



SmartSpace[®]

Location Rules Configuration
Guide

From version 3.5

Copyright © 2023, Ubisense Limited 2014 - 2023. All Rights Reserved. You may not reproduce this document in whole or in part without permission in writing from Ubisense at the following address:

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

Tel: +44 (0)1223 535170

WWW: <https://www.ubisense.com>

All contents of this document are subject to change without notice and do not represent a commitment on the part of Ubisense. Reasonable effort is made to ensure the accuracy of the information contained in the document. However, due to on-going product improvements and revisions, Ubisense and its subsidiaries do not warrant the accuracy of this information and cannot accept responsibility for errors or omissions that may be contained in this document.

Information in this document is provided in connection with Ubisense products. No license, express or implied to any intellectual property rights is granted by this document.

Ubisense encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

UBISENSE®, the Ubisense motif, SmartSpace® and AngleID® are registered trademarks of Ubisense Ltd. DIMENSION4™ and UB-Tag™ are trademarks of Ubisense Ltd.

Windows® is a registered trademark of Microsoft Corporation in the United States and/or other countries. The other names of actual companies and products mentioned herein are the trademarks of their respective owners.

Contents

- Location rules** **1**
 - Overview 1
 - Audience 2
 - Prerequisites 2
 - Installation 2
 - SmartSpace Config 2
- Automated tag association** **2**
 - Creating a Tag Association Point 2
 - Parameters for Tag Association Points 5
 - Support for Multiple Tag Positions 7
 - Support for Tag Types 7
 - Assertions for Tag Association Points 8
 - Web Site Operation 9
 - Association Status Messages 14
 - Data Warning Timeout 15
 - URL to a Specific Association Point 15
- Robust assertion points** **18**
 - Configuring Assertion Points 18
 - Parameters for Assertion Areas 23
 - Assertions for Assertion Areas 25
 - Advanced use 25
 - Details of Assertion Operation 25
- Parking** **27**
 - Location Snapping 27
 - Parameters for Parking Areas 28
 - Multiple Parking Areas 28
 - Layout of Parking Bays 29
- Driven objects** **30**

Configuring Driven Objects	30
Parameters for Driven Objects	32
Assertions for Driven Objects	34
Details of Driven Objects Operation	34
Example Layout Using "Alternates"	34
Stale Location Detection	37
Setting a Stale Timeout	37
Dynamically Setting Stale Timeouts	39
Removing Stale Object Locations	39
Parameters for Stale Location Detection	39
Assertions for Stale Location Detection	40
Location Removal Mechanism	41
Assertions for Location Removal	41
Example of Location Removal Use	41

Location rules

Overview

This document describes the configuration of the features that make up the Location rules component of SmartSpace. The features described are:

Automated 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 location

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 bay snapping

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

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 3.2 and later. See *SmartSpace Installation* on the Ubisense Documentation Portal for information on requirements and installation process. You will need a license for Location rules in order to install it.

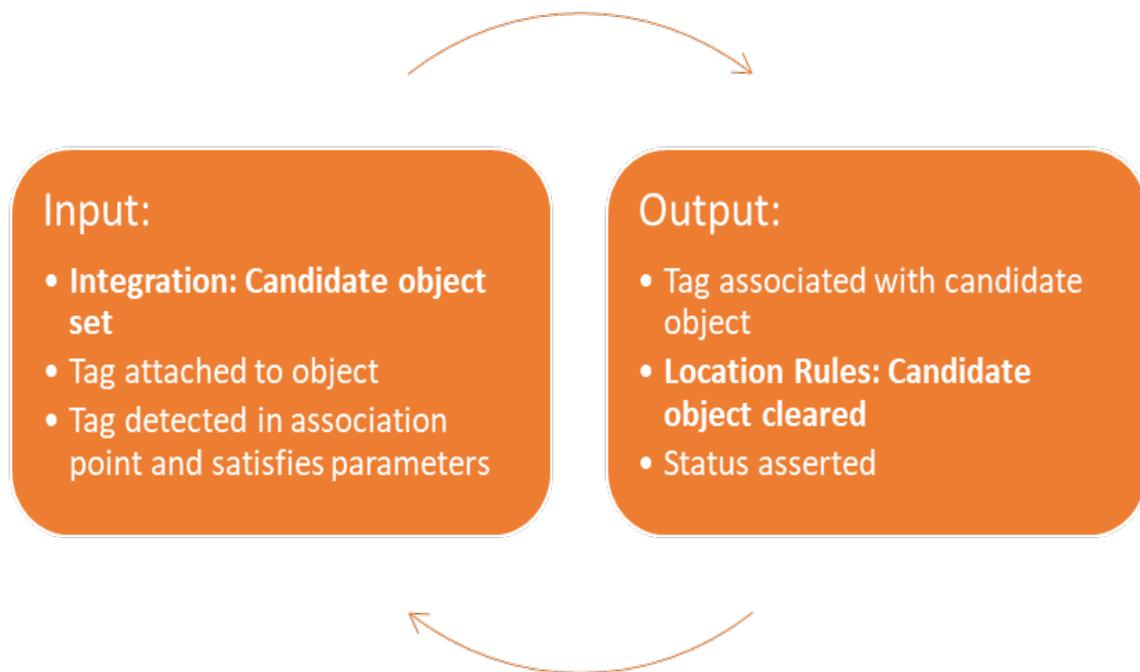
The automatic association status web page is part of SmartSpace Web. See the installation instructions for SmartSpace Web in *SmartSpace Installation* on the Ubisense Documentation Portal for requirements and installation process.

SmartSpace Config

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

Automated 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.



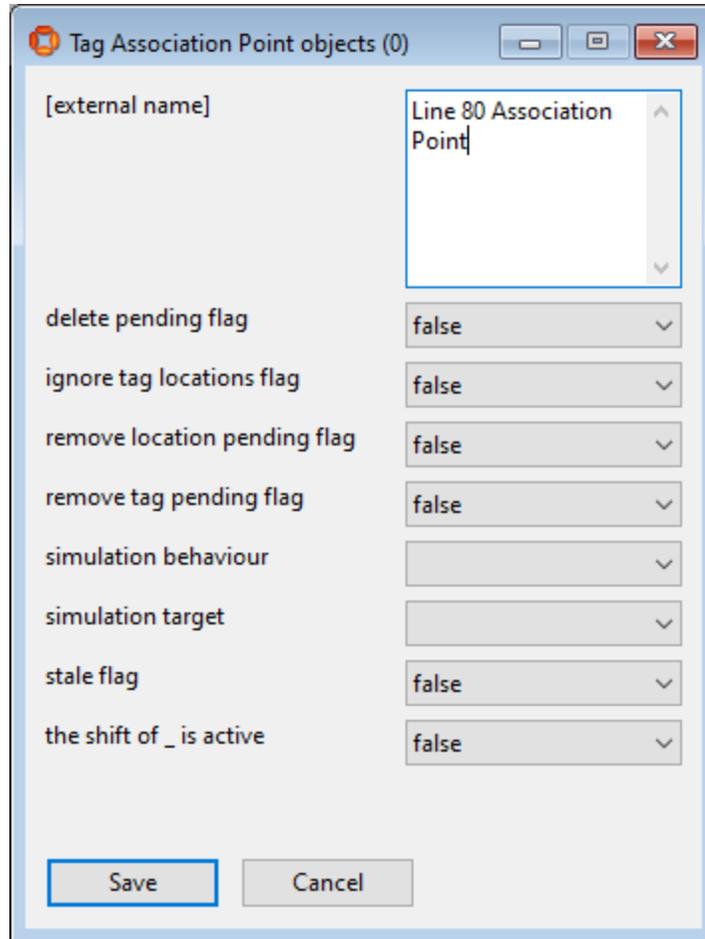
The cycle of operation of an association point

Creating a Tag Association Point

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

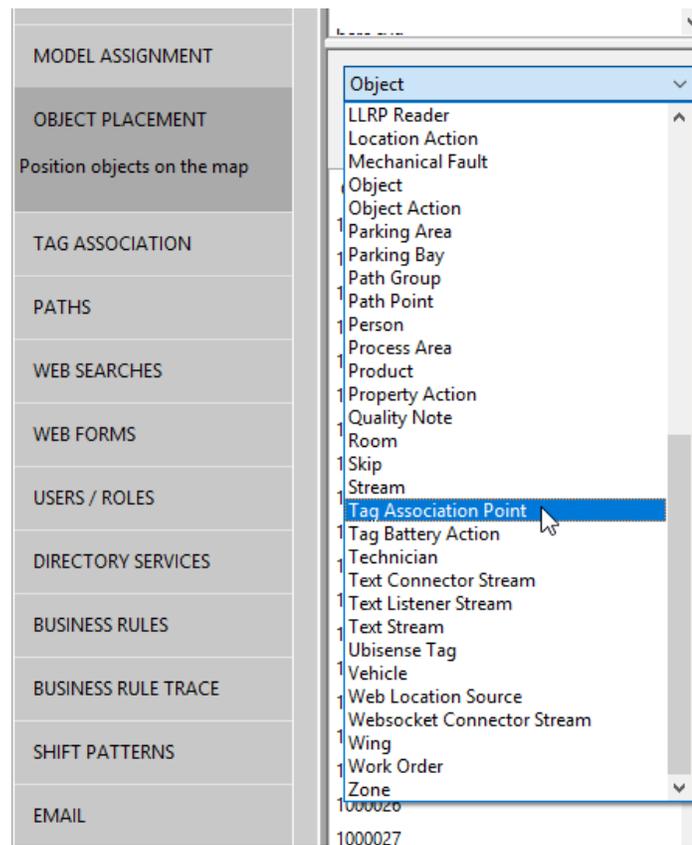
1. If you haven't already done so, set up a representation model for tag association points so you can place them and see where they are placed. See *Importing an object representation* and *Adding a representation to a type* on the Ubisense Documentation Portal for how to set a representation on a type 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 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**.

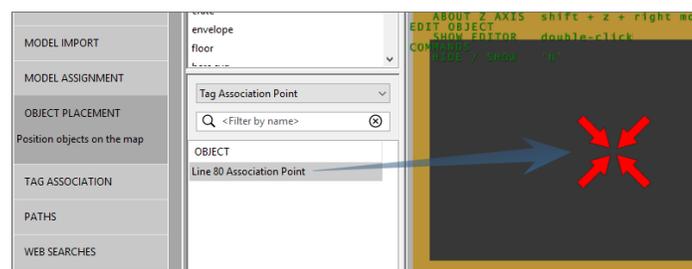


Creating a new tag association point

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.



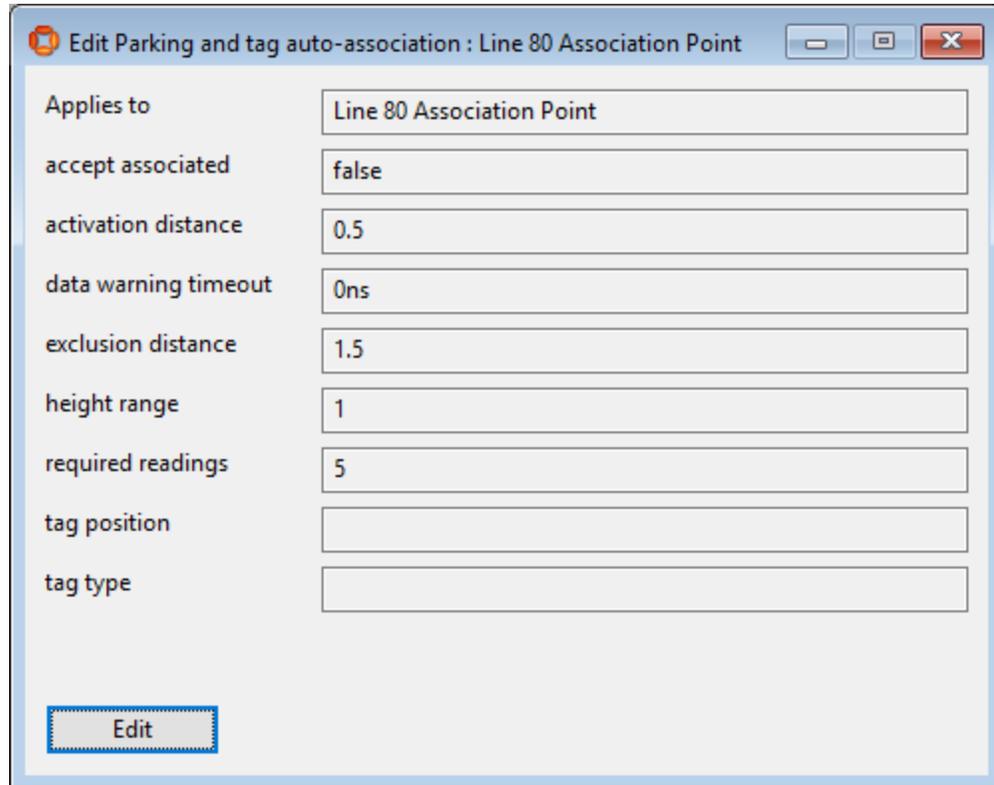
Placing the tag association point: select Tag Association Point type



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. Drag the Tag Association Point type into the object

browser and double-click the association point to display its parameters. Click **Edit** to change them.



Parameter	Value
Applies to	Line 80 Association Point
accept associated	false
activation distance	0.5
data warning timeout	0ns
exclusion distance	1.5
height range	1
required readings	5
tag position	
tag type	

Edit

Parameters for a Tag association point

Note: You can set parameters for all instances of the Tag Association Point type by double-clicking '**Tag Association Point**' **objects** 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. The association will made when the tag has been seen the number of times determined by the [required readings](#) parameter.

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. For a tag to be considered for association, it must both be within the activation distance and be the only tag within the exclusion distance.

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 tags 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.

tag type

From version 3.4, a tag type (with type Tag Type) can be associated with an association point. This parameter is set by choosing one of the currently-defined tag types from a drop-down list. (There is no attempt to reinforce referential integrity so the tag type assigned here will not have a name if the tag type is deleted from the system. Deleting a tag type is not an expected behavior in normal datasets.) Use of this parameter is described in [Support for Tag Types](#).

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. See [Assertions for Tag Association Points](#).

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.

Support for Tag Types

From version 3.4, Tag Association Point objects can have an associated tag type, in which case they will only assign tags that either have no type or have the exact same type. After assignment they will ensure that the assigned tag now has the exact same type, so this mechanism can be used to protect users from assigning inappropriate tags within a tag cycle, and to allow designers to assign tag types (and hence tag battery monitoring and filter properties) automatically.

Assignment process

Pre-assignment check: When a tag is ready to be associated, various properties of the tag are checked (for example, the tag battery status is checked). At this stage, a new check has been introduced on the tag type:

1. If the tag type of the Tag Association Point is undefined, the check is OK
2. If the tag type of the tag to be assigned is undefined, the check is OK
3. Otherwise, the check is OK if and only if the type of the Tag Association Point matches the type of the tag to be assigned.

If the check is OK then the assignment is done (assuming no other check fails).

If the check is not OK then the assignment fails with the error message "tag has the wrong type" and the following health monitoring counters are incremented:

- auto_association_bad_type
- auto_association_errors

Assignment actions: When a tag is assigned by a Tag Association Point, if this point has a tag type specified, then the service will ensure that the tag has this tag type.

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 happen 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.

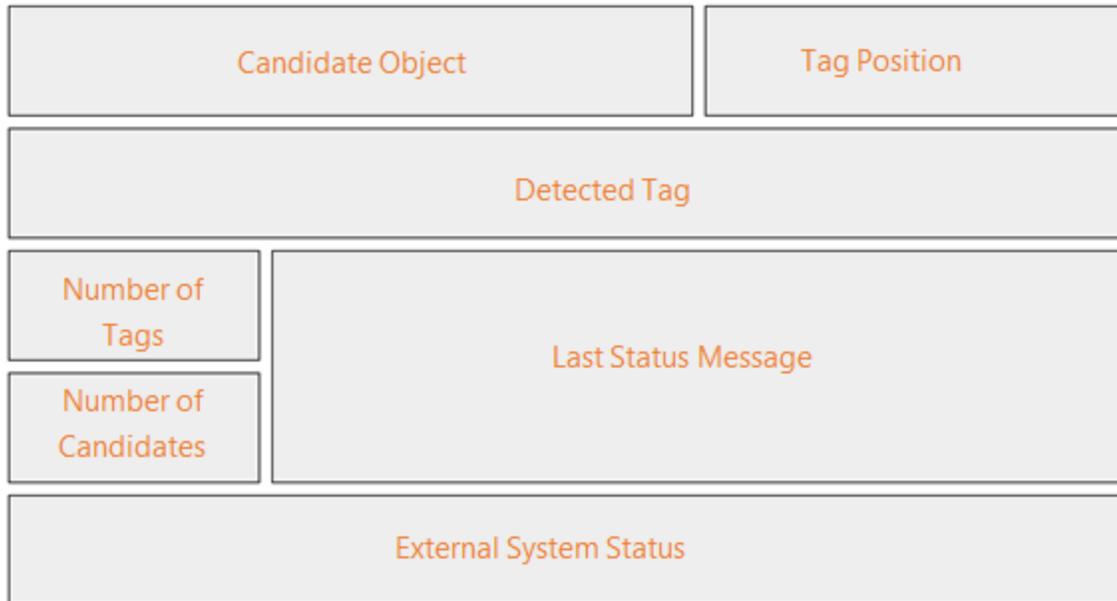
Web Site Operation

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

Line 80 Association Point



SmartSpace © 2017 - Ublisense Limited

The parts of the association status web page

Association Point:

Seconds to wait before highlighting errors:

Line 80 Association Point

The screenshot displays a web interface for 'Line 80 Association Point'. At the top, there are two input fields: 'Association Point: Line 80 Association Point' and 'Seconds to wait before highlighting errors: 20'. Below these is a large yellow bar. Underneath is a green bar containing the timer '00:01:00:02'. To the left of the main content area is a green box with '1T' and a yellow box below it. The main content area is a large yellow box with the text 'Waiting for data'. At the bottom of the interface is a grey bar.

SmartSpace © 2017 - Ubisense Limited

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

Association Point:

Seconds to wait before highlighting errors:

Line 80 Association Point

V102713	Windshield Left
1P	

SmartSpace © 2017 - Ubisense Limited

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

Association Point:

Seconds to wait before highlighting errors:

Line 80 Association Point

V102713	Windshield Left
00:01:00:09	
1T	
1P	

SmartSpace © 2017 - Ubisense Limited

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

Association Point:

Seconds to wait before highlighting errors:

Line 80 Association Point

The screenshot shows a software interface with several grey rectangular areas. A prominent green box on the right side contains the text "Windshield Left" in bold black font. Below this, another green box contains the text "Associated V102713 ↔ 00:01:00:09" in bold black font. The interface also includes a dropdown menu at the top left and a text input field at the top right.

SmartSpace © 2017 - Ubisense Limited

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
tag has the wrong type	the tag type 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 component if licensed, to trigger other actions or track process.

i 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* object 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.

i 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.

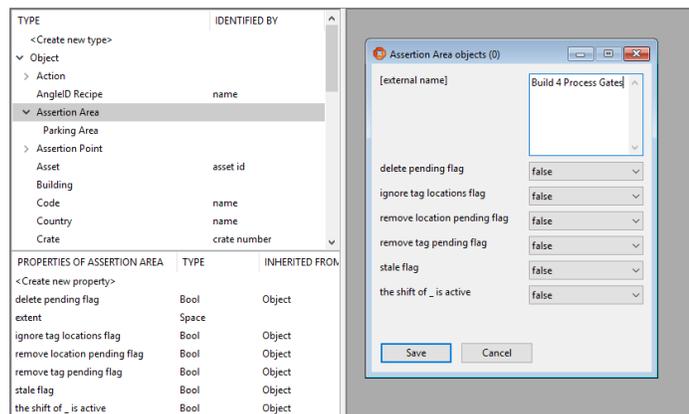
Configuring Assertion Points

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 *Types and objects* and *Tag association* on the Ubisense Documentation Portal for how to do this. We assume the asset type is called *Product*.

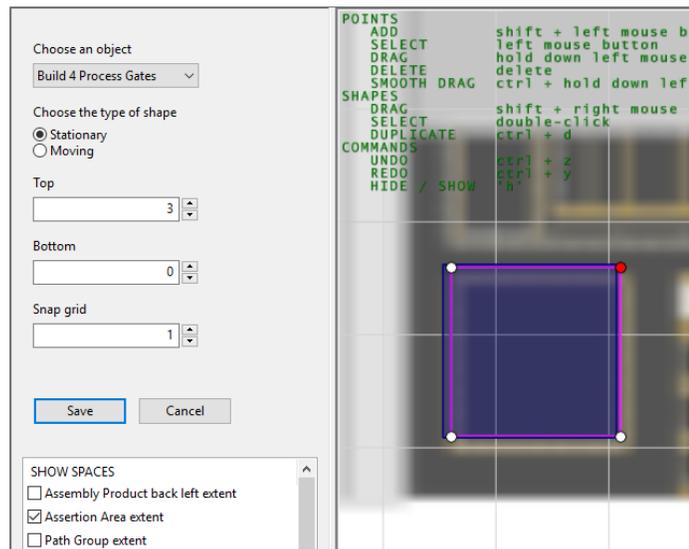
1. If you haven't already done so, set up a representation model for Assertion Point so you can place them and see where they are placed. See [Adding a representation to a type](#) on the Ubisense Documentation Portal for how to set a representation on a type in

SmartSpace Config. Make sure the representation is scaled to a suitable size, and the origin set appropriately.

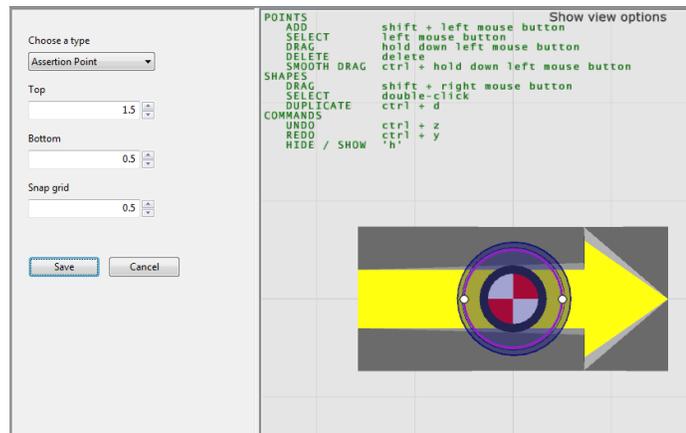
2. Create an object for the assertion area. Using **SmartSpace Config**, under **TYPES/OBJECTS** drag out the *Assertion Area* type into the object browser. Double-click **<Create new object>** from the **Assertion Area objects** window. Enter the name of the assertion area into the name field, and click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



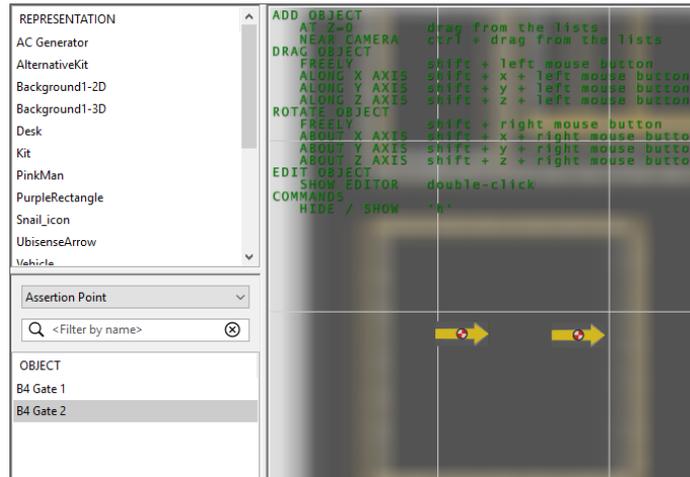
3. Similarly create an instance of type *Assertion Point* for each assertion point you want in the assertion area. In our example, we will create assertion points called "B4 Gate 1" and "B4 Gate 2". Drag the *Assertion Point* type into the object browser, double-click **<Create new object>** and then 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**.



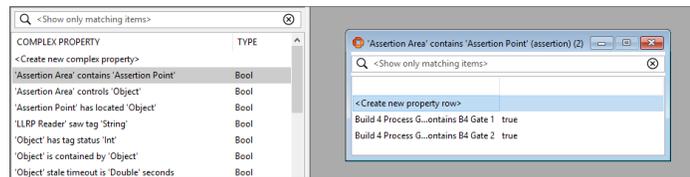
- Also create a default space for the origin of the assertion point type – in this case a small space with a limited height, so that it can be contained within the extent of the assertion area. Select the *Assertion Point* type, then the “origin” spatial property. Under **DEFAULT SPACES**, double-click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



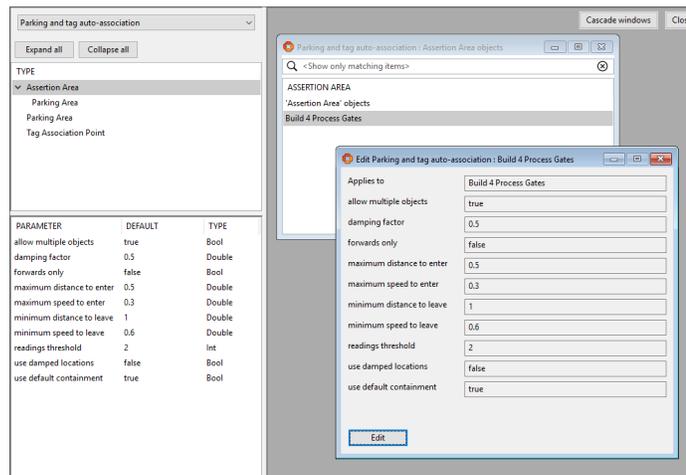
- Place each of the assertion points that were created. In the OBJECT PLACEMENT task, select the *Assertion Point* type from the filter dropdown, and then drag each assertion point instance into the location of shape on the map. If *Assertion Point* does not appear as a choice, you have not set a representation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



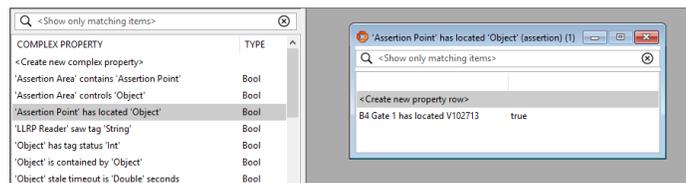
- At this point, the spatial properties you have just set up should automatically populate the Assertion Area contains Assertion Point property. To check this, go to **TYPES / OBJECTS** and in the **COMPLEX PROPERTY** list find 'Assertion Area' contains 'Assertion Point'. Drag this into the right hand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



- Configure the parameters of the assertion area. Go to the **SERVICE PARAMETERS** task, choose Parking and tag auto-association, and then drag the Assertion Area type into the object browser. In the dialog that displays, double-click the area you created. You can now 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.



- Now we define a spatial property of the Product type to automatically make product instances controlled by the assertion area when they enter its extent. In **TYPES / OBJECTS**, select the Product type, and under **PROPERTIES OF PRODUCT**, double-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 **MONITORED SPATIAL RELATIONS**, double-click **<Add new request>** and for Container select *extent of 'Assertion Area'* and for Contained select *origin of 'Product'*. Click **Save**.
- 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.

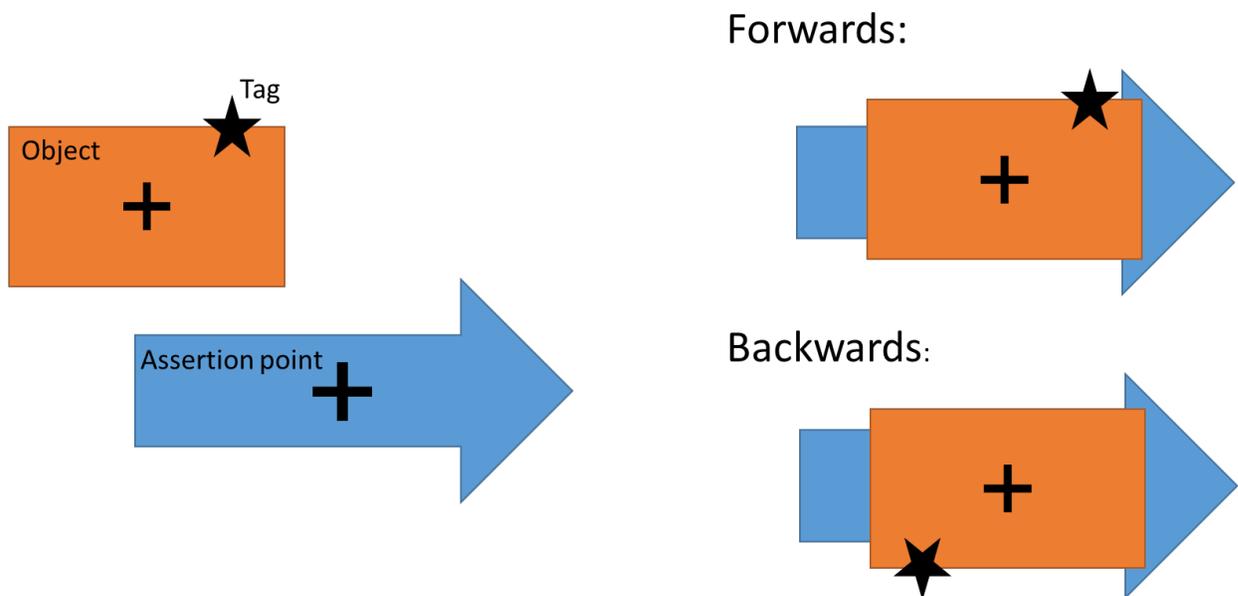


Parameters for Assertion Areas

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

$0 \leq \text{damping factor} < 1$. 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 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.

use damped locations

Where location streams are noisy, smoothing locations by using the calculated damped position for object locations rather than the original tag x,y,z. locations can make dealing with these streams much easier, and can, for example, reduce the amount of jitter you see on the web map

When **used damped locations** is set to true, the damped tag position is used for any subsequent location inference of the associated object while that object is controlled by the association area. Defaults to "false".

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 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 lay out 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 double-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**.

The screenshot shows the 'Create a new object type' dialog in SmartSpace Config. On the left, a sidebar lists various configuration tasks, with 'TYPES / OBJECTS' selected. The main dialog area contains the following fields and options:

- Create a new object type called:** A text input field containing 'Crate'.
- whose parents are:** A list of checkboxes with the following items:
 - Assertion Area
 - Assertion Point
 - Object
 - Parking Area
 - Parking Bay
 - Product
 - Tag Association Point
 - Ubisense Tracked Object
 - Ubisense Trail
 - Ubisense Waypoint
- and whose instances are uniquely identified by a property called:** A text input field containing 'crate number'.
- At the bottom, there are two buttons: 'Save' and 'Cancel'.

2. Repeat to create type "Work Order", named with the property "id".
3. Import and assign representation models for these two types. See *Model import* and *Model assignment* for how to set a representation on a type in SmartSpace Config.

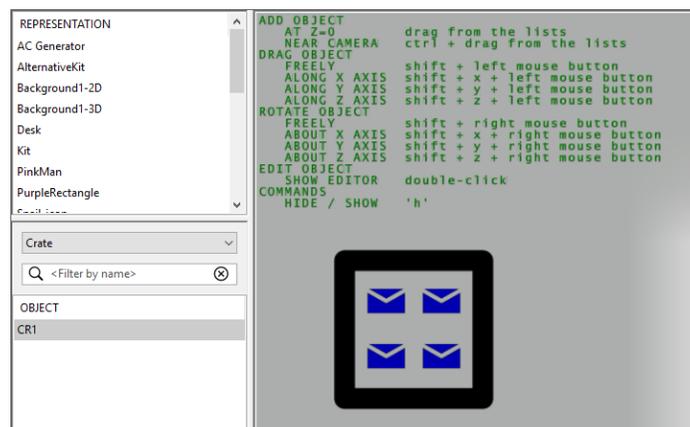
- Now we will set up the default layout parameters for crates. In the **SERVICE PARAMETERS** task, select the **Co-located objects** configuration, and then the **Crate** type. (Use the **Expand All** button to display the object hierarchy, if necessary.) Drag the **Crate** object into the object browser and double click '**Crate**' objects to set the parameters for all crates. Click **Edit** in 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**.

Parameter	Value
Applies to	'Crate' objects
X direction capacity	2
X direction spacing	1
Y direction capacity	2
Y direction spacing	1
Z direction spacing	1
alternate orientation	0
alternate spacing	0
default orientation	0
direction to use alternates	not used
fill X first	false
fill method	insert into first gap
minimum distance to leave	10
origin X	middle
origin Y	middle
starting corner X	left
starting corner Y	top
unset locations on leaving	false

Buttons: Save, Cancel

- 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 double-click **<Create new object>** and enter **CR1** as the crate number. Click **Save**.
- 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 PROPERTY** list drag the *'Object' is contained by 'Object'* property into the right-hand pane. In the window created, double-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 be 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.

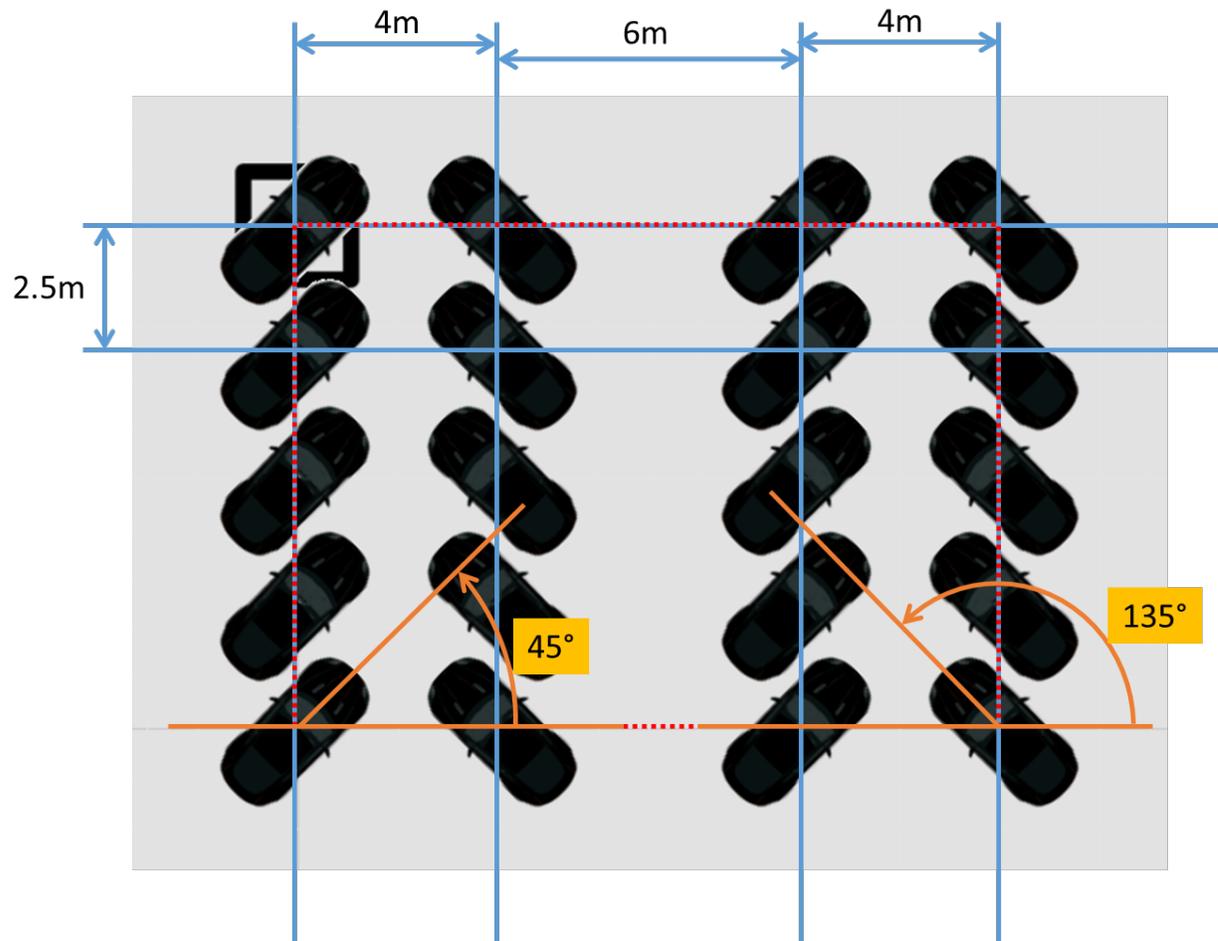
For example, if no tags are sending sightings to the location cell, no update of driven objects will occur. This means that if you add a row to 'Object' is contained by 'Object' you may not see the contained object get located until another location event occurs. In operational use, with tag events arriving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

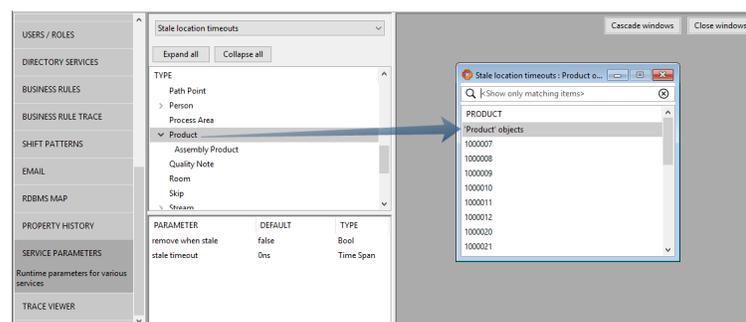
Stale Location Detection

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. These timeouts can also be dynamically overridden. 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.

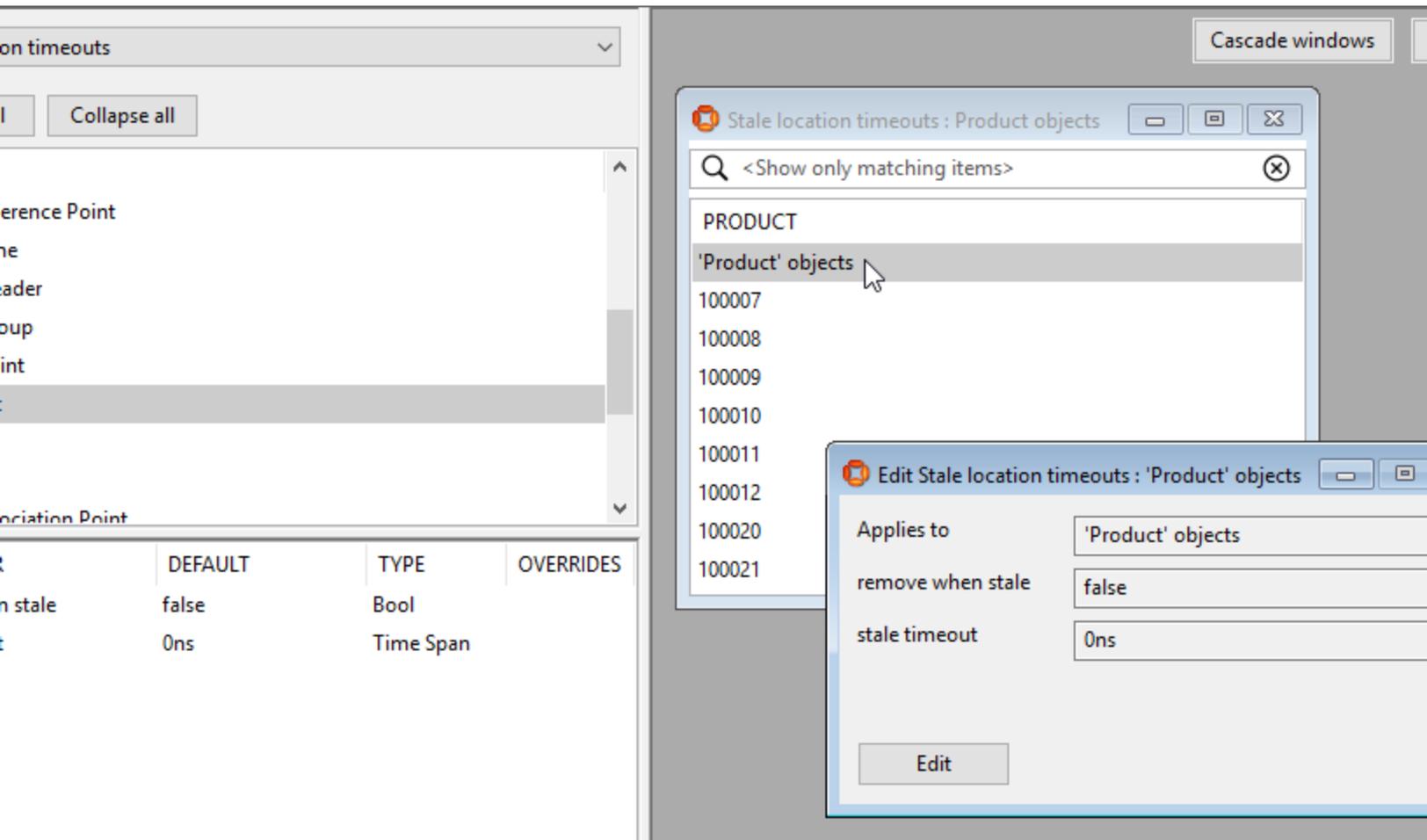
It is up to external integration code, or to business rules (if the Business rules component has been licensed) to decide what to do when a stale assertion has been made for an object. For example, the object might be assigned a new representation indicating that its location is stale. In some cases, it may be appropriate to remove the object location when it becomes stale. The appropriate choice of action depends on the application. A simple method for cleaning up stale locations can be configured without the need for business rules or external integration.

Setting a Stale Timeout

Using SmartSpace Config, in the **SERVICE PARAMETERS** task, select **Stale location timeouts**, and then locate the type of object for which stale location detection is required from the list of objects. (Use the **Expand All** button to display the object hierarchy, if necessary.) Drag the object type into the object browser to open a window listing all instances of this type.



In order to configure stale detection for all objects of a given type, double-click the row '**object name**' objects. To configure a single object, double-click the particular instance you require.



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 $5 \times 60 = 300$ seconds.

The output of stale location detection uses an assertion **stale flag <Object>**. 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.

Dynamically Setting Stale Timeouts

In addition to the method described in [Setting a Stale Timeout](#), there is an assertion, which appears in the TYPES / OBJECTS task as '**Object**' stale timeout is '**Double**' seconds: **Bool**, that allows a timeout specified in the service parameters to be overridden dynamically.

If an object has a timeout specified in this assertion, it will override the timeout specified in the service parameters. This allows the stale timeout for an object to be modified using Business rules, or via one of the integration methods for business properties. For example, if an object's tag is expected to be seen less frequently when it is in a particular part of a process, the assertion can be used to increase the stale timeout while the object is in that part of the process.

Removing Stale Object Locations

A simple method for cleaning up stale objects from the map without the use of a business rule or external integration can be configured by using the **remove when stale** service parameter which appears under **Stale location timeouts** in the SERVICE PARAMETERS task.

This parameter is a Boolean and defaults to false. If it is configured to be true for an object, then when that object's stale flag becomes set, the object's location will also be removed (via the **remove location pending flag**).

This provides a simple configuration to remove stale objects from the map. Business rules would still be needed to support more complex rules for removing objects based on combinations of the stale flag and other process state.

Parameters for Stale Location Detection

The following parameters are used 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.

remove when stale

If this is configured to be true for an object, then when that object's stale flag becomes set, the object's location will also be removed (via the **remove location pending flag**). Defaults to false.

Assertions for Stale Location Detection

The following assertions are used by the Stale location detection feature. They 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 they 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.

'Object' stale timeout is 'Double' seconds: Bool

If an object has a timeout specified in this assertion, it will override (dynamically) the timeout specified in the service parameters.

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 done using external integration code, or trivially with Business Rules if they have been licensed.