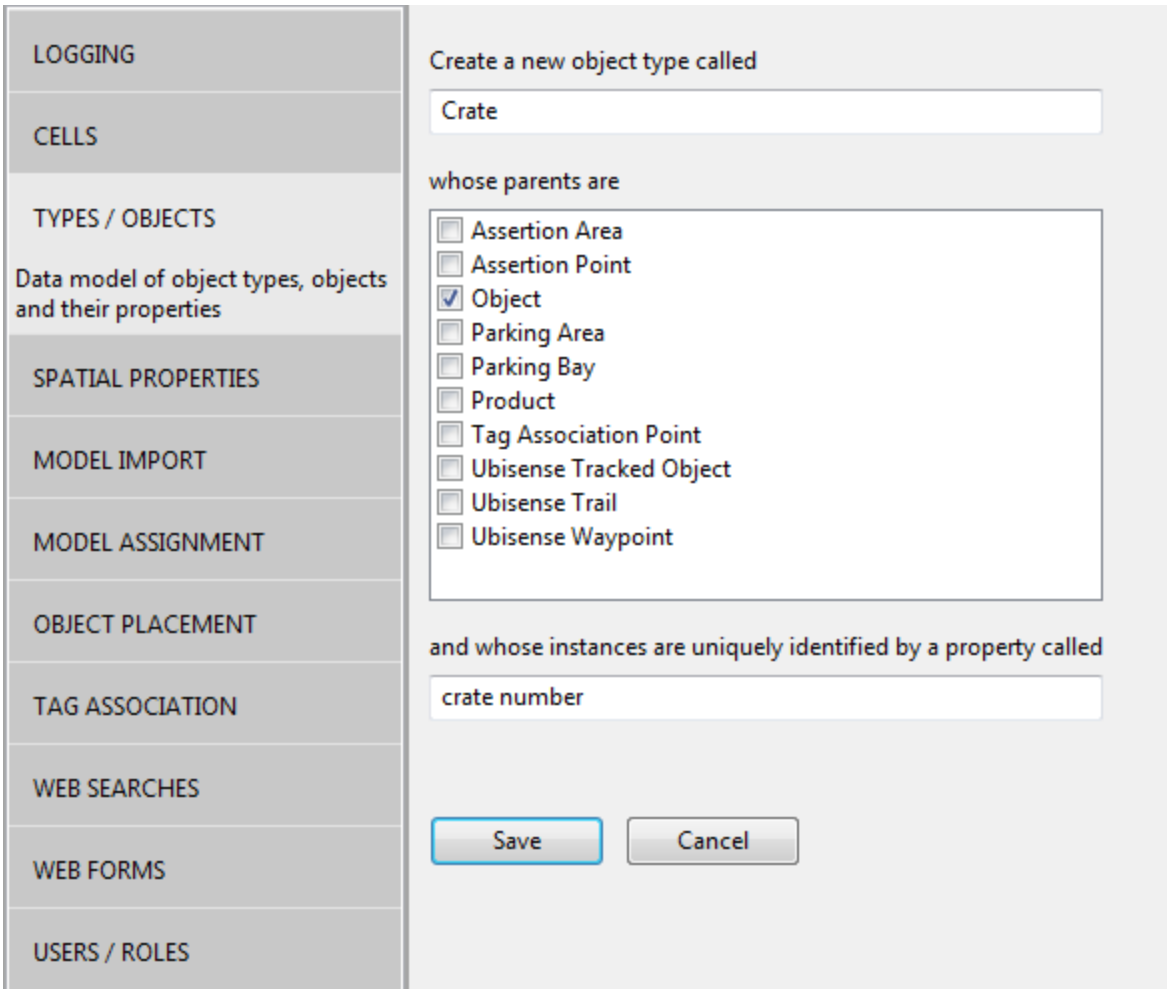
Driven Objects

The Driven Objects feature supports locating an object based on the location of another *container* object. When the container moves, the contained objects are positioned relative to the container. Various layout options can be configured using the parameters of the driven object.

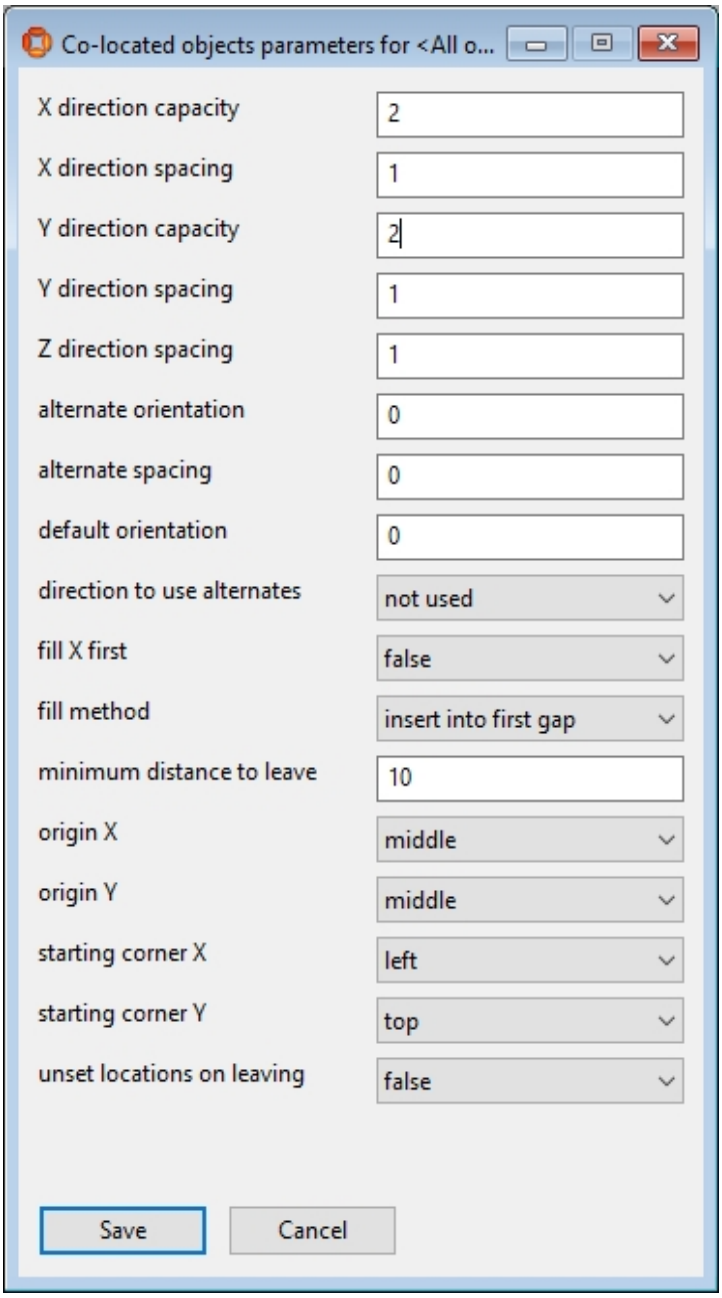
Configuring Driven Objects

We will work through a container configuration, suitable for tracking crates holding work orders.

1. Create the types we will use to represent crates and work orders. In **SmartSpace Config**, select the **TYPES/OBJECTS** task, and click <Create new type>. In the dialog enter the type name "Crate" and tick "Object" from the list of parents. Enter "crate number" as the unique identification property. Click Save.

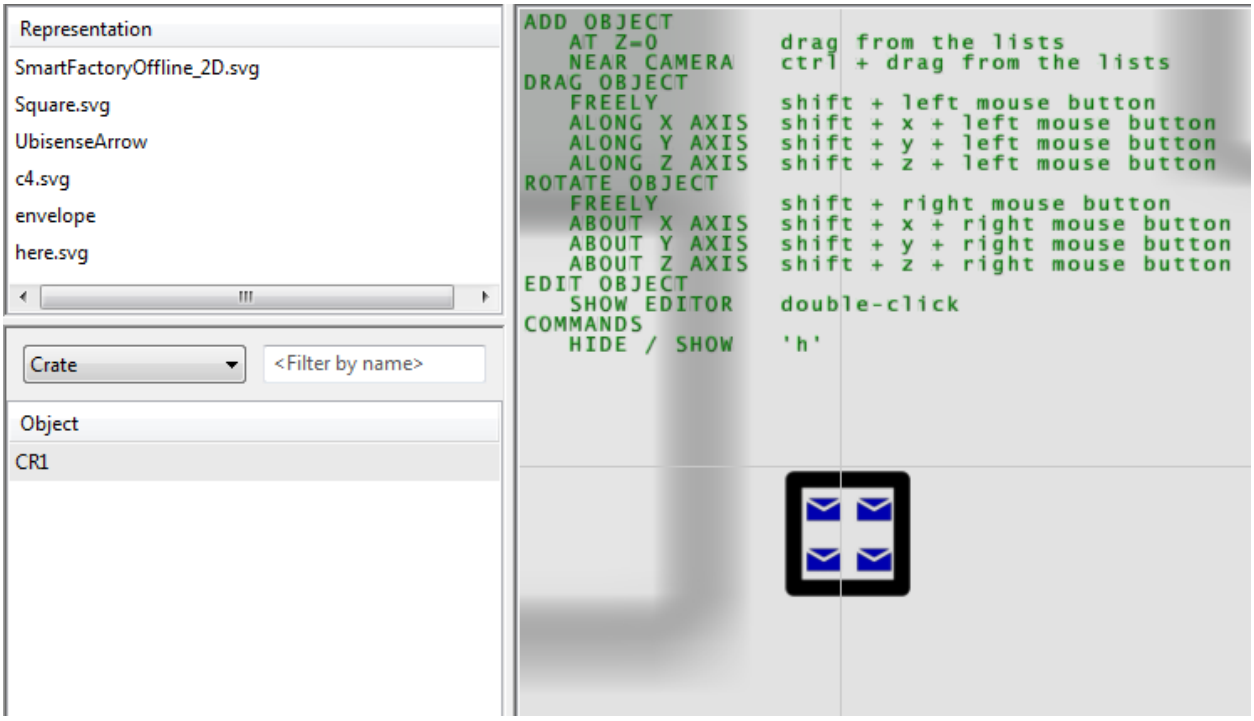


2. Repeat to create type "Work Order", named with the property "id".
3. Import and assign representation models for these two types. See the section on *Importing an object representation* for how to set a representation on a type in SmartSpace Config.
4. Now we will set up the default layout parameters for crates. In the **SERVICE PARAMETERS** task, select the Co-located objects configuration, and then the Crate type. Under the Object list, double click **<All objects of type Crate>**, and then click **Edit** on the parameters window. We will set up the crate to hold four objects in a 2x2 array. Enter the parameters as shown below, then click **Save**.



5. Create an instance of crate and four instances of work order. In the **TYPES/OBJECTS** task, drag type Crate into the right-hand pane, then click **<Create new object>**, and enter CR1 as the crate number. Click **Save**.
6. Repeat using the Work Order type to create four work orders, with ids "A0001", "A0002", "A0003" and "A0004".

7. Set the work orders to be contained by the crate. In **TYPES/OBJECTS**, under the **Complex Properties** list drag the *'Object' is contained by 'Object'* property into the right-hand pane. In the window created, click <**Create new property row**> and pick "A0001" as the first object, and "CR1" as the second. Click **Save**. Repeat for the other work orders.
8. Now place the crate on the map. Use the OBJECT PLACEMENT task, select Crate in the type filter dropdown. Drag the CR1 crate onto the map. The crate and all the work orders will be located. Now use shift+left mouse to drag the crate. The work orders will placed relative to the crate as soon as it is dropped.



Parameters for Driven Objects

The following parameters control the layout behavior of driven objects. In the descriptions below, the x, y and z directions are all relative to the orientation of the container.

X direction capacity

How many slots along the x direction of the container

Y direction capacity

How many slots along the y direction of the container

X direction spacing

The distance between slots along x direction

Y direction spacing

The distance between slots along y direction

Z direction spacing

The distance between slots along z direction (filled after x and y)

unset locations on leaving

When an object is removed from a container, unset its location. Defaults to false, so the object will be left at its last location until it is located by some other means.

default orientation

The orientation in degrees of each slot. Contained object yaw (rotation around z axis) is set to this orientation.

origin X

The horizontal position of the origin relative to the contained slots: “left”, “middle” or “right”

origin Y

The vertical position of the origin relative to first slot: “top”, “middle” or “bottom”

starting corner X

The horizontal position of the first slot to fill: “left” or “right”

starting corner Y

The vertical position of first slot to fill: “top” or “bottom”

fill X first

If true, fill across then down. If false, fill down then across.

fill method

This determines what happens when containments are added and removed. It is set to one of:

- “insert at start”: New containment goes into the first slot, all other slots are shifted to make way. On removal, all slots are shifted to close the gap.
- “insert at end”: New containment goes into the last slot. On removal, all slots are shifted to close the gap.
- “insert into first gap”: New containment goes into the first unoccupied slot. On removal no other contained objects are moved.

alternate orientation

If "direction to use alternatives" is set, this is the orientation in degrees of alternate rows or columns.

alternate spacing

If "direction to use alternatives" is set, this is the spacing of alternate rows or columns

direction to use alternates

In which direction the alternate spacing and orientation are used, one of "x direction" or "y direction" or "not used"

minimum distance to leave

If a contained object has a tag, and that tag is seen more than this distance away from where it would be if the contained position is correct, then the object will leave the container. Set to zero to prevent leaving based on tag locations.

Assertions for Driven Objects

The following assertions are used by the driven objects feature.

<Object> is contained by <Object>

This input assertion indicates that a contained object is driven by a container.

<Object> has <Object> at slot <int>

An output assertion (not visible in **TYPES/OBJECTS**) this is derived from the changes to "is contained by" and the "fill method" parameter of the container. The assertion indicates that a container object has a given contained object at its nth slot.

Details of Driven Objects Operation

If an object with an associated tag is asserted as contained in a container, its location is set based on the container position, and tag locations are ignored unless they are more than a configured distance from the current position generated by the container.

Note that driven object locations are only updated when some location event is received for the location cell.

For example, if no tags are sending sightings to the location cell, no update of driven objects will occur. This means that if you add a row to 'Object' is contained by 'Object' you may not see the contained object get located until another location event occurs. In operational use,

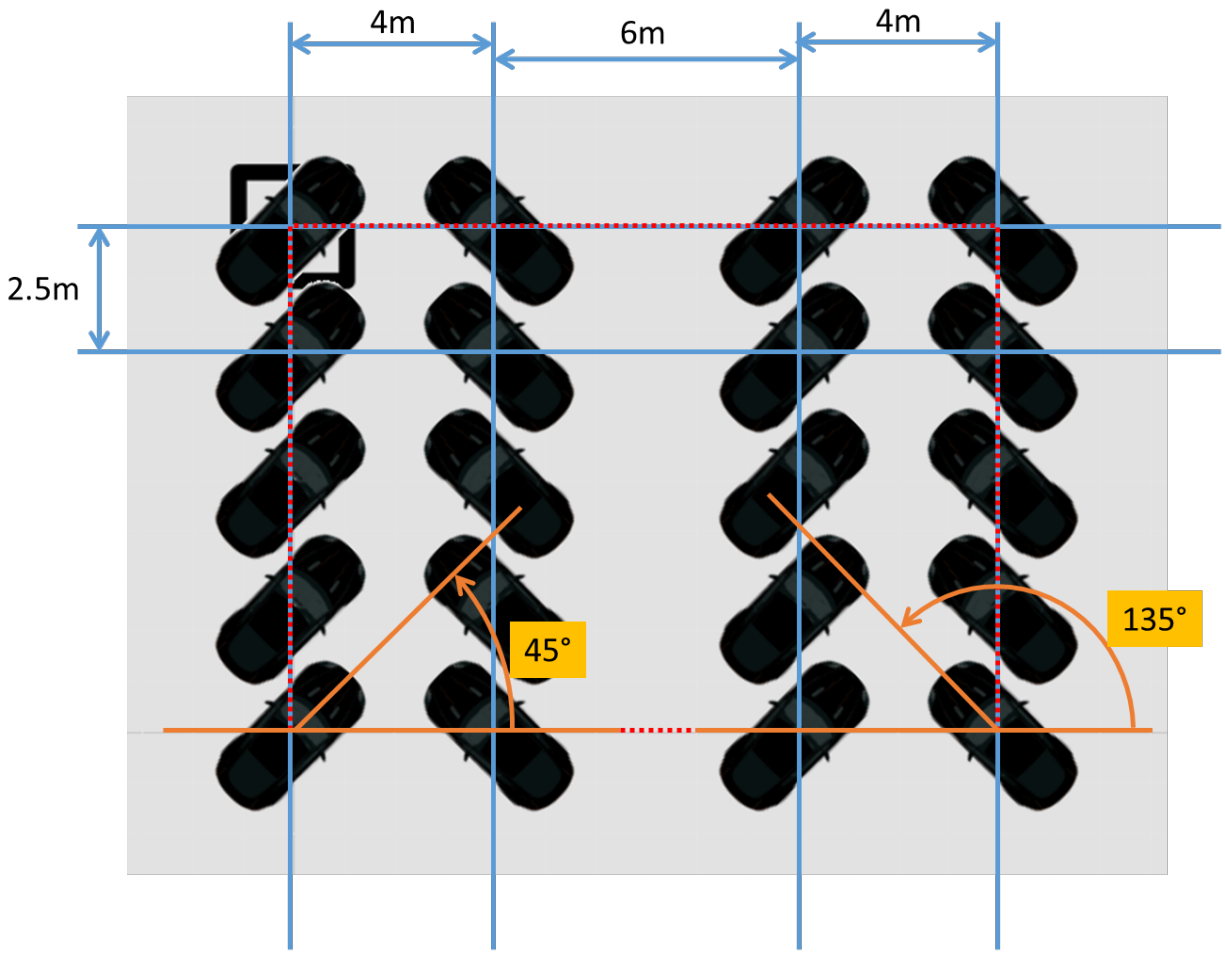
with tag events arriving regularly at each location cell, this is not usually an issue. During demonstrations and development work, this can be confusing behavior until it is understood. In this case, you can simply place the container again to generate all contained locations.

Example Layout Using "Alternates"

The use of the "alternate" parameters can be confusing, so here is an example to illustrating their effects on layout. The parameters used are:

Parameter	Value
X direction capacity	4
X direction spacing	4
Y direction capacity	5
Y direction spacing	2.5
alternate orientation	135
alternate spacing	6
default orientation	45
direction to use alternates	X direction
origin X	left
origin Y	top
starting corner X	left
starting corner Y	top

The resulting layout is shown below, where the black square is the container (with origin in its center), and the black cars are the contained objects (with origin at their rear-view mirror).



The offset of the contained objects from the origin of the container requires some careful consideration. Conceptually the layout generates a set of slot locations which are contained by the box shown as red dotted line. This is the box that is aligned, as configured by the origin parameters, with the origin of the container.

So in this case, because we have specified “left” and “top” for the origin, the top left slot location is aligned with the origin of the container. If we had specified “right” and “bottom” then the bottom right slot location would be aligned with the container origin. If we specified “middle” and “middle”, then the center of that red box would be on top of the container origin.

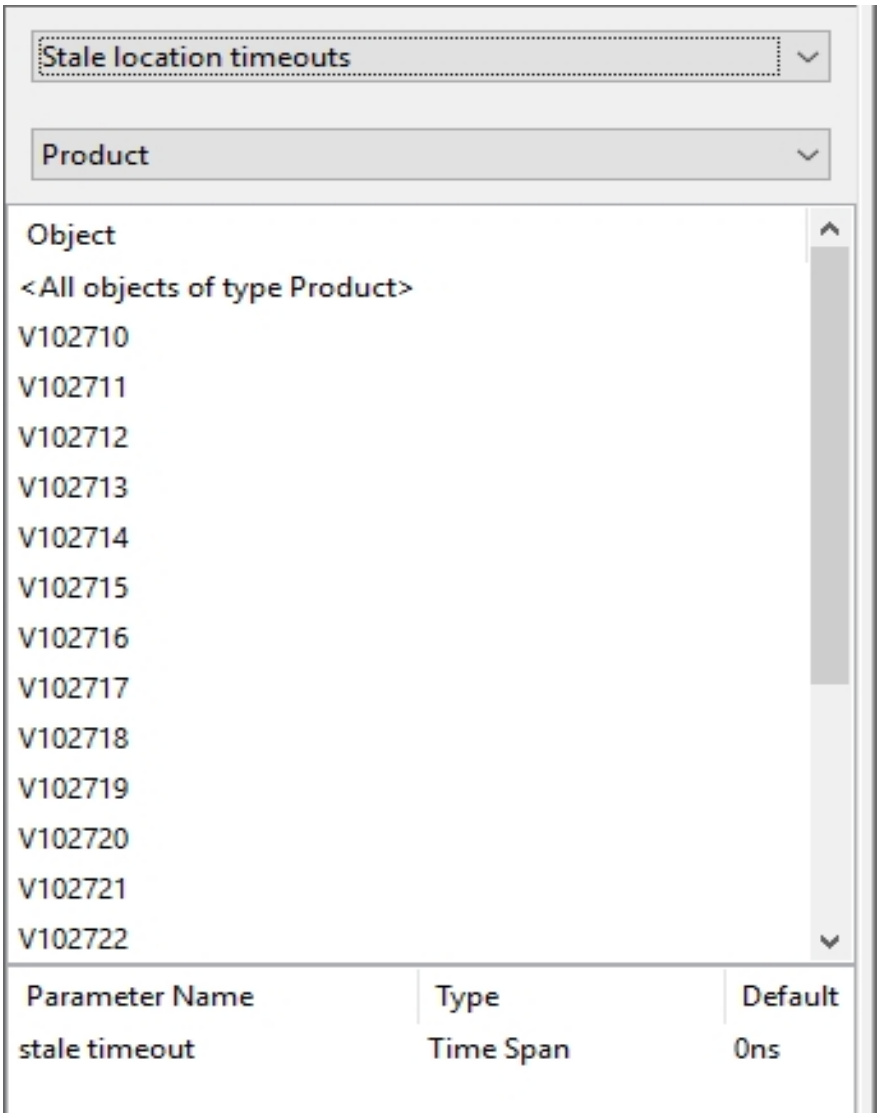
Stale Location Detection

The stale location detection feature is used to report objects for which the associated tag is no longer getting locations. For example, if a tag is taken out of the coverage of the location system, the object will be left at the seen last location. The system allows a maximum time interval to be configured for object types, or for individual objects. If an object has been located using an associated tag, and then the tag stops generating sightings for the time interval configured, an assertion is generated that the object location is "stale". As soon as a new sighting is made of the tag, the stale assertion is retracted.

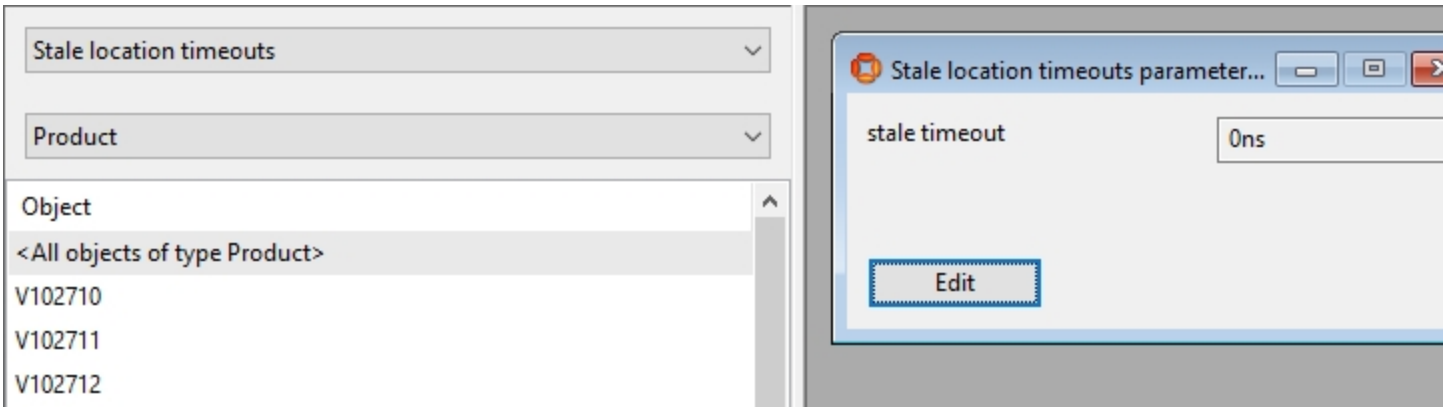
It is up to external integration code, or to business rules (if the Business Rules module has been licensed) to decide what to do when a stale assertion has been made for an object. For example, the object might be assigned a new representation indicating that its location is stale. In some cases, it may be appropriate to remove the object location when it becomes stale. The appropriate choice of action depends on the application.

Setting a Stale Timeout

Using SmartSpace Config, in **SERVICE PARAMETERS**, select Stale location timeouts, and then in the Types dropdown, select the type of object for which stale detection is required.



In order to configure stale detection for all objects of a given type, drag the row "<All objects of type …>" into the right-hand pane. To configure a single object, drag the object instance into the right-hand pane.



Now click Edit and enter the timeout to use in seconds. Normally the timeout to use will be a function of the business process, or will be related to the expected location rate of the tag attached to the object. For example, if the tag is expected to generate a sighting every three seconds, and you wish to detect when the tag has not been seen for three sightings, a timeout of 10 seconds would be appropriate. Alternatively, if the business use case requires the object representation to change if the location is more than five minutes old, set a timeout of 5*60 = 300 seconds.

Parameters for Stale Location Detection

There is only a single parameter to control stale location detection.

stale timeout

How long to wait for a sighting before asserting that an object is stale. Set to zero (the default) to disable stale location detection.

Assertions for Stale Location Detection

The output of stale location detection uses an assertion. This will typically be examined and acted upon using some integration code, or via the Business Rules component if it has been licensed. For testing and development it can be viewed using **TYPES/OBJECTS** in SmartSpace Config.

stale flag <Object>

The stale flag will be set to true for any object currently detected as stale, and will be removed when the object tag is sighted again.

Location Removal Mechanism

The location removal feature is similar to tag removal and object deletion supported via assertions in SmartSpace core. Location removal allows an assertion to be made that causes the current location of the object to be removed, and the assertion will be retracted once this has been done. Location removal doesn't require any configuration parameters – it is purely implemented using a single assertion.

Assertions for Location Removal

remove location pending flag <Object>

When this flag is asserted for an object, the object's location will be removed, and then assertion will be retracted. Any process action that must be executed after the object has been removed can be safely executed when the assertion has been retracted. If the object is still getting located by some other means, such as via driven objects, or an associated tag, then the object might be located again (possibly immediately).

Example of Location Removal Use

One example of the use of the location removal mechanism is to clean up objects that have not been located by their tag for some process-related time interval. To implement this, specify a stale timeout for the object type, and then when the stale flag is set for an object, set the location removal pending flag. This can be can be done using external integration code, or trivially with Business Rules if they have been licensed.