



**Weave**

Business Integration Framework

# **Weave System Administrator Guide Excerpt Feb 2022**

1. Administration Guide	4
1.1 Architecture	6
1.1.1 Concepts	8
1.1.2 Data Concepts	12
1.1.3 About OSGi	14
1.1.4 Getting Started	15
1.1.4.1 Basic Functions	15
1.1.4.2 Search Functions	18
1.1.4.3 Selection functions	18
1.1.4.4 Reporting Functions	19
1.1.4.5 Extended Map Functions	20
1.1.4.6 Natural Language Geocoding	20
1.2 Configuration	21
1.2.1 Configuration Null Values	23
1.2.2 Configuration Processing Instructions	24
1.2.3 Adding custom projections	26
1.3 Security	26
1.3.1 Authentication via URL parameter	35
1.3.2 Encrypting content in the security.xml file	40
1.3.3 Forcing a user to login	42
1.3.4 Securing the Admin UI	43
1.3.5 Spring Security	43
1.3.6 SAML	46
1.3.6.1 SAML - Azure AD sample	48
1.3.7 Clients	50
1.3.8 Windows Security	56
1.3.8.1 Alternate Windows Security Setup	69
1.3.8.2 Windows Authentication Examples	73
1.3.9 Database Security	75
1.4 Logging	77
1.4.1 Logging for Legacy Weave Versions	84
1.5 Mapping	91
1.5.1 WMS Server	92
1.6 Reporting	97
1.6.1 BIRT	97
1.6.1.1 Customising BIRT Reports	101
1.6.1.1.1 Adding Maps to BIRT Reports	106
1.6.1.1.2 Adding Data to BIRT Reports	118
1.6.1.1.3 Reporting no data generated	135
1.7 Application Integration	137
1.7.1 Google Earth	139
1.7.2 Google StreetView	140
1.7.3 Nearmap	143
1.7.4 Integrating with AssetMaster	147
1.7.5 Integration with Authority	150
1.7.6 Integrating with Confirm	154
1.7.7 Integrating with ECM	159
1.7.8 Integrating with Hansen	160
1.7.8.1 Integrating with Hansen (Update)	166
1.7.9 Integration with Pathway	172
1.7.10 Integrating with ProClaim	179
1.7.11 Integrating with Trim	182
1.7.12 WeaveLink	188
1.7.12.1 Calling WeaveLink from a URL	191
1.7.13 Launching Weave	191
1.8 Tile Caching	193
1.8.1 Building a Tile Cache	194
1.9 Spatial Editing Introduction	196
1.10 Non-spatial Editing Introduction	215
1.11 Weave Console	221
1.11.1 Weave Metadata Commands	225
1.12 Server Status	227
1.13 Administration Tool	229
1.13.1 Bookmark Manager Tool	230
1.13.2 Bundles Tool	232
1.13.3 Config Tool	233
1.13.4 Console Tool	235
1.13.5 Data Source SQL Tool	235
1.13.6 Data Sources Tool	237
1.13.7 File System Tool	238
1.13.8 Indexes Tool	239
1.13.9 Log Tool	241
1.13.10 Map Engines Tool	242
1.13.11 Notepad Tool	242
1.13.12 Server Health Tool	243
1.13.13 Server Status Tool	244
1.13.14 Spatial Engines Tool	245

1.13.15 Support Tool ..... 246  
1.14 System Metrics ..... 246  
1.15 Reverse Proxy ..... 249

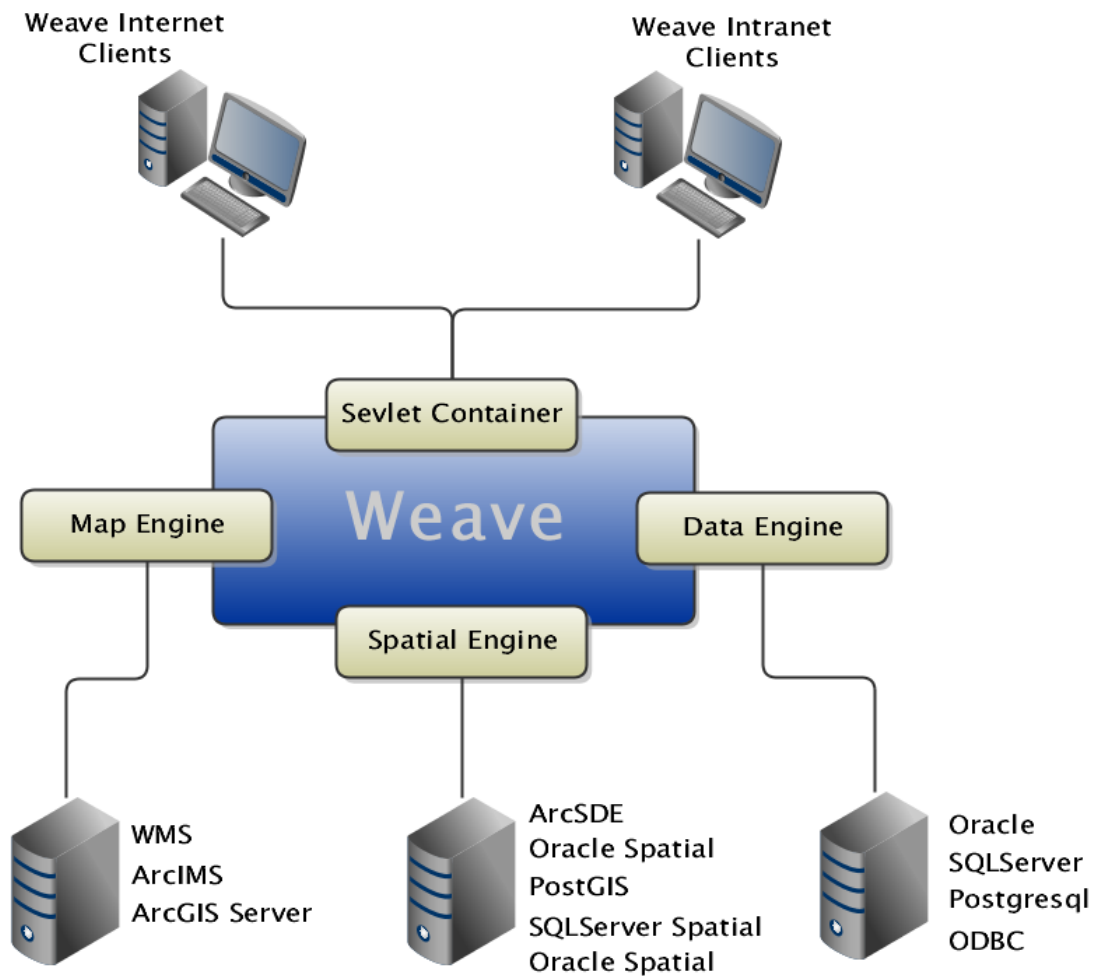
## Administration Guide

This guide provides the basic information needed for configuring a Weave server. It provides an introduction to the way the Weave server works and a reference to the items required to configure the server.

The configuration reference provides a reference for the items that you can use to build up a functional server, along with examples showing the typical usage of those items. Configuring the Weave server primarily involves a single configuration file that requires no previous knowledge to edit. The configuration is stored as an XML file however, so knowledge of XML could be an advantage. The examples presented with the configuration items, while valid, are not based on a single data set. Their purpose is only to provide a reference implementation for the configuration item and not to build up a working system. It is assumed that the Weave administrator will have enough knowledge about the systems that Weave is intended to integrate with so as to be able to adapt the provided examples and to construct a meaningful configuration.

Weave is made up of many different components which contribute to the entire system. As can be seen in Figure 1.1, "Weave Topology", Weave provides many ways to connect and consume data without being reliant on vendor specific technology. This is accomplished by using Open, State of the Art components that allow Weave adapt to your internal data requirements instead of you having to go down a path of 'Vendor lock in'.

We invite you to think about the possibilities available now that you are running Weave. No longer are you required to move along a path that is either not necessary or that you are not ready to take. Your data can come from a variety of sources, be it Open Source or Proprietary. You can pick and chose the best software that meets your needs and be assured that Weave can integrate with it. You also have the freedom to extend Weave to suit your needs. With Weave it is now possible to think in other dimensions when looking at ways to expose data to your users and how to best manage that process.



## **Openness**

Weave makes use of open, non-proprietary APIs and SDKs as well as support for multiple database and spatial engines.

## **Integration of Map Services**

The system will simultaneously work with many map services such as Gaia, Apollo, ArcIMS, ArcGIS Server, AutoCAD Map, MapXtreme, MapServer and others.

## **Integration of Spatial Data**

Weave provides powerful data integration from multiple spatial data sources such as Oracle Spatial, ArcSDE, Shapefiles, PostGIS, WFS, GML and others.

## **Integration of Non-Spatial Data**

Weave provides powerful data integration from multiple non-spatial data sources through JDBC or ODBC connection.

## **Insulation**

The data complexity is hidden from the end users. Users do not need to know SQL or database schemas (only the administrator does).

## **Simple Configuration**

The system is configurable through a simple XML file providing open access.

## **Web/Browser-Based**

Weave has browser-based clients using JavaScript and HTML. No browser plugins are required.

## **Stateful System**

Weave works like a local application despite being based on a freely available web browser. Weave allows users to interact with the data in several steps without cookies (maintaining the state for every user at any point in time), unlike ordinary (stateless) Internet based products. Users can perform individual searches (textual and spatial) and spatial operations (spatial joins, buffers, etc.) and combine the results at will, exploiting the flexibility of the system.

## **Customisable/Extensible**

The system can be customised and expanded by third parties. Weave has been built with its own open software development kit which is supplied with the product at no additional cost.

## Minimal Footprint

There are minimal requirements needed to integrate data sources (i.e. no selection tables, no forced schemas).

## Scalable

The system is portable and scalable to accommodate the future changes in user numbers, hardware, software, networking. The licencing of Weave does not restrict the number of configurations or users served by Weave.

## Security

Role based access control is provided to control user access to functionality and information in a secure and easy manner.

## High Quality Reporting

Weave includes advanced business intelligence reporting tools with embedded map and image integration out of the box.

## Integration With Third Party Applications

Weave provides integration capabilities at the application (browser) level (provided the browser used is IE7+) and also provides integration capabilities at the server side (integrating with other server applications) like SAP, ArcGIS Server, Business Objects, etc.

## Cost Effective

The Weave Business Integration Framework provides a low cost, rapidly configured and deployed, client-server solution.


### Architecture

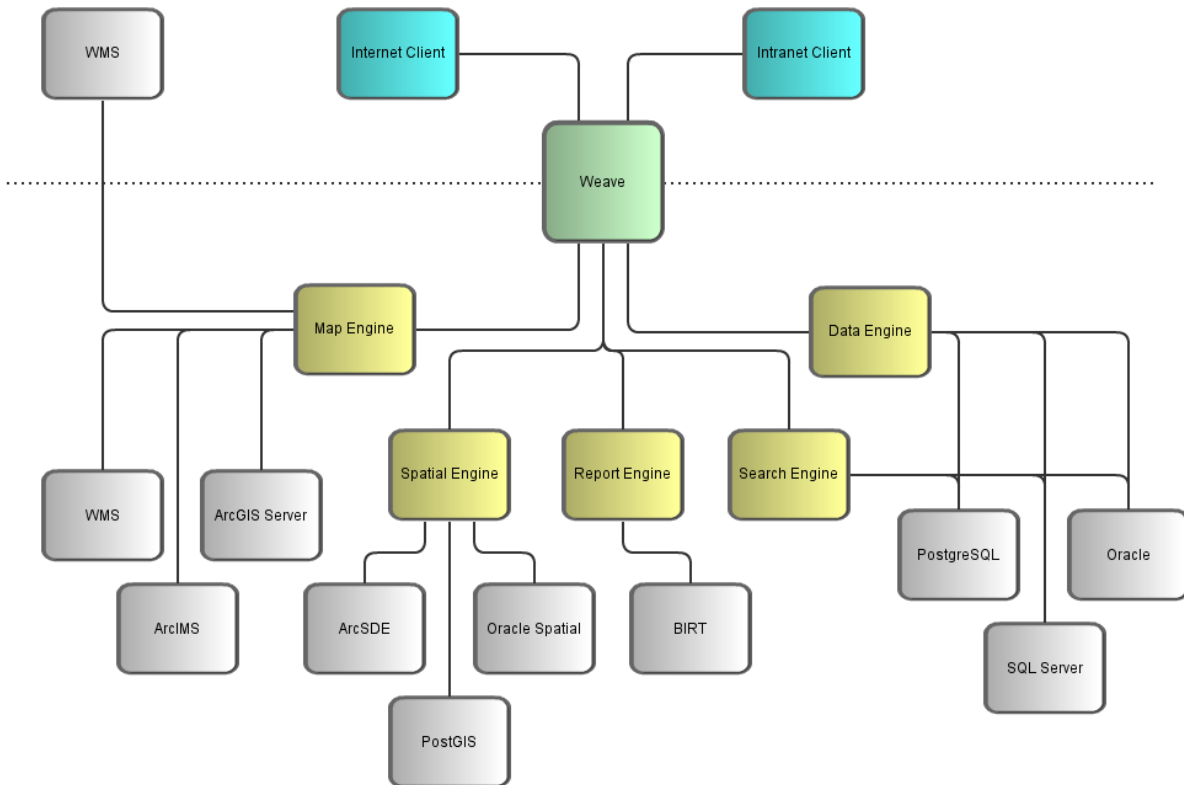
#### Overview

Weave is a Business Integration Framework with many features and functions that can help drive business decisions and solve problems that may not be possible or be too time consuming (and costly) to solve. Weave is implemented as a Client/Server system and may also be described as having a Services Oriented Architecture (SOA).

In order to understand how to configure and get the most out of Weave it is worthwhile having a close look at this architecture.

The server side component of Weave is written in the Java programming language. Java provides Weave with the flexibility and predictability that is required to run a state of the art Business Integration System and which is designed to be independent of vendor specific technology. Another advantage to using Java, is that it enables Weave to run on multiple operating systems. This provides the administrator more flexibility when deploying, customising, and maintaining Weave.

 The process of handling requests from a client can be written in either JavaScript or Java. It is also possible to have .Net based applications communicating with Weave using WebServices.



## Design Drivers

1. Products based on open, state of the art technology
2. Non vendor-specific technology
  - Open Services Gateway Initiative
  - Eclipse development platform
  - Java 2 platform enterprise edition
  - Open GIS Consortium
3. Compliance with Open Systems
4. A scalable Multi-Tier Architecture
  - Supported on Unix, Linux, and Windows
5. Integrate data from multiple sources
  - spatial and non-spatial
6. Custom configuration by site administrator
  - Flexible enough to meet many needs
7. Insulate user from data complexity
  - User doesn't care where data is located
8. Role base access control
  - Control data access in standard manner
9. No browser plugins
  - No proprietary add-ons required for browser
10. Minimal footprint
  - Minimise requirements for clients
11. Sophisticated reporting capability
  - BIRT out of the box
12. Rapid implementation
  - Initial implementation can grow as required
13. Choice
  - Wide support for standards
14. Customer driven product road map
  - We develop the framework, users develop the product

## Advantages

1. A business integration framework providing
  - Rapid incorporation of data held in standard data sources

- Connection to multiple mapping and spatial engines
  - Querying and reporting of non-spatial layers not stored inside a spatial engine
  - Role Based Access Control
  - Full set of business intelligence reporting tools
  - Server side application integration capabilities through Web Services
  - Support of multiple client applications from a single server instance
  - Extensible architecture based on OSGi standard model
  - Continuous Improvement
  - Customer Driven
2. Supports BOTH ArcIMS and ArcGIS Server
    - Lower your risks
    - Migrate as and when you please without redevelopment
  3. Supports Open GIS
    - PostGIS, MapServer, GeoServer, WMS, WFS
    - Intergraph GeoMedia WMS Data Server
  4. Second generation product
    - Mature (over 10 years), proven, many users
  5. No client plug-in
    - Faster, and better support on different Web Browsers
    - All HTML, CSS and JavaScript on client
  6. SDK included
    - Self sufficient
    - Many developers
  7. Business Integration Framework
    - Not just a GIS Viewer

## Concepts

Weave is designed to be as flexible as possible. Key to this flexibility is the notion of separating out the various components into their own features. The three concepts that need to be understood before moving on are

- Mapping Engine
- Spatial Engine
- Data Engine

### Mapping Engine

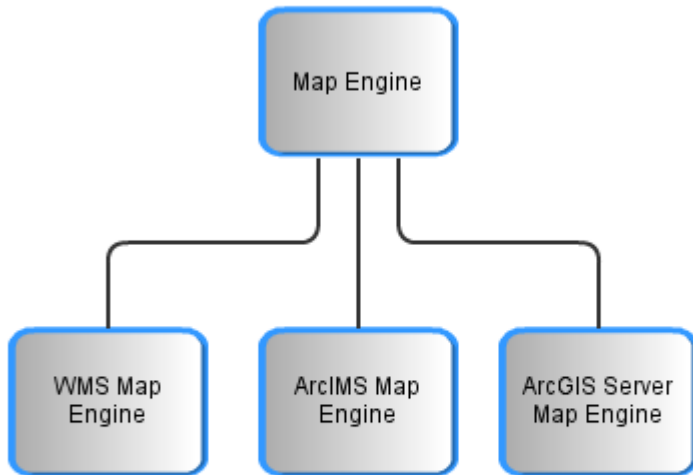
The Mapping Engine is used to communicate with the underlying mapping interfaces supported by the organisations Mapping Software. Its job is to ask the Mapping Software to generate a map of a particular size (in pixels), at a particular location (in metres, feet, degrees, etc) and with a given set of map layers turned on (the layer may actually not be visible at the actual scale selected due to Scale Dependant Renderers that have been applied to the map service).

The Mapping Engine will return either a URL to the generated image or a binary stream of data containing the requested image format. The image is then sent to the client directly from the Weave server. This approach allows the Mapping Engine Software to remain within the bounds of the secure network as the Weave client does not need to know how or where to get the map. It (the Weave Client) asks Weave and Weave will do all the communications with the Map Engine.

### Selection Layers

We did not mention the concept of user selections or acetate information being rendered on the map. The process of rendering selections is taken care of by Weave when the map is requested. Weave generates a second image with just the selections and acetates on them and combines or fuses the two images together to form one and then passes it to the client. If you are migrating from EView, this is important to note as EView required that if a layer was to be a Selection Layer, it had to be included in the ArcIMS service. With Weave this is not required as Weave handles the selection rendering itself. This an advantage as it enables sites to produce a cartographically correct static image of the area of interest and serve that to a particular set of users for viewing whilst still enabling the user to select any defined Entity.

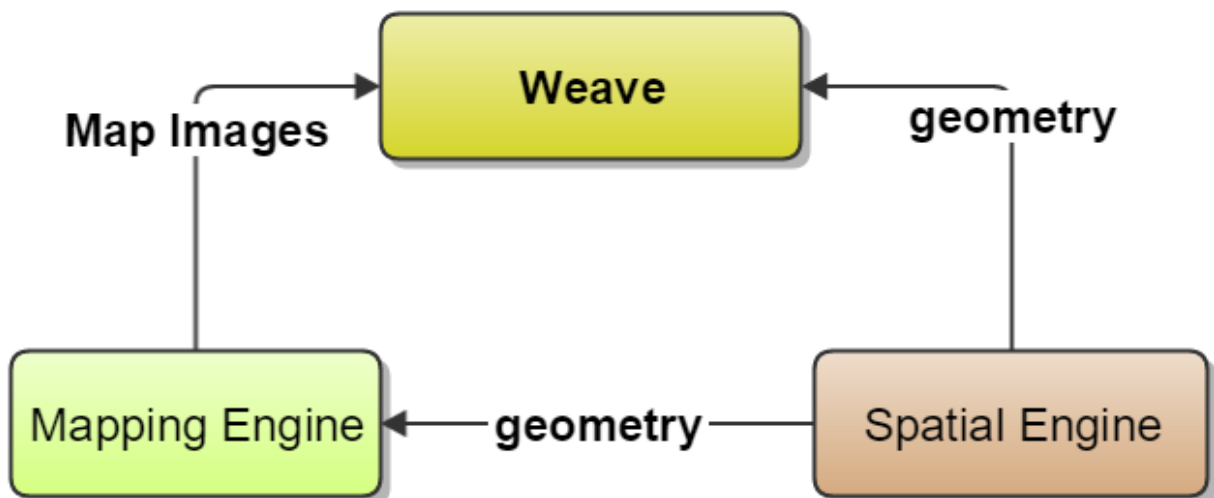




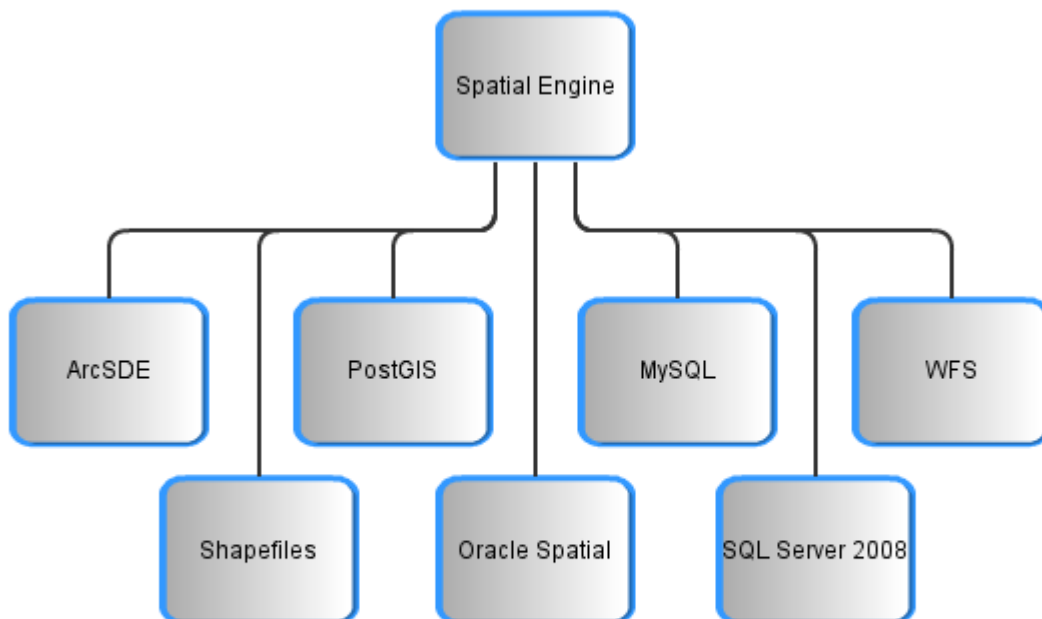
**Spatial Engine**

The Spatial Engine is used to communicate with the underlying Spatial Database (this does not have to be a physical RDBMS, it could be a file based Spatial System like shapefiles) to request geometry and undertake spatial operations (e.g. buffer, intersect, etc). The Spatial Engine does not do any rendering of maps. It is a datastore that provides the information needed to execute spatial functions and render selected features. Different types of Spatial Engines suit different requirements. File based systems are good for small organisations where data is only edited infrequently and by no more than one or two people. If however you have multiple people editing the underlying spatial data and it is changed on a regular basis, then an enterprise system that supports concurrent editing, versioning and role based access would be required.

Most of the time, a link between the Spatial and Mapping Engines is setup by the organisation. The Weave Map Engine requires this link in order to render the selected features on the map when requested.

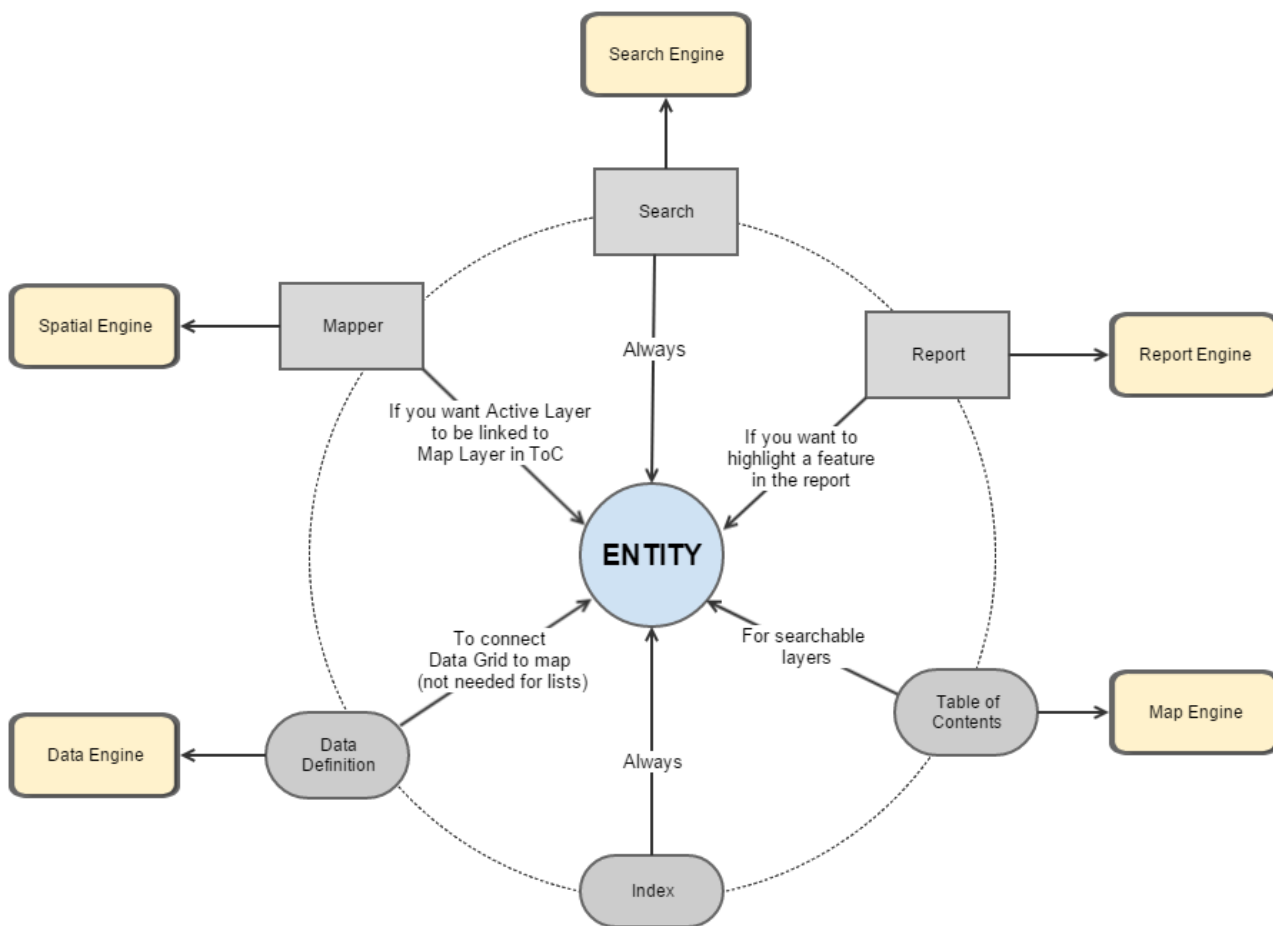


There can be circumstances where there is no link between the Mapping and Spatial Engines. For example, if you were serving static images or have a Mapping Engine that serves static images ( e.g. Image Web Server) then there is no need for such a link to be defined. If the Entities are setup correctly in the Weave configuration, then a user can select features on the map, either spatially or a-spatially and have Weave render them on top of the image that is then sent to the Weave client.



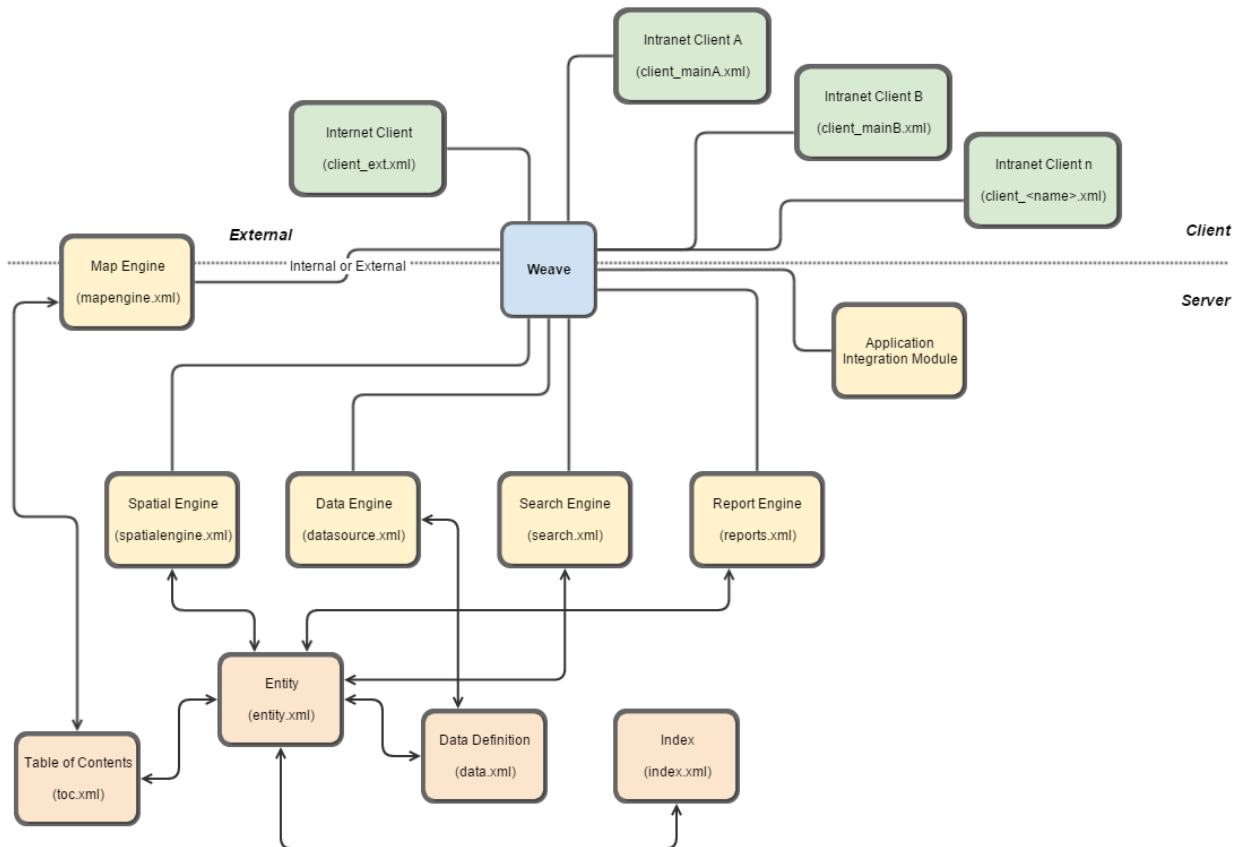
Entity

An entity provides a basic reference to an item that will be searched and report on within Weave. Its relationships within Weave are shown in the diagram below.



Relationships

All the building blocks of Weave can be brought together and their relationships shown in the diagram below.



## Data Concepts

### Basics of data handling in Weave

## Selections

The concept of separation of searching and reporting is an important concept in Weave.

For example, think of search engines like Google where users search for something and get back the results. The only way to search is by typing something in the box, and one type of result displayed; a page of top search results.

In contrast, with Weave there are multiple ways of searching for things and a multitude of ways of displaying the search results. “Searching” could be performed by drawing a polygon on a map to find all the things that fall within that polygon, or by typing in an address to search a database find the matching properties, or even by having a third party system sending a list of unique identifiers. And “Reporting” could be as simple as displaying a list of the owner names of the found properties, or generating a PDF report describing the history of the property, or sending a list of unique identifiers to another third party system.

Because of this separation and the use of a current selection, searching can be combined or split across multiple systems. For example, a user may be working in a property management system and may have selected a number of properties. They then send this selection to Weave (searching by third party system) which updates the current selection for properties. They then perform a spatial intersection operation in Weave to refine the selection of properties to only those that fall within a flood zone (searching via spatial operation), then perform an attribute search to further refine the selected properties to those that are listed in the residents database as being over 65 years old (searching via SQL). Then they display the textual list of owners of the properties (reporting from SQL) before finally generating a flood warning mail merge based on the final list of selected properties (reporting from document) and send those to the residents advising them to evacuate.


To facilitate this Weave uses the concept of a current selection, where each entity you want to work with in Weave is associated with a list of unique identifiers that represent the entities that the user has searched for and is reporting on. This list of identifiers can come from anywhere and as long as each system Weave connects to uses the same identifiers to represent the same entity then Weave can work with those systems to perform searching and reporting.

In summary, a central principal in Weave is the entity, which is nothing more than an internal id and a user facing label. A selection is simply a list of identifiers that belong to an entity.

## Spatial Operation

Before spatial operations (select by polygon, zoom to on a map, etc) can be performed against an entity, a spatial engine must be configured telling Weave of the relationship between the layer/table and the entity and a table column (the key column) must be specified to uniquely identify items under each entity. This is done by creating a [spatial mapping configuration](#).

Once a spatial mapping is in place, spatial operations such as select by polygon, boundary intersection and buffer operations result in key column values being written to the current selection under the relevant entity. Conversely, zoom to current selection and highlight selection on a map causes the unique identifiers in the current selection to be used as the basis for performing a query in the spatial engine for a geometry that matches the list of id's.

 It is not always required to specify a spatial mapping to work with an entity in Weave, but this is necessary to display the entities' location on the map or select the entities using spatial operations. It's in fact possible to setup an entire Weave client without a map or any spatial entities and have the user work purely with the textual attributes of the entities.

The spatial operations on the client are represented with the "select/reselect/unselect by point/line/polygon/circle/rectangle" tools, along with various buffer and spatial intersection tools and setup using the [Spatial Mapper](#) configuration item.

## Attribute Display


If you want to display textual attributes for an entity you create data definitions and link those to an entity for display with a data tag. Data definitions are a tabular representation of textual data that can come from any source, but generally will come from a database and be generated by executing SQL. They can be used for populating drop down lists, generating content for a BIRT report or most often for direct display of attribute data for the user.

To allow a data definition to be used to display attribute data related to a selected entity the data tag is used to link an entity to a data definition and provide a label for displaying to the user. This way there can be more than one set of attribute data that can be displayed to the user for each entity and the user can choose between which attributes to display by selecting the from list of available data's based on the label.

For database/SQL based data definitions which will be used for displaying textual data relating to an entity for the user the data definition must be configured with a key column. In the same way that the key column in the spatial mapping links the currently selected entity to a row in a table in the spatial engine, the key column in a data definition links the currently selected entity to a row in a table in a data source (as data source is a connection to a database). This way when it comes time to display the attributes for a selected entity to the user an SQL statement can be generated that only selects the rows that match the id's from the current selection.

Basing the generation of the attribute data on the unique identifiers related to an entity means that the attribute data to be displayed to the user can come from a number of different databases, including the database underlying the spatial engine (if it has one), and does not require that the data that's displayed to the user is even in the same database as the spatial layers, let alone is directly attached to the spatial records in the spatial database.

Attribute display on the client is represented using the data grid view and created using the [Data Definition](#) configuration items.

 The spatial identify tool is a shortcut between the previous two items, the spatial mapping and the data definitions, that bypasses the updating of the selection tables and simply takes the list of identifiers returned by the select by polygon operation and uses that list to select the record from the data definition to display back to the user.

## Attribute Searches

Data definitions take care of generating textual attribute data for displaying to the user, attribute searches provide the reverse. They allow the user to enter text values which are sent to a database, via an SQL statement, to generate a list of unique id's that match the user entered values, which are then used to update the current selection for the entity that the search is associated with. This is again done by specifying a key column for the search, and it's this column whose values will be extracted and placed into the current selection for the list when a search is performed.

From the above you can see that by ensuring that spatial mappings, data definition and attribute searches all have the same concept of what value represents a given entity it's possible to combine all three, even if the actual source of the values (i.e. A specific column in a database table) isn't the same in all three cases.

Attribute searches are represented on the client using the search view and created using the [Search](#) configuration items.

## Reports

Beyond the basic data display generating a [Report](#) can allow the user to produce a more formatted form of output beyond just the tabular display of the data. The reports can contain data generated by Weave data definitions or can pull in data from any data source available to the BIRT reporting engine.

## Virtual Entities

While it's possible to perform joins when working with data definitions and attribute searches, for example to allow you to list all of the animals registered at a selected property, sometimes you want to work with individual entities that don't have a spatial component.

Continuing our animals example if you wanted to select all of the German Shepherds in a given neighbourhood a naive approach would be to create an attribute search that started with the animal registration table (perhaps selecting all the rows containing the animal code for German Shepherds) then join that table to the property table (assuming the animals table has a foreign key into the property table) allowing us to get the unique property id for the property that the animal is located at. Weave then uses those property id's to update the current selection for properties and now we have a list of all the properties that have registered German Shepherds. But, when we then try and display the information related to the dogs we've just searched for using a data definition that does the reverse of when the search did (that is we start with the property table, selecting just those rows that match the property id's in our current selection, then join that table to the animal registration table to get a list of the animals for each property) we see that there are dogs other than German Shepherds listed.

This occurs because what we've done is to select all of the *properties* that have German Shepherds, we haven't selected all of the *German Shepherds* which is what we actually wanted to do. So when it comes time to display the animal information we're displaying information about all of the animals that are registered at each selected property, which will more than likely include more than just the original German Shepherds.

If you want to work with entities that don't have a spatial representation you have two choices, either the user works with the entities using just attribute searches and data definition and has no way of selecting or viewing the entities via the map display, or you create a spatial view for the non-spatial entity and create a new entity to represent the newly "spatialized" entity. For example with our animals search we create a spatial view in the underlying spatial database that joined the animals registration table to the property spatial table, allowing us to represent each animal record by the property boundary of the property it's registered at and then we create a new animals entity and set up a spatial mapping using the spatial view and setting the key to be the animals registration number rather than the property id.

Then when we create an attribute search to find the German Shepherds the list of id's we end up with are the dogs registration numbers, which are unique for each dog, rather than the property id which isn't unique for each dog. We'd also need to switch out attribute search and data definition so that they now use the animal registration number of the key field rather than the property id. Of course we can also keep the animal property attribute search for time when we do want to locate properties based on the animals that are registered there.

Note that we could also add the newly created animals layer to our map engine as a new layer so that we can display just the animals using their property boundaries on the map, but this isn't necessary.

About OSGi

The Weave server utilises OSGi (Open Services Gateway Initiative) to provide its internal infrastructure. The specific OSGi framework being used with the default Weave installation is the Eclipse Equinox implementation. In this section we attempt to explain "why OSGi?"

#### The Problem

Software complexity is increasing at an alarming rate. Today, a large part of this complexity is caused by shortened product cycles, requirements for drastically increased functionality, and an increasing number of variations of the same product (e.g. different hardware and operating systems). These trends have caused software costs to become a larger percentage of almost any manufacturer's development cost.

For an organisation it is now typical for software development to largely consist of adapting existing functionality to perform in a new environment. In the last 20 years, a large number of standard building blocks have become available and they are heavily used in today's products. A prime example is the success of open software. However, the use of these libraries is not without problems. Integrating many different libraries can be daunting because many libraries have become complex and require their own libraries to function, even if that functionality is never needed for the product. This trend requires monolithic software products to undergo a heavy testing cycle. Add unsynchronized evolution of the different libraries and it becomes clearer why software development is so costly today.

A key issue is that today's software environments focus on writing new software, instead of integrating existing software into new systems. In reality, integrating existing code has become a large part of the work of software developers. Therefore, there is a need for tools that standardise the integration aspects of software so that reusing existing components becomes reliable, robust and cheap.

OSGi technology is the dynamic module system for Java. The OSGi Service Platform provides functionality to Java that makes Java the premier environment for software integration and thus for development. In turn, Java provides the portability that is required to support products on many different platforms. The OSGi technology provides the standardised primitives that allow applications to be constructed from small, reusable and collaborative components. These components can be composed into an application and deployed.

The OSGi Service Platform provides the functions to change the composition dynamically on the device of a variety of networks, without requiring restarts. To minimise the coupling, as well as make these couplings managed, the OSGi technology provides a service-oriented architecture (SOA) that enables these components to dynamically discover each other for collaboration. The OSGi Alliance has developed many standard component interfaces for common functions like HTTP servers, configuration, logging, security, user administration, XML and many more.

OSGi technology enables improved time-to-market and reduced development costs because OSGi technology provides for the integration of pre-built and pre-tested component subsystems. The technology also reduces maintenance costs and enables unique new aftermarket opportunities because components can be dynamically delivered to devices in the field.

The OSGi specifications are so widely applicable because the platform is a small layer that allows multiple Java based components to efficiently cooperate in a single Java Virtual Machine (JVM). It provides an extensive security model so that components can run in a shielded environment. However, with the proper permissions, components can reuse and cooperate, unlike other Java application environments. The OSGi Framework provides an extensive array of mechanisms to make this cooperation possible and secure.

The presence of OSGi technology-based middleware in many different industries creates a large software market for OSGi software components. The rigid definition of the OSGi Service Platform enables components that can run on a variety of devices, from very small to very big. Adoption of the OSGi specifications can reduce software development costs as well as provide new business opportunities to expand Weave in places never previously thought.

#### Weave and OSGi

Weave is built using the Eclipse Framework. The Eclipse Framework is an implementation of the specification set out by the OSGi alliance. Eclipse moved to the OSGi at version 3.0 to help consolidate their plugin architecture and help future proof the underlying system. In doing so, a sub project called Equinox was created which deals only with implementing the OSGi specification.

The Equinox Framework also provides utilities and helper classes that enable Weave to easily embed other frameworks and systems on top of it. An example of this is Business Intelligence Reporting Tool (BIRT). Using a plugin approach enables Weave to be a truly modular application made up of smaller parts that contribute to make the entire system.

Embedding BIRT inside Weave was made easier due to the fact that BIRT itself is built on top of Eclipse and as we have mentioned, Eclipse is built on top of OSGi. A more generic example of embedding a third party system into Weave is allowing the ActiveMQ to be run as a service within Weave.

ActiveMQ is an open source implementation of the JMS (Java Messaging Service) specification. ActiveMQ allows for both synchronous and asynchronous communications to be sent and received from a variety of sources. These messages can be sent via a variety of protocols, e.g. TCP, STOMP, etc. Weave was able to use the embeddable nature of ActiveMQ to create a bundle to start ActiveMQ on a defined port, and have other bundles listen for messages via the way defined in the JMS specification. The main usage for this has been to asynchronously update vehicle locations on a map without the need to refresh the map.

**i** From now on the terms plugin and bundle are interchangeable. The OSGi specification uses the term bundle and Eclipse uses the term plugin which was used before Eclipse adopted OSGi.

#### Getting Started

#### Basic Functions

**Show Active Map Tool**

Weave currently displays the active tool as a depressed button. It follows the standard user interface conventions that most desktop applications adhere to.

**Zoom (in/out X2, full, multiple previous, selected features)**

Weave provides the ability to zoom in and out using the standard rubber banding method. Zooming in and out by a fixed amount is also supported. Zooming to the previous extent is supported and is designed to emulate the browser back button. When an extent is changed the previous extent is pushed onto a stack. If the user selects the Previous Extent button the extent at the top of the stack is read and rendered on the map. The extent prior to moving to the previous extent is pushed onto the Next Extent stack and is thus enabled to move to at any stage. Weave also provides the ability to navigate around the map using the arrow keys and the plus and minus keys to zoom in and out.

**Zoom to scale (preset or free text)**

Weave provides the functionality out of the box to allow the user to select from a list of predefined scales or allow the user to type in a given scale. Depending on the configuration of the client, the map interface may also be restricted to a series of predefined scales (i.e. Fixed Scales). This provides the user with a familiar interface which resembles Google Maps and Virtual Earth styled browser applications. In this mode the user can select from a list of predefined scales. When predefined scales are configured, the free text mode where the user can zoom to any scale is disabled. This type of setup works well for internet facing Weave clients which serve cached maps.

**Zoom to Named Area/Bookmark (e.g. hardcoded list with extents or pre-configured spatial query)**

Weave can be configured to have an entity which has a search containing a list of locations taken from a database. The administrator can create a data definition and the results of that would populate a combo box component on the client.

Pre-configured bookmarks for Users and or Groups are also available via the 'Bookmark Tool'.

**Pan (drag as well as map border arrows)**

Weave provides the standard method of panning out of the box whereby the user selects the pan tool and then drags the map, moving it to the desired location. Another method of panning that is available is for the user to click on one of the 4 arrow button at the top of the zoom slider. This option is dependant on the user being given access to the map control.

**Locality Map (includes navigation)**

An overview window may be configured in the client. In the example shown here it is placed in the bottom right corner of the Map View. When hidden, a small arrow is used to indicate where to click and a panel containing the overview map then pops out. The map that is used for the overview map is configurable and can be either a map setup using the Weave mapengine or it may reference a WMS map service that bypasses Weave altogether. The Overview map adjusts its extent as the user changes extents so as to give a better context to the user when looking at datasets that cover a large area.

A red rectangle is rendered on top of the Map to indicate the current map window extent. The user may change the extent of the Map View by either a single click at a desired location on the overview map, or by dragging the rectangle to a new location, at which point the Map View and Locality Map will then update.

**Set active layer for Identify**

The active layer for an Identify is set using the Weave Active Entity combo box. All selections or operations also involve setting the active entity.

**Identify with/without popups (active layer)**

The standard identify is displayed within a 'html div' that avoids creating new browser windows. The corresponding internal window can then be dragged around the screen allowing the user to position the window in the desired location. It is also possible for the user to minimise the window, thus enabling the ability to view data or maps behind the window.

Weave does not popup the identify dialog in a new window. As all modern browsers now block popups it is desirable not to require the user to unblock specific urls

The Identify tool works on the active entity and will not work on other entities unless the active entity is changed. Changing the active entity does not require that the user re-execute the identify operation, the data is automatically requested for the new data definition using the current shape.

**Drill Identify With/Without Popups (all Visible Layers)**

Weave out of the box provides a spatial identify button that enables the user to identify and drill down through all entities visible to the user. This provides the user rapid access to aspatial data for a range of entities.

**Scale Bar**

To aid the user in interpreting distances and areas on the map display, a dynamic scalebar is provided that has the ability to have its display units adjusted interactively by the user if required (e.g. feet, metres, kilometres, miles)

**Show Current Mouse Co-ordinates**



Weave does not make use of the Browser status bar for mouse coordinates as it may either be hidden by the end user or not available in different browsers. Instead, Weave provides a Map Control that displays the current map coordinates (in the configured projection) directly on the map. The location of the Map Control is set during configuration of the client.

### Show Current Mouse Co-ordinates (Projections)

Weave also has the ability to show the current mouse position in various configured Map Projections. The projected values update as the mouse is moved across the map display.

In the above example the client shows four coordinate systems simultaneously. Another example may be where the user is viewing an area on the edge of an MGA zone and requires the two adjacent MGA Zones to be displayed. The number of, and type of projections displayed for the current mouse location are set during configuration of the client.

### Display Tabular Presentation of Data

By default, Weave displays aspatial data as a tabular report (Grid). Multiple database queries can be setup by the administrator and access given to different Users and Roles. The grid employs a paging interface where only a subset of information is ever retrieved from the server (e.g. 10 records at a time). Other features of the Grid are

1. Filter elements in that grid
2. Remote Sorting
3. Zoom to selected feature(s)
4. Pan to selected features(s)
5. Refine the selection based on the selected feature(s) in the grid
6. Remove the selected feature(s) in the grid from the selection set
7. Reorder the columns
8. Hide or show particular columns

### Online Help

Weave has a fully customisable Help system that allows a site to completely change the content of the online help in order to suit the needs of that site. Using standard HTML and CSS (JavaScript can also be used if required) a site can add custom HTML to the help bundle. On the left side of the window is a tree which allows a site to group related help items. The help tool can also be configured to make use of images, and can link to other documents or systems such as voice or video recordings.

### Spatial Bookmarks

Weave can save bookmarks for registered users. This enables bookmarks to be configured for particular users or groups and a user may also create and persist bookmarks of their own. In creating a new bookmark, the current Map View extent is used. An anonymous user may use pre-configured bookmarks and create bookmarks of their own however their custom bookmarks will not be persisted.

The user may use any name for the bookmark (e.g. 'Wind Farm Boundary'). If the user defaults the name, then the system applies the name in the format Bookmark-'n'

### Measure distance and area

Weave provides the ability to measure areas and polylines. Weave supports interactive segment lengths, segment bearings and total areas.

### Table Of Contents

The Weave client can be configured to display a table of contents. Some aspects of the Table of Contents are:

1. **{\*}Turn layers on/off{\*}**: The Table of Contents provides the ability to turn on and off layers
2. **{\*}Turn labels on/off for layer{\*}** : The table of contents provides for a label entry in the Table of Contents under the required layer. That way it is a child and directly related to the layer and may be turned on or off, independent of the parent layer.
3. **{\*}Group multiple physical layers{\*}**: The Table of Contents View within Weave provides the ability to group multiple layers into a single virtual layer. (e.g. Roads: Major Roads, Minor Roads)
4. **Create folders for layers**: Layers can be grouped into folders as well as having the Group used as a checkbox to control the children beneath it. The table of contents supports the cascading of layers and may have multiple levels of folders within folders.

### View Metadata (based on URL with ID)

The Weave client has support for metadata via a right click menu option in the Table of Contents.

### Ability to resize the map and browser

The map will automatically resize itself if the browser window changes size or a View is hidden or shown that changes the size of the Map View.

### Timeouts for process > x seconds

By default, on the client side, all Post requests to the server have a timeout of 60 seconds and various other timeouts are available on other configuration entries.

### Ability to configure the number of returned features.

The map engine has the ability to limit the number of selected features. As Weave renders its own selections, this process is across all map engine implementations.

#### Legends

There are a great many map engines that Weave can use. Each map engine may have its own implementation to generate a legend and not all of these map engines support generating legends in which case a static legend is created and configured for the client.

#### Search Functions

##### Quick Search

Using a standard Weave search it is possible for the user to select any number of named features. The Search is setup at an administrator's level providing combo boxes which help the user find the information quickly and easily.

##### Multi Parameter Search

This is standard Weave functionality where a Search is created by the administrator to allow users to select various parameters that will execute a search of the server. The selected entities can then be zoomed to and highlighted.

Subcategories can be emulated using cascading list boxes where, as the parent list box is updated, the linked child list is refined to only show values that belong to the parent.

##### Dynamic List of attributes in pulldown menu for queries.

Using the standard Weave data definition, the administrator is easily able to setup both dynamic and static combo boxes.

1. Database data definitions are created by using the standard Weave xml configuration defining a 'Distinct' query that will retrieve all values from a database table(s).
2. A static data definition allows the administrator to create a virtual table mirroring an actual table in a database. Values can be updated on the fly.

##### Type Ahead in Search Entry

The existing combo box allows users to type the start of the word to refine the list of values that the user can select from or enter.

##### Search all datasets - what data is available at a point, line or polygon? Including integration with aspatial attributes

See item in "Basic Map Functions" regarding Spatial Identify.

#### Selection functions

##### Set active layer for select

The active layer (or entity) in Weave is set using the Active Entity combo box. Icons to the left of the Entity label indicate whether it is a simple database entity or a spatial entity backed by a spatial engine.

The entity name may be entered via the keyboard using the type ahead capability, or point and click with the mouse to choose an entity from the drop down list. The number of elements that are currently selected is also provided (shown in brackets) next to each entity in the combo box.

##### Select features using radius from point

The Weave client has the ability to select a set of features by drawing a circle interactively on the map.

##### Select features using a point, line, box and polygon

The Weave client supports using a point, circle, polyline, polygon and rectangle to interactively select, add to selection, and remove from selection, features on the map.

##### Select features based on a query

Weave uses the term Search instead of query and provides a rich set of functionality to search for features. The Administrator has full control over what entity is searchable, what searches have been defined for the entity and who will see the searches.

The following attribute types are supported for searches:

1. String
2. Numeric
3. List (either from a database query or hard coded values)
4. Date
5. Time
6. Date/Time

##### Select features based on a query and filter

Weave allows a user to query and only show those features that are returned from the query. Weave controls the rendering of the selection itself.

#### **Select features based on the selection of another layer**

Currently Weave has two methods of selecting features based on a selection in another entity.

1. Using the spatial search whereby the administrator sets up a Search that allows the user to select an attribute from a particular layer, the bounds of the extent of the feature is then intersected with the features in the current entity.
2. Using the buffer tool, setting a distance of 0 and selecting a different entity in the combo box would select features that intersect the features in the entity selected in the combo box.

#### **Select features based on a buffer/distance from currently selected feature**

The buffer tool gives the user the ability to select features in the current entity or intersect features in another entity by placing a value in the distance parameter textbox. The selection is then updated drawing the outline of the buffer selected and refreshing the map with the selected features.

#### **Display simple report based on configured fields and alias names**

Weave provides the Grid action to allow the user to view Grid (or report) based data on the client. The Administrator can define the columns to be presented and apply aliases to those columns.

#### **Filter on Selected Features**

The ability to refine and or filter selected features is provided by Weave via the Map window or Data Grid.

#### **Clear Selections and Markup**

Clearing the selected features in either the active entity or all entities is done as an independent operation. Weave also allows the user to delete all, or a single Markup element using the redlining toolbar.

#### **Ability to select large feature datasets**

As Weave uses paging on the server when requesting attribute data, limiting the number of results returned is not an issue. Even with a very large dataset and large selection, raw database performance is excellent. It may sometimes be required that the selection set when rendering the selection on the map be limited to certain values. Within the MapEngine configuration it is possible for the administrator to set the maximum value to be rendered by Weave.

#### **Reporting Functions**

Weave is integrated with, and provides the BIRT system as its default reporting tools.

The Report Engine of Weave is designed to be as generic as possible. BIRT is an implementation of the Reporting Engine and adds value to the BIRT UI designer by adding a map component and an ability to reuse data definitions created in Weave.

BIRT is an Eclipse-based open source reporting system for web applications, especially those based on Java and J2EE.

BIRT has two main components:

1. a report designer based on Eclipse (see below), and
2. a runtime component that you can add to an app server. BIRT also offers a charting engine that lets you add charts to your own application.

The BIRT Project currently supports two releases 2.2.2 (Included in many commercial applications) and 2.3.2 (Latest).

BIRT enables a rich variety of reports to be added to or used by Weave. Some of the capabilities are as follows:

1. Lists: The simplest reports are lists of data. As the lists get longer, grouping can be added to organise related data together (orders grouped by customer, products grouped by supplier). If the data is numeric, then totals, averages and other summaries can be added.
2. Charts: Numeric data is much easier to understand when presented as a chart. BIRT provides pie charts, line & bar charts and many more. BIRT charts can be rendered in SVG and support events to allow user interaction.
3. Crosstabs: Crosstabs (also called a cross-tabulation or matrix) shows data in two dimensions: e.g. sales per quarter or hits per web page.
4. Letters & Documents Notices, form letters, and other textual documents are easy to create with BIRT. Documents can include text, formatting, lists, charts and more.
5. Compound Reports: Many reports need to combine the above types into a single document. For example, a customer statement may list the information for the customer, provide text about current promotions, and provide side-by-side lists of payments and charges. A financial report may include disclaimers, charts, tables all with extensive formatting that matches corporate color schemes. BIRT reports consist of four main parts: data, data transforms, business logic and presentation.
6. Data: Databases, web services, Java objects all can supply data to a BIRT report. BIRT provides JDBC, XML, Web Services, and Flat File support, as well as support for using code to get at other sources of data. BIRT's use of the Open Data Access (ODA) framework allows anyone to build new UI and runtime support for any kind of tabular data. Further, a single report can include data from any number of data sources. BIRT also supplies a feature that allows disparate data sources to be combined using inner and outer joins.

7. Data Transforms: Reports present data sorted, summarised, filtered and grouped to fit the user's needs. While databases can do some of this work, BIRT must do it for "simple" data sources such as flat files or Java objects. BIRT allows sophisticated operations such as grouping on sums, percentages of overall totals and more.
8. Business Logic: Real-world data is seldom structured exactly as one would like for a report. Many reports require business-specific logic to convert raw data into information useful for the user. If the logic is just for the report, it can be scripted using BIRT's JavaScript support. If an application already contains the logic, it can be called into existing Java code.
9. Presentation: Once the data is ready, a wide range of options for presenting it to the user are provided that include tables, charts (see following example), text and more. A single data set can appear in multiple ways, and a single report can present data from multiple data sets.

### **Extended Map Functions**

#### **Complex Markup ability**

Weave has the ability to add markup to the current map being used. It supports the following features:

1. adding points, polylines and polygons
2. attaching text to each of the above
3. modifying the vertices of a geometry element
4. rotating a geometry element
5. resizing a geometry element
6. changing the fill and line style of a geometry element
7. changing the opacity of a geometry element
8. changing the weight and style of the lines of a geometry element
9. deleting either a selected geometry element or all geometries in the redline layer.

#### **Spatial Analysis.**

The following functions are supported.

1. Intersection of selected features in another entity
2. Buffer selected features in the current entity
3. Buffer features in current layer using the selected features in another entity

#### **Export Layer(s) to Shapefile**

Weave supports exporting the current selection of an entity or entities out to a Shapefile using the 'Zip and Ship' tool. This provides a rapid spatial data extraction and export capability. The resulting zip file can be opened or saved at the user desktop.

#### **Extraction of selected features and/or associated data**

Weave supports the ability to extract the current selected features from a data definition into HTML or CSV format. The function is a plugin that can be extended to any type of file format.

#### **Produce plots - based on simple and complex templates**

Plots are produced as part of the built-in reporting capability. Maps are embedded on a report 'page' and linked to the current map window. Vector, and raster representations are supported and map reports may make use of parameters and scripts to provide override options to default map templates.

Both tabular and map output can be included in the production of written and printed reports. The reports may simply include the content (raster and vector) of the current map display and/or it may include predefined map layers that are mandated for inclusion in the report. In addition the report can be configured to generate a zoomed in map for each of the selected spatial entities.

The reports can be configured to use templates that provide appropriate marginal information to the map along with mandatory departmental clauses and statements. A wide variety of output formats are supported.

#### **Include WMS layer(s)**

Weave has the ability to add multiple WMS Services and WMS Layers to a Weave client. The current versions of WMS and 1.1 and 1.3 of the OGC WMS Specification are supported.

#### **Apply security to tools, reports, data, print templates.**

Weave provides Role Based Access Control. Users can be restricted to spatial and aspatial datasets via a client that is configured for a particular User or Group of users. The Weave administrator has the ability to configure Roles and apply them to different user interfaces, reports, entities, and data, etc.

#### **Integration with Google Earth**

Weave has a Google Earth plugin that allows the draping of a WMS service over the Google Earth service.

#### **Natural Language Geocoding**

Organisations today fight an ever increasing problem and that is information overload. How do you enable users to find correct information quickly, easily and in a familiar way?

A core capability of Weave that is proving to be of significant value to Weave users is the Weave Natural Language Geocoding Engine.

This capability is quite unique to the Weave Business Integration Framework and Cohga has found that sites such as Melbourne City Council, Melbourne Fire Brigade, and North shore City Council for example are already gaining in productivity through the use of this capability.

The Natural Language Geocoding Engine provides a more intuitive method for a user to query or search for useful data that may reside in a multitude of systems at a user site without the training and experience of knowing where or how to gain access to that data.

Typical Geocoding engines rely on well defined structured data to allow geocoding to be undertaken on a dataset. e.g. House Number, Street, Street Type, Suburb and PostCode. There are several disadvantages that arise with this type of geocoding:

1. What if your data is not structured in this way?
2. What if you do not know the structure of the data and it is just unformatted text?
3. What if you are not interested in Address information.
4. What if you need to search across multiple datasets each with a different structure or from a different GIS dataset.

Typical geocoding engines require that the user types the Address in it's full format. e.g. "4 Smith Street, Yarraville"

What if the user is unsure about the street name? It would be nice to allow the user to start with "Yarraville" then "Smith" then "4".

Note that the word street was not entered nor is a delimiter required to distinguish the suburb.

Weave provides the tools to harvest and create indexes of user data and it is through these linked indexes that a user may search their data using free format queries and be provided with dynamic feedback of results as the query is being entered.

The Natural Language Geocoding Engine has the ability to rapidly search through textual based information much faster than Database Queries. It employs an indexing scheme to pre-index the textual and spatial data associated with a feature and store it in a binary format on the disk. Generally an index is ~1/3 the size of the original dataset, however this depends on the amount of information contained within the index. There are two steps to working with the Natural Language Geocoding Engine and these are harvesting, and searching .

The data harvesting and indexing process is totally separate to the interactive searching of the index where the user enters a query. Searching can also be performed on the indexes while they are being updated. Once the index is updated, the new version is used on the next search. This capability allows for the service to remain up for long periods. Weave also has the capability to schedule the updating of indexes so that the indexes are maintained automatically.

Searches can be done across all indexes with merged results or specific ones with or without the user needing to know neither where the data resides nor how to gain access to it. Weave provides ranked searching which enables the best results to be returned first.

The indexing process is rapid with current users able to index more than 3 million features in approximately 9 minutes. This indexing can be performed as a background, batch or incremental process. Incremental harvesting is as fast as batch harvesting and index size is roughly 20% to 30% the size of data being harvested

By making use of the Natural Language Geocoding Engine that is provided with Weave, a user site can expect a typical search to take between 3 to 30ms for more than 3 million records to be searched.

Like all data displays in Weave, search results can be requested in pages to help minimise bandwidth. i.e. only get 10 or 20 results at a time.

In summary, some key aspects of the Weave search capability are as follows:

1. ranked searching with best results returned first
2. many powerful query types: phrase queries, wildcard queries, proximity queries, range queries and more
3. field based searching (e.g. title, author, contents)
4. date-range searching
5. sorting by any field
6. multiple-index searching with merged results
7. simultaneous index update and searching

## Configuration

The Weave configuration is stored in an XML file, `config.xml`, and it describes the items that Weave should use to do things like connecting to databases, laying out a Table of Contents or defining access control.

The basic configuration file contains a standard XML header, a root `config` tag that defines the namespaces for the items that will be defined within the config file, and the tags within the root config tag that define all of the configuration items that setup the Weave server for operation.

The following example shows the bare minimum require of the `config.xml` file, including the xml header, the root config tag and the setting of the default namespace.

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0">
  <!-- configuration items defined here -->
</config>
```

**⚠ Character Encoding**

The xml header shown above defines a default character encoding for the file as UTF-8, which is indicating that the config.xml file is actually saved using the UTF-8 character encoding. This is important if you're using non-latin characters in your config.xml file, for example those with umlauts, since they're represented differently depending upon what character encoding the file is saved in. If you're using a different character encoding, for example "ISO-8859-1" or "windows-1252" then the encoding should be set to that in the xml header to ensure that Weave reads the file using the correct character encoding.

The configuration of the Weave server involves adding configuration items to this basic config file until all the parts that are required to build a running systems are in place.

Before we can start adding configuration items to the config.xml file we need to familiarize ourselves with namespaces, which is used to join configuration items to the server components that are responsible for implementing them.

**Namespaces**

The Weave configuration file uses namespaces to provide flexibility in the configuration. Rather than having a single monolithic pre-defined configuration format the Weave server uses plugins to process different parts of the configuration file and the namespaces are used to indicate to the server which plugins should be used to process which parts. In this way new components can be added to the system and can include their own configuration options in config.xml without having to make any changes to the core Weave infrastructure.

The namespaces for internal Weave plugins follow the format `urn:pluginid#version`, but there are situations where other namespaces can be used in the config file for example when referencing XMLSchema elements.

The configuration file outlined above is of little use in its current form as it doesn't define any configuration items for the server to use so let's expand the configuration to include an example configuration item.

The first thing that we need to do is add a namespace for the plugin that will be processing our new configuration item. For our example we will use a dummy plugin with the id `com.cohga.weave.example`. We will be using version 1.0 so the namespace we will need to add is `urn:com.cohga.weave.example#1.0`

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:example="urn:
com.cohga.weave.example#1.0">
</config>
```

What we have done here is to create a namespace place holder called `example`, that is used later in the configuration file to reference the actual namespace, `urn:com.cohga.weave.example#1.0`.

Adding the namespace is not enough to create a new configuration item. It simply tells the configuration file reader that anything referencing the namespace should be handled by the particular plugin, but more on that later.

For now we'll use the namespace to create the new configuration item. Normally this is where you would consult the [Configuration Reference](#) to find out the structure/content of the item, since each plugin has its own requirements for what needs to be included within it.

For this example we will we'll just make something up and add the item.

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:example="urn:
com.cohga.weave.example#1.0">

  <example:greeting id="greeting.english">
    <text>Hello World</text>
  </example:greeting>

</config>
```

Here we've created an example configuration item.

Now we have a working configuration file that will create an example item when the server starts. But what is actually happening here?

## Parsing

The important things to note in our example are the use of the namespace to enclose the configuration of this particular item, the setting of the configuration item type (in this example it's *greeting*), the unique identifier given to the item (*greeting.english*), and the actual content of the configuration item.

The id, in this example (*greeting.english*), is used to uniquely identify individual configuration items, and should be unique for each combination of namespace and type. That is you can have the same id for an *example:greeting* configuration item, an *example:message* configuration item and a *test:greeting* configuration item, but you can have the same id for two *example:greeting* configuration items.

The item needs to be uniquely identified for two reasons, the first one is that if the content of the configuration item is changed the configuration file reader can notify the system that this particular item has been changed, the other reason is so that the item can be referenced by other configuration items.

Not all items actually require an id however, if there will only ever be one instance of the particular object that the configuration item is supposed to represent, and the item never needs to be referenced by other configuration items, then it may be possible to not use the id. An example of this would be the configuration of a mail server. Since there is only a single mail server required for Weave to send email and no other configuration item will be required to reference that particular configuration, then only one configuration item is required.

When the configuration file reader parses the config.xml file it looks at all of the top level items (i.e. those immediately below the root config tag), and determines the namespace for those items. For each of these items it locates a plugin registered against that namespace and passed the entire contents of the configuration item to the plugin, along with the version number used in the namespace definition and the configuration item type.

It is up to the plugin to examine the item passed to it to extract the information that it requires. In the above example, the plugin will extract the id, *greeting.english*, and the text, "Hello World", and do whatever is appropriate for the plugin.

Other configuration items can refer to this particular greeting, using its id, and the system will link the two items at runtime.

In this manner, adding required namespaces and configuration items, the configuration file is built up to compose a working Weave server using only those components that the server requires.

The [Configuration Reference](#) details what core components are available for you to use as part of your configuration and what the appropriate contents should be for each.

## Processing Instructions

When processing the configuration file, Weave provides a couple of processing instructions that may help to make your life a little easier. They are `include` and `set`. These instructions provide a means to alter the configuration items before they are processed.

The [Processing Instructions](#) page outlines the use of these instructions.

## Null Values

Occasionally there are times when you need to un-set a value, you do this using a null.

The [Null Values](#) page outlines how to include 'null' in your configuration and why you would do this.

### Configuration Null Values

Within a configuration item you can set a value to explicitly be `null`. What that means in terms of configuration is dependent upon the context of where it is set.

To set a value to `null` you need to add a special namespace to the `config` tag, <http://www.w3.org/2001/XMLSchema-instance>, and then use that to set the value `nil` to `true` for the value you want to set to `null`.

So in the example below the `overflowToDisk` value will explicitly set to null

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:xsi="http://www.
w3.org/2001/XMLSchema-instance" ...

    <data:datadefinition id="...">
        <datasourcedataconnection datasource="..."
table="..." key="...">
            ...
            <cache id="existing.cache.config">
                <overflowToDisk xsi:nil="true"/>
            </cache>
```

```

        </datasourcedataconnection>
    </data:datadefinition>

</config>

```

Initially you may think that setting the value to `null` would appear to be the same thing as not setting the value at all, but as the example above shows what we're doing with the `cache` tag is to override the settings in the cache definition, which is defined elsewhere, for this particular `datasourcedataconnection`.

So in this example the `datasourcedataconnection` will use the cache definition supplied by `existing.cache.config` but with the `overflowToDisk` value reset to its default. The default will be used because that's the normal handling for a cache definition where a value isn't set. Other configuration items may have different interpretations of what `null` means for a specific value and this is outlined in the configuration reference for that item.

### Configuration Processing Instructions

When processing the configuration file, Weave provides a couple of processing instructions that may help to make your life a little easier. They are `include` and `set`. These instructions provide a means to alter the configuration items before they are processed.

### include

When the `include` processing instruction is used, the contents of the file specified are read and inserted into the configuration tree at the location where the `include` tag was specified.

```

<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" ...

    <?include client_main.xml?>

</config>

```

the `client_main.xml` file could contain

```

<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:client="urn:com.cohga.html.client#1.0">

    <client:config id="main">
        <default>true</default>
        <debug>>false</debug>
        <theme>grey</theme>
        ...
    </client:config>

</config>

```

One other thing to note about that last example is that the included `client_main.xml` file uses two `config` tags, but these are different tags because they have different namespaces.

The `include` processing instruction can be particularly useful when moving a development configuration to a production server. The development and production servers can have a different version of the included file(s) but with the main configuration file remaining the same between the two systems.

It could also be used when repeated elements are required. For example, definitions of connection pools, and the same file used with multiple `include` instructions.



The name of the include file can be just the file name in which case the file is searched for in the same directory as the config.xml file. However it could also be a fully qualified file name which specifies an exact location for the file or it can also reference a URL, allowing for the contents to be fetched from another server.

If the include reference points to a file then the system will also monitor the included files for changes.

## set

The `set` instruction can supplement the `include` instruction by providing support for variable substitution within the configuration file. A variation on the previous example can be used when only the hostname needs to change between the development and production servers, so this time config.xml could contain

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0"...

    <jdbc:dataSource id="datasource.main">
        <driver>oracle.jdbc.driver.OracleDriver</driver>
        <url><![CDATA[jdbc:oracle:thin:@${HOSTNAME}:1521:GIS]]
    ></url>

        <username>reader</username>
        <password>0hn0w3rdun</password>
        <pool:pool>
            <maxActive>15</maxActive>
            <minIdle>2</minIdle>
            <maxIdle>2</maxIdle>
            <testOnBorrow>true</testOnBorrow>
            <timeBetweenEvictionRunsMillis>60000<
/ timeBetweenEvictionRunsMillis>
            <minEvictableIdleTimeMillis>1200000<
/ minEvictableIdleTimeMillis>
            <whenExhaustedAction>grow<
/ whenExhaustedAction>
        </pool:pool>
    </jdbc:dataSource>

</config>
```

Then the `datasource.xml` file on the development server could contain

```
<?set HOSTNAME=development.example.com?>
```

and the `datasource.xml` file on the production server could contain

```
<?set HOSTNAME=production.example.com?>
```

The `${HOSTNAME}` portion of config.xml would be replaced with the appropriate hostname. Also, note that the value substituted is taken verbatim as everything following the `=` up to the `?>`, which includes white space and case is important.

Of course `set` can also be used to locate commonly referenced items in a single location at the start of the config.xml file to allow them to be easily changed later. For example passwords that are updated every month or defaults for functions such as connection timeouts.

## Externally defined values

In addition to the set processing instruction, the value for a substitution variable can also come from system properties or from a property as defined during the startup of the Java Virtual Machine (JVM option). `{property_name}` notation can then be used to substitute the values in place.

### Adding custom projections

As of Weave 2.5.28 it's possible to include additional EPSG definitions via an `epsi.properties` file included in the workspace directory.

The file contents should include a single line definition per-projection in the format of `<code>=<definition>`, where `<code>` is the numeric EPSG code and `<definition>` is the OGC WKT projection definition. e.g.

```
3111=PROJCS["GDA94 / Vicgrid94", GEOGCS["GDA94", DATUM["Geocentric Datum of Australia 1994", SPHEROID["GRS 1980", 6378137.0, 298.257222101, AUTHORITY["EPSG","7019"]], TOWGS84[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], AUTHORITY["EPSG","6283"]], PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]], UNIT["degree", 0.017453292519943295], AXIS["Geodetic longitude", EAST], AXIS["Geodetic latitude", NORTH], AUTHORITY["EPSG","4283"]], PROJECTION["Lambert_Conformal_Conic_2SP", AUTHORITY["EPSG","9802"]], PARAMETER["central_meridian", 145.0], PARAMETER["latitude_of_origin", -37.0], PARAMETER["standard_parallel_1", -36.0], PARAMETER["false_easting", 2500000.0], PARAMETER["false_northing", 2500000.0], PARAMETER["scale_factor", 1.0], PARAMETER["standard_parallel_2", -38.0], UNIT["m", 1.0], AXIS["Easting", EAST], AXIS["Northing", NORTH], AUTHORITY["EPSG","3111"]]
3857=PROJCS["WGS 84 / Pseudo-Mercator", GEOGCS["WGS 84", DATUM["World Geodetic System 1984", SPHEROID["WGS 84", 6378137.0, 298.257223563, AUTHORITY["EPSG","7030"]], AUTHORITY["EPSG","6326"]], PRIMEM["Greenwich", 0.0, AUTHORITY["EPSG","8901"]], UNIT["degree", 0.017453292519943295], AXIS["Geodetic longitude", EAST], AXIS["Geodetic latitude", NORTH], AUTHORITY["EPSG","4326"]], PROJECTION["Popular Visualisation Pseudo Mercator", AUTHORITY["EPSG","1024"]], PARAMETER["semi_minor", 6378137.0], PARAMETER["latitude_of_origin", 0.0], PARAMETER["central_meridian", 0.0], PARAMETER["scale_factor", 1.0], PARAMETER["false_easting", 0.0], PARAMETER["false_northing", 0.0], UNIT["m", 1.0], AXIS["Easting", EAST], AXIS["Northing", NORTH], AUTHORITY["EPSG","3857"]]
```

## Security


- When trouble-shooting security issues it's often useful to confirm who the user is and what, if any, additional information is associated with that user.

Previously this was possible (to a limited extent) by starting a client in [debug mode](#) and viewing the page source, which would contain information about the user in a comment at the bottom of the page. But as of [Weave 2.6.5](#) there's also a link that will return all the available information about the user in a single response, that link is `/weave/whoami`. If you open that page you will see a response that provides all of the details for the user.

If the user is not logged in the page should indicate that the user is an anonymous user but will still provide some additional details, such as their IP address. If the user is logged in it will show their username and roles along with the other details. This can be particularly useful to verify that the roles that are associated with a user are the same as the role names that are being used in an ACL, or that the IP address being returned for a user is actually the users IP address and not something else, for example the IP address of a reverse proxy.

## Overview

The front line for security in Weave is Acegi security (now Spring Security). Acegi security system is a formidable, easy-to-use alternative to writing custom security code for Java enterprise applications. Weave utilises the plugin nature of the Acegi to provide a highly customisable authentication mechanism while minimising the amount of security-specific code.

 Starting with Weave version 2.5, a newer version of Acegi security, known as Spring Security is supported (at the time of this writing, Spring Security version 3.2). For more information on Spring security configuration, refer [Spring Security](#).

## Authentication and Authorisation

Authentication and authorisation are multi-stage processes. Firstly the user must be identified. By default they are an "anonymous" user, but this can be changed by users going to the login page (either directly themselves or being forced to by Weave) or by enabling Windows integrated authentication (which uses their Windows login username). Third-party authentication systems can be configured for this purpose via an Acegi security plugin.

Secondly, we need to determine what Roles the user has, again this can be done a number of ways, one of which is going to Active Directory (AD) and determining what AD Groups the user is a member of. In fact, any user attribute that's associated with a user in AD can be used to determine their role but group membership is the most common.

Thus, determining a user's identity and their roles occur outside of Weave and is handled by Acegi security.

Specifics of Windows authentication [are covered here](#)

## Weave Access Control Lists

Once a user is authenticated and their roles are known, Weave Access Control Lists (ACLs) come in to play. ACLs provide a way of attaching an access decision to an object.

E.g. a) for object A: users with ROLE\_X will be denied access but users with ROLE\_Y will be allowed

b) for object B: anonymous users and users with ROLE\_X will be denied access but everyone else will be allowed access

As well as roles, a user's userid can also be used in an ACL when a specific users need to be targeted, either to remove access or grant it.

Refer to the section on [Access Control List](#) for more information.

## Acegi Security System

The following sections describe details about how Acegi security works internally.

Acegi security system uses security filters to provide authentication and authorisation services to enterprise applications. The framework offers several types of filters that you can configure according to your application requirements.

You can configure security filters for the following tasks:

1. Prompt the user for login before accessing a secure resource.
2. Authenticate the user by checking a security token such as a password.
3. Check whether an authenticated user has the privilege to access a secure resource.
4. Redirect a successfully authenticated and authorised user to the secure resource requested.
5. Display an Access Denied page to a user who does not have the privilege to access a secure resource.
6. Remember a successfully authenticated user on the server and set a secure cookie on the user's client. The next authentication can then be performed using the cookie and without asking the user to log in.
7. Store authentication information in server-side session objects to securely serve subsequent requests for resources.
8. Build and maintain a cache of security information in server-side objects to optimise performance.
9. When the user signs out, destroy server-side objects maintained for the user's secure session.
10. Communicate with a variety of back-end data storage services (like a directory service or a relational database) that are used to store users' security information and the ECM's access-control policies.

As this list suggests, Acegi filters allow you to do almost anything you might require to secure enterprise applications.

## Architecture and components

This section introduces Acegi's components; next, you will learn how the framework uses inversion of control (IOC) and XML configuration files to combine components and express their dependencies.

## The Big Four

Acegi consists of four main types of component: filters, managers, providers, and handlers.

**Filters** - These mostly high-level components provide common security services like authentication processing, session handling, and logout. Filters are only a high-level abstraction of security-related functionality: managers and providers are used to actually implement authentication processing and logout services.

**Managers** - Managers manage lower-level security services offered by different providers.

**Providers** - A variety of providers are available to communicate with different types of data storage services, such as directory services, relational databases, or simple in-memory objects. This means you can store your user base and access control policies in any of these data storage services and Acegi's managers will select appropriate providers at run time.

**Handlers** - Tasks are sometimes broken up into multiple steps with each step performed by a specific handler. For example, Acegi's logout filter uses two handlers to sign out an HTTP client. One handler invalidates a user's HTTP session and another handler destroys the user's cookie. Having multiple handlers provides flexibility when configuring Acegi to work according to your application requirements. You can select the handlers of your choice to execute the steps required to secure your application.

## Inversion of control

Acegi's components are dependent on each other to secure enterprise applications.

For example, an authentication processing filter requires an authentication manager to select an appropriate authentication provider. This means you must be able to express and manage the dependency of Acegi's components.

An IOC implementation is commonly used to manage the dependencies of Java components. IOC offers two important features:

1. It provides a syntax to express which components are required in an application and how they depend on each other.
2. It ensures that the required components are available at run time.

## The XML configuration file

Acegi uses the popular open-source IOC implementation that comes with the Spring framework (see Resources) to manage its components. Spring uses an XML configuration file to express the dependency of components, similar to the following:

```
<beans>
  <bean id="filterChainProxy" class="org.acegisecurity.util.
FilterChainProxy">
    <property name="filterInvocationDefinitionSource">
      <value> value here </value>
    </property>
  </bean>

  <bean id="authenticationProcessingFilter"
class="org.acegisecurity.ui.webapp.
AuthenticationProcessingFitler">
    <property name="authenticationManager" ref="authManager"/>
    <!-- Other properties -->
  </bean>

  <bean id="authManager"
class="org.acegisecurity.providers.ProviderManager">
    <property name="providers">
      <!-- List of providers here -->
    </property>
  </bean>
  <!-- Other bean tags -->
</beans>
```

As you can see, the Spring XML configuration file used by Acegi contains a single <beans> tag that wraps a number of other <bean> tags. All Acegi components (that is, filters, managers, providers, etc.) are actually JavaBeans. Each <bean> tag in the XML configuration file represents a Spring security component.

## Further notes about the XML configuration file

The first thing you will note is that each `<bean>` tag has a `class` attribute that identifies the class that the component uses. The `<bean>` tag also has an `id` attribute that identifies the instance (Java object) acting as a Acegi component.

For instance, the first `<bean>` tag of the above example identifies a component instance named `filterChainProxy` which is an instance of a class named `org.acegisecurity.util.FilterChainProxy`.

The dependency of a bean is expressed using child tags of the `<bean>` tag. For example, notice the `<property>` child tag of the first `<bean>` tag. The `<property>` child tag defines values or other beans on which the `<bean>` tag depends.

So in the example, the `<property>` child tag of the first `<bean>` tag has a `name` attribute and a `<value>` child tag, which respectively define the name and value of the property on which the bean depends. Likewise, the second and third `<bean>` tags of the example define that a filter bean depends on a manager bean. The second `<bean>` tag represents the filter bean and the third `<bean>` tag represents the manager bean.

The `<bean>` tag for the filter contains a `<property>` child tag with two attributes, `name` and `ref`.

The `name` attribute defines a property of the filter bean and the `ref` attribute refers to the instance (name) of the manager bean.

The next section shows you how to configure Acegi filters in an XML configuration file.

## Security filters

As previously mentioned, Acegi uses security filters to provide authentication and authorisation services to enterprise applications. You can use and configure various types of filters according to your application requirements. The following sections introduce the five most important Acegi filters.

## Session Integration Filter

Acegi's Session Integration Filter (SIF) is normally the first filter you will configure.

SIF creates a security context object, which is a placeholder for security-related information.

Other Acegi filters save security information in the security context and also use the information available in the security context.

SIF creates the security context and calls other filters in the filter chain.

Other filters then retrieve the security context and make changes to it.

For example, the Authentication Processing Filter (which is discussed next) stores user information such as username, password, and e-mail address in the security context.

When all the filters have finished processing, SIF checks the security context for updates.

If any filter has made changes to the security context, SIF saves the changes into a server-side session object. If no changes are found in the security context, SIF discards it.

SIF is configured in the XML configuration file as follows:

```
<bean id="httpSessionContextIntegrationFilter"
      class="org.acegisecurity.context.
      HttpSessionContextIntegrationFilter"/>
```

## Authentication Processing Filter

Acegi uses the Authentication Processing Filter (APF) for authentication.

APF uses an authentication (or login) form, in which a user enters a username and password and triggers authentication.

APF performs all back-end authentication processing tasks such as extracting the username and password from the client request, reading the user's parameters from the back-end user base, and using the information to authenticate the user.

When you configure APF, you must provide the following parameters:

- Authentication manager specifies the authentication manager to be used to manage authentication providers.
- Filter processes URL specifies the URL to be accessed when the client presses the Sign In button on the login form. Upon receiving a request for this URL, Spring security invokes APF.
- Default target URL specifies the page to be presented to the user if authentication and authorisation is successful.
- Authentication failure URL specifies the page the user sees if authentication fails.

APF fetches the username, password and other information from the user's request object. It then passes this information to the authentication manager. The authentication manager uses an appropriate provider to read detailed user information (such as username, password, e-mail address and the user's access rights or privileges) from the back-end user base, authenticates the user, and stores the information in an Authentication object.

Finally, APF saves the Authentication object in the security context created earlier by SIF.

The Authentication object stored in the security context will be used later to make authorization decisions.

Configure APF as shown in the following example:

```

<bean id="authenticationProcessingFilter"
      class="org.acegisecurity.ui.webapp.AuthenticationProcessingFilter"
    >
  <property name="authenticationManager"
            ref="authenticationManager" />
  <property name="filterProcessesUrl"
            value="/j_acegi_security_check" />
  <property name="defaultTargetUrl"
            value="/protected/protected1.jsp" />
  <property name="authenticationFailureUrl"
            value="/login.jsp?login_error=1" />
</bean>

```

You can see from this code that APF depends on the four parameters discussed in "The Big Four". Each parameter is configured as a <property> tag in the example.

## Logout Processing Filter

Acegi uses a Logout Processing Filter (LPF) to manage logout processing. LPF operates when a logout request comes from a client. It identifies the logout request from the URL invoked by the client.

LPF is configured as shown in the following example:

```

<bean id="logoutFilter" class="org.acegisecurity.ui.logout.
LogoutFilter">
  <constructor-arg value="/logoutSuccess.jsp" />
  <constructor-arg>
    <list>
      <bean class="org.acegisecurity.ui.logout.
SecurityContextLogoutHandler" />
    </list>
  </constructor-arg>
</bean>

```

You can see that LPF takes two parameters in its constructor: the logout success URL (/logoutSuccess.jsp) and a list of handlers. The logout success URL is used to redirect the client after the logout process is complete. Handlers perform the actual logout process; I have configured only one handler because it is enough to invalidate the HTTP session.

## Exception Translation Filter

The Exception Translation Filter (ETF) handles exceptional cases in the authentication and authorization procedure, such as when authorization fails. In these exceptional cases, ETF decides what to do.

For example, if a non-authenticated user attempts to access a protected resource, ETF serves the login page inviting the user to authenticate.

Similarly, in case of authorization failure, you can configure ETF to serve an Access Denied page.

ETF is configured as shown in the following example:

```

<bean id="exceptionTranslationFilter"
      class="org.acegisecurity.ui.ExceptionTranslationFilter">
  <property name="authenticationEntryPoint">
    <bean
      class="org.acegisecurity.ui.webapp.
AuthenticationProcessingFilterEntryPoint">

```

```

        <property name="loginFormUrl" value="/login.jsp" />
    </bean>
</property>
<property name="accessDeniedHandler">
    <bean class="org.acegisecurity.ui.AccessDeniedHandlerImpl">
        <property name="errorPage" value="/accessDenied.jsp" />
    </bean>
</property>
</bean>

```

As you can see from the above example, ETF takes two parameters named `authenticationEntryPoint` and `accessDeniedHandler`. The `authenticationEntryPoint` property specifies the login page and the `accessDeniedHandler` specifies the Access Denied page.

## Interceptor filters

Acegi's interceptor filters are used to make authorization decisions. You need to configure interceptor filters to act after APF has performed a successful authentication. Interceptors use your application's access control policy to make authorisation decisions.

Later we will show you how to design access control policies, how to host them on a directory service, and how to configure Acegi to read your access control policy. For the moment however, we will stick to showing you how to configure a simple access control policy using Acegi.

You can divide configuring a simple access control policy into two steps:

1. Writing the access control policy.
2. Configuring Acegi's interceptor filter according to the policy.

Step # Writing a simple access control policy

Start by looking at the next example, which shows how to define a user and the user's role:

```
alice=123,ROLE_HEAD_OF_ENGINEERING
```

The access control policy shown above defines a user named `alice`, whose password is `123` and whose role is `ROLE_HEAD_OF_ENGINEERING`.

Step 2. Configuring Acegi's interceptor filter

Interceptor filters use three components to make authorisation decisions, which I have configured in the next example:

```

<bean id="filterInvocationInterceptor"
    class="org.acegisecurity.intercept.web.FilterSecurityInterceptor">
    <property name="authenticationManager" ref="
authenticationManager" />
    <property name="accessDecisionManager" ref="
accessDecisionManager" />
    <property name="objectDefinitionSource">
        <value>
            CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
            PATTERN_TYPE_APACHE_ANT
            /protected/**=ROLE_HEAD_OF_ENGINEERING
            /**=IS_AUTHENTICATED_ANONYMOUSLY
        </value>
    </property>
    <!-- More properties of the interceptor filter -->
</bean>

```

As the example shows, the three components you need to configure are the authenticationManager, accessDecisionManager and objectDefinitionSource:

- The authenticationManager component is the same as the authentication manager discussed when we introduced the Authentication Processing Filter. The interceptor filter can use the authenticationManager to re-authenticate a client during the authorization process.
- The accessDecisionManager component manages the process of authorization, which will be discussed further later.
- The objectDefinitionSource component contains access control definitions according to which authorization will take place. For example, the objectDefinitionSource property in the example contains two URLs (/protected/\* and /) **in its value. The value defines roles for these URLs. The role for the /protected/ URL is ROLE\_HEAD\_OF\_ENGINEERING.** You can define any roles you like, according to your application requirements.

Recall from the previous example that you defined ROLE\_HEAD\_OF\_ENGINEERING for the user named alice. This means alice will be able to access the /protected/\* URL.

### How filters work

As you have learned already, Acegi's components are dependent on each other to secure your applications. Later you will see how to configure Acegi to apply security filters in a specific order, thus creating a chain of filters. For this purpose, Acegi maintains a filter chain object which wraps all the filters you have configured to secure your application.

The following steps describe the life cycle of the filter chain:

1. A browser client sends an HTTP request to your application.
2. The container receives the HTTP request and creates a request object that wraps information contained in the HTTP request. The container also creates a response object, which different filters can process to prepare an HTTP response for the requesting client. The container then invokes Acegi's filter chain proxy, which is a proxy filter. The proxy knows the sequences of actual filters to be applied. When the container invokes the proxy, it passes request, response, and filter chain objects to it.
3. The proxy filter invokes the first filter in the filter chain, passing request, response, and filter chain objects to the filter.
4. Filters in the chain do their processing one by one. A filter can terminate its processing at any time by calling the next filter in the chain. A filter may even choose to not perform any processing at all. For example, APF might terminate its processing upon discovering that an incoming request did not require authentication.
5. When authentication filters have finished processing they pass request and response objects to the interceptor filter configured in your application.
6. The interceptor decides whether the requesting client is authorized to access the requested resource.
7. The interceptor transfers control to your application. For example, the JSP page requested by the client in case of successful authentication and authorization.
8. Your application writes contents over the response object.
9. The response object is now ready. The container translates the response object into an HTTP response and sends the response to the requesting client.

To help you further understand Acegi filters we will give you a closer look at the operation of two of them: the Session Integration Filter and the Authentication Processing Filter.

### How SIF creates a security context

How SIF creates a security context:

1. Acegi's filter chain proxy invokes SIF and passes request, response, and filter chain objects to it. Note that normally you will configure SIF as the first filter in the filter chain.
2. SIF checks whether it has already processed this Web request or not. If it finds that it has, it does no further processing and transfers control to the next filter in the filter chain (see Step 4 below). If SIF finds that this is the first time it has been called during the given Web request, it sets a flag, which will be used next time to indicate that SIF has been called.
3. SIF checks whether a session object exists and contains a security context. It retrieves the security context from the session object and places it in a temporary placeholder called security context holder. If the session object does not exist, SIF creates a new security context and puts it in the security context holder. Note that the security context holder exists in the application scope so that it is accessible to other security filters.
4. SIF calls the next filter in the filter chain.
5. Other filters may edit the security context.
6. SIF receives control after the filter chain processing is complete.
7. SIF checks whether any other filter changed the security context during its processing. For example, APF may have stored user details in the security context. If so, it updates the security context in the session object. This means that any changes made to the security context during filter chain processing now reside in the session object.

### How APF authenticates a user

How APF authenticates a users:

1. The previous filter in the filter chain passes request, response, and filter chain objects to APF.
2. APF creates an authentication token with the username, password, and other information fetched from the request object.
3. APF passes the authentication token to the authentication manager.
4. The authentication manager may contain one or more authentication providers. Each provider supports exactly one type of authentication. The manager checks which of its providers support the authentication token it received from APF.
5. The authentication manager passes the authentication token to the provider suitable for authentication.
6. The authentication provider extracts the username from the authentication token and passes it to a service called user cache service. Acegi maintains a cache of users who have been authenticated. The next time the user signs in, Acegi can load his or her



details (such as username, password, and privileges) from the cache instead of reading from back-end data storage. This improves performance.

7. The user cache service checks whether details of the user exist in the cache.
8. The user cache service returns the details of the user to the authentication provider. If the cache does not contain user details, it returns null.
9. The authentication provider checks whether the cache service returned details of the user or null.
10. If the cache returned null, the authentication provider passes the username (extracted in Step 6) to another service called user details service.
11. The user details service communicates with the back-end data storage (such as a directory service) that contains details of the user.
12. The user details service returns details of the user or throws an authentication exception if it cannot find details of the user.
13. If either the user cache service or the user details service returns valid user details, the authentication provider matches the security token (such as a password) supplied by the user with the password returned by the cache or user details service. If a match is found, the authentication provider returns the details of the user to the authentication manager. Otherwise, it throws an authentication exception.
14. The authentication manager returns details of the user back to APF. The user is now successfully authenticated.
15. APF saves the user details in the security context created in Step 3 shown in the previous section.
16. APF transfers control to the next filter in the filter chain.

## A simple Weave configuration

You have learned quite a bit about Acegi so far, so we will now have a look at what you can do with what you have learned so far.

Securing Weave requires protecting certain resources, and if a user attempts to access any of the protected resources they will be presented with a login page. When the user signs in using the login page, the application automatically redirects to the requested protected resource.

The user can go directly to the login page, in which case the application presents the login page where the user can sign in. After signing in, the application redirects the user to a start page which is the default resource presented whenever the user signs in without requesting a particular resource.

## Configuring Weave

For Weave the XML file that contains the beans used to configure Acegi is called `security.xml` and is stored in the same directory as the main `config.xml` file.

The following shows a simple `security.xml` file that secures Weave with information contained directly within the `security.xml` file itself



The above example is a fairly basic security filter that stores user information directly in the file but one thing to note here is that only one `client` is actually protected, `private`.

If there is more than one `client` configured for Weave then a user could have access to that one because from the above configuration the user will be logged on anonymously unless they have previously gone to the login page, or if they accessed the 'private' client directly.

So what is actually going on then? Well the important parts here are the two lines in the `objectDefinitionSource`:

```
/private.html=ROLE_USER
/**=IS_AUTHENTICATED_ANONYMOUSLY
```

Taking the second line first, this specifies that any resource (not previously matched) requires that the user be (at least) an anonymous user, but because of the first line any request for a resource matching `private.html` require that the user be part of the `ROLE_USERS` group, which means that they'll have to login to get the the `private` client since the user must access `private.html` to get to the `private` client.

Obviously there is more to the first line than meets the eye. It turns out that this URL is interpreted by the server as being requests to load a specific client configuration (called `private` in the above example) from `config.xml`.

So for our above example we can assume that the `config.xml` file contains at least something similar to the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<config
  xmlns="urn:com.cohga.server.config#1.0"
  xmlns:client="urn:com.cohga.html.client#1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

```

    <client:config id="private">
        <client:default>true</client:default>
        <client:debug>>false</client:debug>
        <client:title>Weave Private HTML Client</client:title>
        ...
    </client:client>
</client>

```

Further, if we wish to provide a public and private client then we could add an additional `client` configuration, as follows:

```

<?xml version="1.0" encoding="UTF-8"?>
<config
  xmlns="urn:com.cohga.server.config#1.0"
  xmlns:client="urn:com.cohga.html.client#1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <client:config id="private">
    <client:default>>false</client:default>
    <client:debug>>false</client:debug>
    <client:title>Weave Private Client</client:title>
    ...
  </client:client>

  <client:config id="public">
    <client:default>true</client:default>
    <client:debug>>false</client:debug>
    <client:title>Weave Public Client</client:title>
    ...
  </client:client>
</config>

```

Now the user has two different client configurations available, `private` and `public`, and they can access `private` using the URL `/weave/private.html` and `public` using `/weave/public.html`.

And, because of the way we have setup Acegi they will be asked to login to access `private` but will have direct access to `public`.

Of course this can be extended to provide multiple clients, both secure and un-secure, but creating different `client` configurations in `config.xml` and adding additional lines to the `objectDefinitionSource`.

Note that in future this may change and the security information provided in the `objectDefinitionSource` will be provided to Acegi based on the ACLs configured in `config.xml` rather than having to duplicate the information (see Access Control Lists later for more information). And, in fact by using ACLs in `config.xml` you can get by with just the `IS_AUTHENTICATED_ANONYMOUSLY` line and leave the `config.xml` file to determine who can access the client config, the disadvantage to this is that currently if the Weave server determines that the user does not have access to a client config then it will redirect the user to the login page, whereas if Acegi performs the access control then it may try perform the user authentication through some other means, for example using NTLM.

## Roles

By using different roles, different users can be granted access to different `client` configurations.

For example, assuming we have three `clients` configured in `client.xml`, `public`, `internal` and `private`, and we want anyone including the public to have access to `public`, we want anyone except the public to have access to `internal` and we only want a small group of people to have access to `private`, then our `objectDefinitionSource` would be:

```

/private.html=ROLE_ADMINISTRATOR
/internal.html=ROLE_USER,ROLE_ADMINISTRATOR
/**=IS_AUTHENTICATED_ANONYMOUSLY

```

Then we would update our `userDetailsService` to include the new `USER_ADMINISTRATORS` role for those users who should get access to the `private client` (note that we have also given administrators access to the `internal client`).

```

<bean id="userDetailsService" class="org.acegisecurity.userdetails.
memory.InMemoryDaoImpl">
  <property name="userMap">
    <value>
      alice=123,ROLE_USER,ROLE_ADMINISTRATOR
      bob=password,ROLE_USER,disabled
      carol=secret,ROLE_USER
      ted,abc,ROLE_USER
    </value>
  </property>
</bean>

```

## Access Control Lists

As mentioned in the introduction, the second half of the security system requires the configuration of Access Control Lists (ACL) to restrict access to items you wish to protect.

When a process initiated by the user attempts to access a restricted item (which is anything with an ACL attached) the groups that the user belongs are checked against the ACL to determine if the user should be given access to the item.

ACL's can be setup to either deny access to everything and selectively enable access, or to allow access to everything and selectively deny access. The former is more secure and recommended.

An ACL provides a list of groups that can either be denied access or allowed access and are processed linearly until there is a match that positively denies or grants approval. Also, there is a special group, `anonymous`, that can be used to allow or deny access to users that are logged in anonymously.

Additionally, ACL's can reference other ACL's to provide a hierarchy with recursive checking performed to determine accessibility.

ACL's can be setup either in-line, that is included directly in the item they're guarding access to, or can be configured individually and referenced by an item.

### IMPORTANT NOTE

#### THERE IS NO ACCESS CONTROL AT ALL UNLESS AT LEAST ONE ACL IS DEFINED

This means that a default installation will provide no access control to any items until at least one ACL is defined, even if it is `default.acl`.

Once at least one ACL is defined then access control restrictions will be enabled. This means that if you are creating a public (or non-sensitive) installation then you do not need to worry about access control lists at all.

It is recommended that if you are going to have any sensitive information then the default ACL should be enabled first and set to deny access to everyone.

#### Authentication via URL parameter

While the security implications should be obvious, it's possible to setup the Acegi security configuration so that the user is authenticated via a parameter provided in the startup URL.

This requires the installation of a specific bundle, that adds the required functionality, along with changes to the `security.xml` file to make use of this functionality.

These changes do not necessarily replace any existing authentication methods, that is, you can use this to authenticate one set of users and continue to use other methods to authenticate other users.

Authenticating a user via a parameter provided in the startup URL could be handy if you wish to provide multiple configurations that are accessible based on the users roles and want to have read/write access to one group of users that must login with a username and password, but you want to provide easy read-only access to another group of users.

## Installation

The UID Security bundle can be downloaded [here](#).

Once downloaded it should be copied to the `weave\platform\plugins` directory, and a server restart will make it available.

Note that at this stage there will be no change or reduction in the security compared to before this bundle was installed, the `security.xml` file must be updated to indicate that the functionality provided by this bundle is to be used.

## Configuration

The first change required to `security.xml` is to change the `authenticationProcessingFilter`.

## Updating the authentication processing filter

Generally `security.xml` will be configured with `org.acegisecurity.ui.webapp.AuthenticationProcessingFilter` as the `authenticationProcessingFilter`, this needs to be replaced with `com.cohga.server.security.uid.AuthenticationProcessingFilter`.

So an existing `authenticationProcessingFilter`

```
<bean id="authenticationProcessingFilter" class="org.acegisecurity.ui.
webapp.AuthenticationProcessingFilter">
    <property name="authenticationManager" ref="
authenticationManager"/>
    <property name="authenticationFailureUrl" value="/login.html?
failure=true"/>
    <property name="defaultTargetUrl" value="/index.html"/>
    <property name="filterProcessesUrl" value="/security
/j_acegi_security_check.do"/>
    <property name="rememberMeServices" ref="rememberMeServices"/>
</bean>
```

would become

```
<bean id="authenticationProcessingFilter" class="com.cohga.server.
security.uid.AuthenticationProcessingFilter">
    <property name="authenticationManager" ref="
authenticationManager"/>
    <property name="authenticationFailureUrl" value="/login.html?
failure=true"/>
    <property name="defaultTargetUrl" value="/index.html"/>
    <property name="filterProcessesUrl" value="/security
/j_acegi_security_check.do"/>
    <property name="rememberMeServices" ref="rememberMeServices"/>
</bean>
```

The `com.cohga.server.security.uid.AuthenticationProcessingFilter` is an extension of the `org.acegisecurity.ui.webapp.AuthenticationProcessingFilter` that will check for a `uid` parameter in the URL and try and authenticate it.

## The authentication process

The authentication process is the same as it was before the switch to the `com.cohga.server.security.uid.AuthenticationProcessingFilter`, that is the `authenticationManager` configured in the `authenticationProcessingFilter` will be called upon to validate that user information, in this case the `uid` parameters value (rather than a username/password provided by the normal login form).

The `authenticationManager` will try and lookup a `userid` and validate that the password for the `userid` is correct, via the `users.properties` file, LDAP, Active Directory, a JDBC database connection, etc., but in this situation we don't have a password to check. Actually, more correctly we do have a password, but it'll be empty, so unless the password corresponding to the `userid` is also empty no match will be found.

This means that you could just add users to be validated via the `uid` parameter to your existing `authenticationManager` source and just make sure that the password is blank for those users, and no further change would be required to `security.xml` to enable `uid` authentication.

✘ You should make sure that passwords for users you don't want authenticated via `uid` are not empty if you're using `com.cohga.server.security.uid.AuthenticationProcessingFilter`, otherwise anyone will be able to login as that user.

## Separating users

If you only have a handful of `userid`s you want to be able to connect via the `uid` parameter and you don't want to, or can't, add users to the existing `authenticationManager` source then you can supplement the `authenticationManager` with a separate source of user details to validate against.

To do this you need to add a new `authenticationProvider` to the `authenticationManager` that points to a `userDetailsService` that provides the additional user list.

For example, assuming that we're currently using an LDAP server for user authentication, then we'd have an `authenticationManager` something like

```
<bean id="authenticationManager" class="org.acegisecurity.providers.
ProviderManager">
    <property name="providers">
        <list>
            <ref local="ldapAuthenticationProvider"/>
            <bean class="org.acegisecurity.providers.
anonymous.AnonymousAuthenticationProvider">
                <property name="key" value="
changeThis"/>
            </bean>
            <bean class="org.acegisecurity.providers.
rememberme.RememberMeAuthenticationProvider">
                <property name="key" value="
changeThis"/>
            </bean>
        </list>
    </property>
</bean>
```

❗ The list of `authenticationProviders` may be different, for example if you're just using `user.properties` to list users then it may contain a reference to a `DaoAuthenticationProvider`, or if you're using a database to store username and passwords it may contain a `JdbcDaoSupport` provider. The point here is that we're going to add another `authenticationProvider` that looks at a different list of users.

## A simple authentication provider

Our first step is to create the new `authenticationProvider`, before we add it to the `authenticationManager`.

In our example where we're going to use a `DaoAuthenticationProvider` and have that look at a list of users embedded directly in `security.xml`. We could just as easily use a `DaoAuthenticationProvider` looking at a file called `uid.properties`, a `JdbcDaoSupport` looking at database tables, `LdapAuthenticationProvider` looking at sub-branch in our LDAP server.


To create our new `authenticationProvider` we need the `DaoAuthenticationProvider` along with a `userDetailsService` which we add with the following additions to `security.xml`


```

<bean id="uidAuthenticationProvider" class="org.acegisecurity.
providers.dao.DaoAuthenticationProvider">
    <property name="userService" ref="uidDetailsService"/>
    <property name="userCache">
        <bean class="org.acegisecurity.providers.dao.cache.
EhCacheBasedUserCache">
            <property name="cache">
                <bean class="org.springframework.
cache.ehcache.EhCacheFactoryBean">
                    <property name="cacheManager">
                        <bean class="org.
springframework.cache.ehcache.EhCacheManagerFactoryBean"/>
                    </property>
                    <property name="cacheName"
value="userCache"/>
                </bean>
            </property>
        </bean>
    </property>
</bean>

<bean id="uidDetailsService" class="org.acegisecurity.userdetails.
memory.InMemoryDaoImpl">
    <property name="userMap">
        <value>
            north=,ROLE_NORTH,ROLE_READONLY
            south=,ROLE_SOUTH,ROLE_READONLY
        </value>
    </property>
</bean>

```

 Note the comma immediately after the equals sign for each user listed in the `uidDetailsService`. This is important since the password is listed between the equals sign and the comma, which for our purposes when using a `com.cohga.server.security.uid.AuthenticationProcessingFilter`, needs to be empty. Following this initial comma is a comma separated list of the roles that will be assigned to each user.

 In our example above each user has 2 roles, this may or may not be the same way that your roles need to be configured, and is just one possible way of assigning roles to these users

This created the `DaoAuthenticationProvider` called `uidAuthenticationProvider` that references the list of users named `uidDetailsService`.

The final step would be to add this new `authenticationProvider` to the `authenticationManager` as by changing it to

```

<bean id="authenticationManager" class="org.acegisecurity.providers.
ProviderManager">
    <property name="providers">
        <list>
            <ref local="uidAuthenticationProvider"/> <!--
NEW UID AUTHENTICATION PROVIDER -->
            <ref local="ldapAuthenticationProvider"/>

```

```

        <bean class="org.acegsecurity.providers.
anonymous.AnonymousAuthenticationProvider">
            <property name="key" value="
changeThis"/>
        </bean>
        <bean class="org.acegsecurity.providers.
rememberme.RememberMeAuthenticationProvider">
            <property name="key" value="
changeThis"/>
        </bean>
    </list>
</property>
</bean>

```

### Usage

Once we've switched over to the new `authenticationProcessingFilter` we should be able to start a specific Weave client configuration with an additional `uid` parameter, as follows

```
http://server:8080/weave/north_region.html?uid=north
```

And assuming that the appropriate access controls lists have been created referencing the correct roles, for example

```

<acl:acl id="editors">
    <entry type="deny">ROLE_READONLY</entry>
</acl:acl>

<acl:acl id="north">
    <entry type="allow">ROLE_NORTH</entry>
    <entry type="deny">*</entry>
</acl:acl>

<acl:acl id="south">
    <entry type="allow">ROLE_SOUTH</entry>
    <entry type="deny">*</entry>
</acl:acl>

```

You can then setup client configuration items similar to

```

<client:config id="north_region">
    <acl id="north"/>

    <!-- other configuration here -->

    <!-- add a tool only for users fully logged in -->
    <item action="example.editing.tool" acl="editors"/>

    <!-- rest of configuration here -->

```

```

</client:config>

<client:config id="south_region">
  <acl id="south"/>

  <!-- other configuration here -->

  <!-- add a tool only for users fully logged in -->
  <item action="example.editing.tool" acl="editors"/>

  <!-- rest of configuration here -->

</client:config>

```

Then assuming that your previous `authenticationProvider` assigned `ROLE_NORTH` and/or `ROLE_SOUTH` to other users, and does not assign `ROLE_READONLY`, then user logging in via the `uid` parameter won't have access to the `example.editing.tool` whereas those logging in via the previous methods will.

And, of course we can also attach these access control lists to other items, such as report, searches, etc.

Encrypting content in the `security.xml` file

As of Weave 2.5.16 and 2.5.15.5 it's possible to encrypt the password, and in fact any values, that are stored in the `security.xml` file using the same encryption that is used to encrypt passwords in the other Weave configuration files.

**i** The encrypted values generated with the `encrypt` command are dependant upon the `private.key` file, if you want to use the same encrypted text on multiple Weave instances you should ensure they all use the same `private.key` file as the instance used to generate the original encrypted text.

## Directly encrypting passwords

If you just want to encrypt the passwords stored in the `security.xml` file you first need to encrypt the original value using the `osgi encrypt` command, e.g.

```

osgi> encrypt $up3r$3cr3tP@55w0rd
ENCEAUFJCUABCFXEMFQABFJEBZJAKFCXGME

```

Then you need to add a new bean to the `security.xml` file so that the encrypted values are decoded when processing the `security.xml` file, e.g.

```

<bean class="com.cohga.beans.factory.config.
EncryptedPasswordProcessor">
  <property name="key" value="private.key"/>
</bean>

```

The `key` value should almost always be set to `private.key`, which is the name of the key file that Weave uses to encrypt/decrypt the passwords.

**i** The above example shows how the bean would be defined in the older Acegi Security file, in the newer Spring Security file the `bean` and `property` tags would probably need to include a `beans` namespace prefix, e.g.

```

<beans:bean class="com.cohga...
  <beans:property name="key...
</beans:bean>

```



Once you've added the `EncryptedPasswordProcessor` bean you can then update the passwords in the `security.xml` file to use the encrypted value, e.g.

```
<bean id="initialDirContextFactory" class="org.acegisecurity.ldap.
DefaultInitialDirContextFactory">
  <constructor-arg value="ldap://10.0.12.213:389/" />
  <property name="managerDn">
    <value>CN=Administrator,CN=Users,DC=adel,DC=cohga,
DC=com</value>
  </property>
  <property name="managerPassword">
    <value>ENCEAUFJCUABCFXEMFQABFJEBZJAKFCXGME</value>
  </property>
</bean>
```

Here the `managerPassword` value has been changed to use the encrypted password.

## Using an external properties file

To do this you need to first encrypt the passwords at the osgi console using the `encrypt` command, e.g.

```
osgi> encrypt $up3r$3cr3tP@55w0rd
ENCEAUFJCUABCFXEMFQABFJEBZJAKFCXGME
```

Then you create a properties file to store the value(s), named `security.properties` for example, and add a key/value line for each value you want to use, e.g.

```
password=ENCEAUFJCUABCFXEMFQABFJEBZJAKFCXGME
```

After you've setup the properties file with all the encrypted values you want to use in `security.xml` you need to update `security.xml` so that it firstly knows that it should get values from the new properties file and secondly replace the unencrypted values with references to the encrypted properties.

The first step, loading the properties, is done by adding a new reference to an `EncryptedPropertyPlaceholderConfigurer`, note that this is an extension of the standard `PropertyPlaceholderConfigurer` which allows you to store properties to use in `security.xml` in an external file but the file contents aren't encrypted. This new bean can be added anywhere in the file and is not directly referenced by any other beans, you just need to add the definition.

```
<bean class="com.cohga.beans.factory.config.
EncryptedPropertyPlaceholderConfigurer">
  <property name="key" value="private.key" />
  <property name="locations" value="security.properties" />
</bean>
```

The `key` value should almost always be set to `private.key`, which is the name of the key file that Weave uses to encrypt/decrypt the passwords. The `locations` value should point to the name of the file that contains the properties that you've encrypted, the file can be called anything as long as this properties points to that file.

**i** The above example shows how the bean would be defined in the older Acegi Security file, in the newer Spring Security file the `bean` and `property` tags would probably need to include a `beans` namespace prefix, e.g.

```
<beans:bean class="com.cohga...
```

```

    <beans:property name="key..."
  </beans:bean>

```

Once you've added the `EncryptedPropertyPlaceholderConfigurer` bean you can then update other properties in the `security.xml` file to reference the values from the file using `${name}` as a placeholder to the property value, e.g.

```

<bean id="initialDirContextFactory" class="org.acegisecurity ldap.
DefaultInitialDirContextFactory">
  <constructor-arg value="ldap://10.0.12.213:389/" />
  <property name="managerDn">
    <value>CN=Administrator,CN=Users,DC=adel,DC=cohga,
DC=com</value>
  </property>
  <property name="managerPassword">
    <value>${password}</value>
  </property>
</bean>

```

Here the `managerPassword` value will be taken from the `password` value stored in the `security.properties` file.

**i** As mentioned earlier any value can be replaced with a value from the properties file, so if you didn't want to have a username visible in the file you could also store that value in the properties file.

Also, the values in that file don't have to be encrypted, if you just want to store values in an external file, for example if you want to have the same `security.xml` file in a test and production servers but they both reference different values you could also store those values in an external properties file and reference them in `security.xml` instead.

If you don't need the values to be encrypted at all you could just use the `org.springframework.beans.factory.config.PropertyPlaceholderConfigurer` bean and don't bother to set the `key` property.

### Forcing a user to login

If you're using the login page to get the users to login, as opposed to using Windows integrated authentication, you can force the user to have to login before they can do anything.

By default an anonymous user (one that hasn't logged in yet) can see a list of clients that are available, and from there they can login with the login button, but by using the changes listed below the user is forced to login before they can perform any operations at all.

To make the change you need to change the `objectDefinitionSource` in the following section in `security.xml`

```

    <bean id="filterInvocationInterceptor" class="org.
acegisecurity.intercept.web.FilterSecurityInterceptor">
      <property name="authenticationManager" ref="
authenticationManager" />
      <property name="accessDecisionManager">
        <bean class="org.acegisecurity.vote.
AffirmativeBased">
          <property name="
allowIfAllAbstainDecisions" value="false" />
          <property name="decisionVoters">
            <list>
              <bean class="org.
acegisecurity.vote.RoleVoter" />
              <bean class="org.
acegisecurity.vote.AuthenticatedVoter" />
            </list>
          </property>
        </bean>
      </property>
    </bean>

```

```

                </property>
            </bean>
        </property>
        <property name="objectDefinitionSource">
            <value>
                CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
                PATTERN_TYPE_APACHE_ANT
                /**=IS_AUTHENTICATED_ANONYMOUSLY
            </value>
        </property>
    </bean>

```

To force authentication the `objectDefinitionSource` should be changed to:


```

        <property name="objectDefinitionSource">
            <value>
                CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
                PATTERN_TYPE_APACHE_ANT
                /login.*=IS_AUTHENTICATED_ANONYMOUSLY
                /login/**=IS_AUTHENTICATED_ANONYMOUSLY
                /**=IS_AUTHENTICATED_FULLY
            </value>
        </property>

```

This allows the resources that the user needs to login to be available to anyone, but requires the user to be full authenticated before they can access anything else.

### Securing the Admin UI

 This page applies to sites using the Acegi security provider but should be easily adapted for Spring Security

The Administration GUI can be accessed from the `/weave/admin.html` URL once the Weave instance is running.

To secure access to the GUI you need to edit the `security.xml` file and add three entries to the list of paths that need to be secure. Three new entries for the resources used by the admin UI need to be added to the `filterInvocationInterceptor`.

In a default Weave installation the final entry in this list should be:

```
/**=IS_AUTHENTICATED_ANONYMOUSLY
```

To secure the administration GUI you need to add three new entries before that one.

```
/admin.html=ROLE_ADMIN
```

```
/admin/**=ROLE_ADMIN
```

```
/services/admin/**=ROLE_ADMIN
```

Note that `ROLE_ADMIN` may need to change depending upon how you've updated the authentication in the rest of the `security.xml` file. The role represents a group that the user must belong to before they can access the Administration GUI. For example, if you're using the default `users.properties` file then the following entries will grant access to Bob and Ted, but not Alice:

```
bob=password,ROLE_ADMIN
```

```
ted=password,ROLE_ADMIN,ROLE_USER,ROLE_GIS,ROLE_PLANNING
```

```
alice=password,ROLE_USER,ROLE_GIS,ROLE_PLANNING
```

If you're using LDAP, Active Directory or some other authentication mechanism then the role you need to set will be based on the groups the user is granted based on those authentication sources. These roles are the same ones that you would use when creating an ACL within Weave itself.

### Spring Security

Weave 2.5 provides the opportunity to switch from the Acegi security to Spring security (which is the newer version of the Acegi security framework).

If your current security.xml file works with Weave 2.4 then you do not need to change it to use Spring security when switching to Weave 2.5, the upgrade to provide support for Spring security was done because Acegi security is no longer developed and has been succeeded by Spring security, which is still being developed and provides support for additional security mechanisms not supported by the Acegi security framework.

Weave 2.5 provides support for both system, and will dynamically choose which framework to use based on the content of the security.xml file. The security.xml file for Spring security is different from the Acegi version, but they're both based on the same concepts.

Below is the minimum security.xml file required to use Spring security with Weave 2.5.

It outlines the basic changes that must be made to a the equivalent basic Spring security configuration file to properly support Weave. Some of the changes are to ensure that users can and can't access the parts of Weave they need to, and the others are to make the file compatible with what Weave is expecting based on what it previously did for Acegi security.

Beyond the contents of this file any customisations you need to make to support you security requirements (i.e. Active Directory, LDAP, etc.) *should* be made based on the standard Spring security documentation available at [Spring security documentation](#)

### Base security.xml file

```
<?xml version="1.0" encoding="UTF-8"?>

<beans:beans xmlns="http://www.springframework.org/schema/security"
  xmlns:beans="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema
/beans
          http://www.springframework.org/schema/beans/spring-
beans-3.2.xsd
          http://www.springframework.org/schema/security
          http://www.springframework.org/schema/security/spring-
security-3.2.xsd">

  <!-- Allow access to these resources by anyone -->
  <http pattern="/resources/favicon/favicon.ico" security="none"
/>

  <http pattern="/resources/images/**" security="none"/>

  <http auto-config="true" create-session="always">
    <!-- Only "Admin" users should be able to access the
admin pages -->
    <!-- You can change ROLE_ADMIN to match a different
role if you are using
          an external authentication source, for example
AD or LDAP, and it uses
          a different role for the users that you want to
have access the Admin UI -->
    <intercept-url pattern="/admin.html" access="
ROLE_ADMIN"/>
    <intercept-url pattern="/admin.htm" access="
ROLE_ADMIN"/>
    <!-- These are the pages where we want to know who
the user is, but they can be anonymous -->
    <intercept-url pattern="/index.html" access="
IS_AUTHENTICATED_ANONYMOUSLY"/>
    <intercept-url pattern="/index.htm" access="
IS_AUTHENTICATED_ANONYMOUSLY"/>
    <intercept-url pattern="/login.*" access="
IS_AUTHENTICATED_ANONYMOUSLY"/>
```

```

        <intercept-url pattern="/login/**" access="
IS_AUTHENTICATED_ANONYMOUSLY"/>
        <intercept-url pattern="/**" access="
IS_AUTHENTICATED_ANONYMOUSLY"/>
        <form-login login-page="/login.html" login-processing-
url="/security/j_spring_security_check.do" authentication-success-
handler-ref="authenticationSuccessHandler"/>
        <logout logout-url="/security
/j_spring_security_logout.do" logout-success-url="/index.html" />
        <!-- The default anonymous username is
"anonymousUser", we change it here to "anonymous"
        because that is what it was in older versions of
Weave, plus it looks nicer when it's displayed. -->
        <!-- Also, we change the default roles from just
ROLE_ANONYMOUS, to anonymous and ROLE_ANONYMOUS,
        because the older Weave ACL parser and it's
documentation says that "anonymous" can be used as
        a role for anonymous users. Internally Weave now
uses ROLE_ANONYMOUS, but we want to make sure old
        ACL's still work, so make sure you always
include at least ROLE_ANONYMOUS -->
        <!-- In summary, without the following line an
anonymous user would be displayed as "anonymousUser"
        and any ACL's that wants to check for an
anonymous user would have to use "ROLE_ANONYMOUS" -->
        <anonymous username="anonymous" granted-authority="
anonymous,ROLE_ANONYMOUS"/>
    </http>

    <!-- This custom authentication success handler is used to
allow us to manually change the target page after a successful
authentication -->
    <beans:bean id="authenticationSuccessHandler" class="org.
springframework.security.web.authentication.
SimpleUrlAuthenticationSuccessHandler">
        <!-- This is the page we will redirect to if the user
goes straight to the login page and logs in -->
        <beans:property name="defaultTargetUrl" value="/index.
html"/>
        <!-- The reason we have this bean is so that we can
set the following value -->
        <!-- This allows us to manually set the target page
after performing authentication -->
        <beans:property name="targetUrlParameter" value="
_spring_security_redirect"/>
    </beans:bean>

    <authentication-manager alias="authenticationManager">
        <authentication-provider user-service-ref="
userDetailsService"/>
    </authentication-manager>

    <user-service id="userDetailsService" properties="users.
properties"/>

```

```
</beans :beans>
```

## SAML

Security Assertion Markup Language (SAML) is an open standard where you can use one set of credentials to log in to many different websites.

As of Weave 2.6.5, SAML 2.0 is supported by Weave as a means of authenticating users.

SAML is the specification that outlines the steps required to perform the authentication but there are multiple different implementations of SAML Identity Providers (IdP), Weave itself is a SAML Service Provider (SP), this includes:

- Active Directory Federation Services (ADFS) - a Single Sign-On (SSO) solution created by Microsoft
- Azure Active Directory (Azure AD) - a cloud based SSO solution provided by Microsoft
- Keycloak - an open source identity and access management solution provided by Red Hat
- Cognito - an AWS identity provider solution provided by Amazon
- Okta - a cloud based SAML identity provider created by Okta, Inc.

Weave itself uses the Keycloak Java library, which is different from the Keycloak identity provider listed above but is provided by the same team, for communicating with whichever SAML Identity Provider you wish to integrate with. This page documents how to configure Weave so that it will use this Keycloak library to communicate with your chosen SAML IdP.

### Cookies

The SAML authentication process relies upon the caller having a valid "session", both for the authentication process itself and when sending requests to the server after authentication (for requests that are required to be authenticated). Currently this session tracking requires the use of browser cookies.

### Reverse Proxy

If there is a reverse proxy used in front of the Weave server then additional configuration must be setup for Weave to tell it that there is a reverse proxy. This should really be done for any Weave instance behind a reverse proxy but it is not required. However, if you're using SAML authentication and the Weave server is behind a reverse proxy, then it is required that you perform this additional setup.

More information about setting up Weave when it's behind a reverse proxy can be found [here](#).

Note that for SAML authentication to work the context path exposed by the reverse proxy must be the same as that exposed by Weave. i.e. by default Weave is published at `/weave` but if you want to expose it as `/maps` normally you would have the reverse proxy expose `/maps` and forward the request to the Weave server at `/weave`, but this won't work. You need to change the Weave server so that it also exposes Weave at `/maps`, and to do this you should edit the `...\weave\jetty_base\webapps\weave.xml` file and change the `contextPath` setting. Note that this also applies if you don't want to use a context path for Weave and just expose it at `/`, just set the `contextPath` value to `"/"` in `weave.xml` in that case.

### HTTPS

It is likely that the SAML Identity Provider you're trying to integrate with Weave will require that the callback request it makes to the Weave server be made using a secure connection, meaning that the Weave server will have to be configured, and accessed by the user, via HTTPS rather than plain HTTP.

### Acegi, Spring Security, Container and SAML

With the addition of SAML support, Weave now provides four methods of authenticating users. These methods are:

- Acegi and Spring security which are configured using a single XML file, `security.xml` (or `acegi.xml` and `spring.xml` in Weave 2.6.5 or later).
- Container based security, which is the default if no configuration file is available for the other authentication mechanisms. This method relies upon the underlying Web Application Server to perform the authentication.
- And now SAML which requires two configuration files, `keycloak.properties`, a Weave specific configuration file documented [here](#), and `keycloak-saml.xml`, a Keycloak specific XML configuration file [documented here](#).

Note that configuring an application to authenticate with a SAML IdP requires specialist knowledge about SAML and the specific Identity Provider you're using. *This page is not intended to be a tutorial on SAML.*

### Disable/Enable Security Providers

Since you can only utilize a single security provider for a given Weave instance, the security provider is normally determined by the existence of its related configuration file, `security.xml` (or `acegi.xml` and `spring.xml` in 2.6.5+) for Acegi and Spring Security, `keycloak.properties` and `keycloak-saml.xml` for SAML, or nothing for container security.

But in some situations you may need to disable one or more of the installed security providers. One way to do this would be to remove the relevant `com.cohga.server.security.acegi`, `com.cohga.server.security.spring`, `com.cohga.server.security.container` or `com.cohga.server.security.keycloak` plugins from the `...\weave\platform\plugins\` directory. But if that is not possible (for example if you're running Weave from a Docker image that includes all the plugins and a default configuration file) you can also set a system property to disable one or more of the providers.

Setting the system properties `weave.acegi.security`, `weave.spring.security`, `weave.container.security` and/or `weave.keycloak.security` to `false` will disable the respective plugin from being used even if its configuration file(s) exist.

#### keycloak.properties

The `keycloak.properties` file is processed by Weave to determine some global settings required by Weave when authenticating incoming requests, like the name of the `keycloak-saml.xml` file if you don't want to use the default name, or a list of URL's that do not need to be authenticated.

```
allowAnonymous=false
logRequests=false
includePaths=
excludePaths=
includeBeforeExclude=false
```

Above are the current default values (*not* the recommended values) for the properties available to be set via `keycloak.properties`.

- `allowAnonymous`
  - By default, every user will have to login before they can access anything. By setting `allowAnonymous` to `true` users can access Weave, as an anonymous user, without having to login.
- `logRequests`
  - Certain HTTP headers are utilised when performing the SAML authentication process but logging those headers continually would pollute the Weave log with information that would be useless most of the time. So you can set `logRequests` to `true` while trying to debug authentication issues.
- `includePaths`
  - `includePaths` can be set to a comma-separated list of URL patterns that should always be forced to be authenticated via SAML, even if `allowAnonymous` is set to `true`. Any request URL that matches one of the entries in `includePaths` will be required to be authenticated before it is processed by the server, either before the request is made or by performing the SAML authentication process as part of the request processing.
  - There's no recommended value for the `includePaths` setting, but if you're doing something like setting up a Weave server that's just used by other servers and not as a user-facing Weave instance, then it could be useful for locking down the server in combination with `excludePaths`.
- `excludePaths`
  - `excludePaths` can be set to a comma-separated list of URL patterns that should never be authenticated via SAML, even if `allowAnonymous` is set to `false`. This is for things like server health check URL's that need to be made available to monitor the server and you don't want/need to perform authentication before processing the request.
  - The recommended value for `excludePaths` is: `/metrics,/server/ping,/server/healthcheck,/server/health.do,/server/mq.do`
- `includeBeforeExclude`
  - Normally Weave checks the `excludePaths` list before the `includePaths` list to determine if a given request should be forced to be authenticated or forced to not be authenticated. You can reverse this check so that `includePaths` is checked first then `excludePaths` by setting `includeBeforeExclude` to `true`.
  - This setting is provided because both `includePaths` and `excludePaths` can include wildcards and you may wish to include a more permissive value in one list whilst including a more specific value(s) in the other list. By providing this setting, you can determine which list it is easier to make permissive and which one to make more specific.
- `roleFilter.*`
  - You can add a number of `roleFilter` entries to ensure that a user has a particular roles before they can access the given url.
  - Any suffix can be used for the filter name, and the values are applied in alphabetical order based on the filter name (not the order listed)
  - The value for each filter should contain a URL path to match followed by an equals sign followed by a comma-separated list of roles that the user must have at least one of before they can access the given path
  - This should be done for at least the Admin UI (replacing `ROLE_ADMIN` with the role that admin users will have)
  - This is only available with version 12.0.4 or later of the Keycloak security plugin
- `userProperties`
  - The path to a user properties file that can be used to replace the roles assigned to users
  - If the user is listed in this file then the roles assigned in this file will completely replace those provided by the SAML server, that is, there is currently no way to assign additional roles, or remove individual roles from those provided by the SAML server.
  - The file should contain a single line for each user in the format:
    - `username@domain.name=role1,role2,...,roleN`
  - Note that that is assuming that the usernames returned from the SAML server contain a domain, e.g. `@domain.name` in the above example, if the SAML server isn't returning the domain in the username then it does not need to be included in the user properties file.
  - If one of the roles, or the only role, listed for the user is "disabled" then the user will be provided with no roles, this can be used to override what's in the SAML server and disable access for the user
- `userPropertiesHasPassword`
  - If the user properties file specified by the `userProperties` setting has been copied from an existing `users.properties` file then file will likely contain passwords that aren't used by Keycloak and should be ignored as a "role" assigned to the user when reading the file. So if this property is set to "true" then it indicates that the Keycloak user properties file *does* contain a password and the password should be ignored, but if this property either isn't set or is set to anything other than "true" then it indicates that the Keycloak user properties file *does not* contain a password

- `userPropertiesHasDomain`
  - This flag indicates that the usernames stored in the Keycloak user properties file contain a domain for the user, i.e. the user is listed as “`user@example.com=...`” rather than just “`user=...`”.
  - Most times the username returned from the SAML server *will* contain the domain and the Keycloak user properties file should be setup so that each user entry also contains a domain, this setting is primarily intended to make it easy to migrate an existing `users.properties` file, and can likely be ignored if you’re creating a new Keycloak user properties file from scratch, in which case you’d include the full username, including the domain, in the file
  - If this property is set to “`true`” or not set at all then it’s assumed that the Keycloak user properties file contains the domain for a user, if this property is set to “`false`” then it’s assumed that the Keycloak user properties file does not contain a domain for the user, which may be the case if the file is copied from an existing `users.properties` file.
  - This may be required to be explicitly set to `false` if you’re copying an existing `users.properties` file that doesn’t contain a domain but the SAML server is returning usernames that do contain a domain. Note that this assumes that the user names listed in the file (without a domain) are unique, which may not be the case, in which case the file should be updated to include the domain for each user and this property not be set (or set to `true`).

The following would be a recommended `keycloak.properties` file (if anonymous user access is also required)

```
allowAnonymous=true

excludePaths=/metrics,\
/server/ping,\
/server/healthcheck,\
/server/health.do,\
/server/mq.do

roleFilter.1=/admin.*=ROLE_ADMIN
roleFilter.2=/admin/**=ROLE_ADMIN
```

**Note:** The role of `ROLE_ADMIN` above is an example only. Replace it with the role which should be granted access to the Admin UI.

`keycloak-saml.xml`

`keycloak-saml.xml` is the Keycloak specific XML configuration file [documented here](#). The settings you need to specify in that file will be determined by the identity provider you’re using and will likely require assistance from someone familiar with that particular identity provider but is something that Cohga can provide assistance with.

One thing to note from the documentation linked above is that when it refers to external files, for example, a Java keystore containing certificates required to protect/verify communications with the SAML server, it’s assuming that those files are embedded within a “web application” and references those files relative to a `WEB-INF` directory. However for Weave, those files are located adjacent to the `keystore.properties` file so do not require that the files are referenced relative to a `WEB-INF` directory.

Note that a number of values in the `keycloak-saml.xml` that reference external files, mostly related to various certificate and private keys, can instead be directly embedded in the file, as base64 encoded strings. This can make maintenance of the file easier by limiting the number of files that need to be tracked. For example `KeyStore\PrivateKey` vs `PrivateKeyPem`.

### Some additional notes about `keycloak-saml.xml` settings

Some settings in `keycloak-saml.xml` will have an effect on Weave if not set correctly.

- `autodetectBearerOnly`
  - This *must* be set to `true`. If set to `true` it will allow the Weave client, and any other “client” that uses the Weave REST API, to determine that a request sent to the server requires authentication before it will be processed. If the value is not set to `true` then a request that requires authentication would likely be redirected to a HTML page to start the authentication process, which is not what you want when using a REST API.

### SAML - Azure AD sample

The following is an example `keycloak-saml.xml` file for integrating Weave with Azure AD.

Note that there are some values in this file that will need be replaced with values from your setup and will need to be extracted from Azure. The `SP entityID` attribute (`APPLICATION_ID_FROM_AZURE` in the example below) taken from the *Application (client) ID* and the *Directory (tenant) ID* (`DIRECTORY_ID_FROM_AZURE` in the example below) for the various Azure URL endpoints.

When configuring the application authentication in Azure the redirect URL (back to Weave) should be to `https://host.domain/weave/saml` (assuming the default application context is still `/weave`, and Weave is exposed on port 80)

Note, this example does not perform any verification of the information exchanged between the Azure AD identity provider and Weave, which is **not** recommended. You can find a more complete example that does use certificates at [Keycloak - Securing Applications and Services Guide](#).



```

<keycloak-saml-adapter xmlns="urn:keycloak:saml:adapter"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:keycloak:saml:adapter
https://www.keycloak.org/schema/keycloak_saml_adapter_1_10.xsd">

    <SP entityID="spn:APPLICATION_ID_FROM_AZURE"
        sslPolicy="EXTERNAL"
        nameIDPolicyFormat="urn:oasis:names:tc:SAML:1.1:
nameid-format:unspecified"
        logoutPage="http://www.cohga.com/"
        forceAuthentication="false"
        isPassive="false"
        turnOffChangeSessionIdOnLogin="false"
        autodetectBearerOnly="true">

        <!-- This sets up the attribute returned by Azure AD
to use as the username -->
        <PrincipalNameMapping policy="FROM_ATTRIBUTE"
attribute="http://schemas.xmlsoap.org/ws/2005/05/identity/claims
/emailaddress" />

        <!-- This sets up the attribute returned by Azure AD
to determine the roles the user has -->
        <!-- these roles should be referenced in your Weave
ACL configurations -->
        <RoleIdentifiers>
            <Attribute name="http://schemas.microsoft.com
/ws/2008/06/identity/claims/role" />
        </RoleIdentifiers>

        <IDP entityID="idp"
            signaturesRequired="false">
            <SingleSignOnService requestBinding="POST"
                bindingUrl="https://login.
microsoftonline.com/DIRECTORY_ID_FROM_AZURE/saml2"
                signRequest="false"
                validateAssertionSignature="false"
                validateResponseSignature="false" />

                <SingleLogoutService
                    requestBinding="POST"
                    responseBinding="POST"
                    postBindingUrl="https://login.
microsoftonline.com/DIRECTORY_ID_FROM_AZURE/saml2"
                    redirectBindingUrl="https://login.
microsoftonline.com/DIRECTORY_ID_FROM_AZURE/saml2"
                    signRequest="false"
                    signResponse="false" />

        </IDP>
    </SP>

</keycloak-saml-adapter>

```

Home > Default Directory

## Default Directory | App registrations

Azure Active Directory

- Overview
- Preview features
- Diagnose and solve problems

**Manage**

- Users
- Groups
- External Identities
- Roles and administrators
- Administrative units
- Enterprise applications
- Devices
- App registrations**

Home > Default Directory >

+ New registration | Endpoints | Troubleshooting | Refresh | Download | Preview features | Got feedback?

All applications | **Owned applications** | Deleted applications | Applications from personal account

Start typing a display name to filter these results

Application (client) ID starts with [x] Add filters

2 applications found

Display name ↑↓	Application (client) ID	Created on ↑↓	Certificates & secrets
weave	...e7-465d-9b21-793201c6...	11/15/2018	-
Weave Demo	...7d5-4e71-97c3-10af13e67...	1/4/2021	-

Home > Default Directory >

## Weave Demo

Search (Ctrl+/)

Delete | Endpoints | Preview features

- Overview
- Quickstart
- Integration assistant

**Manage**

- Branding
- Authentication
- Certificates & secrets
- Token configuration
- API permissions

Home > Default Directory > Weave Demo

Essentials

Display name <a href="#">Weave Demo</a>	Client credentials <a href="#">Add a certificate or secret</a>
Application (client) ID ...5-4e71-97c3-10af13e67954	Redirect URIs <a href="#">1 web, 0 spa, 0 public client</a>
Object ID ...4c-41b2-a2fe-fef497bb7c70	Application ID URI <a href="#">Add an Application ID URI</a>
Directory (tenant) ID ...bb-489a-b40c-49fbc1ea7c26	Managed application in local directory <a href="#">Weave Demo</a>
Supported account types <a href="#">My organization only</a>	

Home > Default Directory > Weave Demo

## Weave Demo | Authentication

Search (Ctrl+/) | Save | Discard | Got feedback?

- Overview
- Quickstart
- Integration assistant

**Manage**

- Branding
- Authentication**
- Certificates & secrets
- Token configuration
- API permissions
- Expose an API
- App roles
- Owners

Platform configurations

Depending on the platform or device this application is targeting, additional configuration may be required such as redirect URIs, specific authentication settings, or fields specific to the platform.

+ Add a platform

**Web** Quickstart Docs Docs

Redirect URIs

The URIs we will accept as destinations when returning authentication responses (tokens) after successfully authenticating or signing out users. The redirect URI you send in the request to the login server should match one listed here. Also referred to as reply URLs. [Learn more about Redirect URIs and their restrictions](#)

[Add URI](#)

### Clients

### Client configurations, aliases and URLs

When you configure a client by creating a <client:config> section in a config.xml file you're actually creating a unique URL that the user can access that particular client from.

For example if you create a client config with the id of 'main'

### Basic client configuration

```
<client:config id="main">
  <title>My Super Client</title>
  <!-- other config content goes here -->
</client:config>
```

The the users will be able to access the client using `/weave/main.html`.

Similarly if you create a client config with the id 'test' or 'external' then direct URL would be `/weave/test.html` and `/weave/external.html` respectively.

**i** The extensions `.html` and `.htm` can both be used to access the clients.

## index.html and the default client

`index.html` is a special page that's handled by Weave differently. For a start this is the page that will be displayed if they don't specify a client directly, that is, `/weave/` will in fact display the equivalent content of `/weave/index.html`.

Furthermore `index.html` does some additional processing depending upon who the user is and how many clients configurations you've created.

Ignoring security for a moment, assuming everyone has access to everything, if you only have a single client configured then `index.html` will just forward the user to that one client, e.g. `/weave/` will forward to `/weave/index.html`, then `/weave/index.html` will forward to `/weave/main.html` (assuming the only client you have configured has the id of 'main').

If however you have multiple clients configured then if the user accesses `index.html`, either directly or indirectly, then the user is presented with a page containing a list of the clients that are available, and they must choose a specific client that they want to access before the client proper will be loaded.



You can disable the display of a specific client in this list by setting `publish` to `false` in the client config.

### Hiding a client configuration

```
<client:config id="test">
  <publish>false</publish>
  <title>My Test Client</title>
  <!-- other config content goes here -->
</client:config>
```

Note that if this results in there only being a single published client available then the user will again be automatically redirected to that client.

You can also completely disable a client configuration by setting `enable` to `false`.

### Disabling a client configuration

```
<client:config id="test">
  <enable>false</enable>
  <title>My Test Client</title>
  <!-- other config content goes here -->
</client:config>
```

This would be the equivalent of completely removing the client configuration from the config.xml file, and is useful for quickly removing access to a client without having to remove it from the .xml file.

### Access control and index.html

Bringing security back into the picture alters the landscape slightly. Specifically in the situation where a client config has an ACL attached to it and the user does not pass the ACL check, then the user will not have access to that client config.

This could mean that the user is reduced to a single client config (assuming there is one other client config with an ACL that the user passes or it doesn't have an ACL at all) in which case they'll be directed to that config in the same way as they would if there were only one client config.

Or, if they have more than one client config accessible, after the ACL checks, then they'll again be presented with the page listing all available client, but the page will only display those clients that the user has access to (and that are published and enabled).

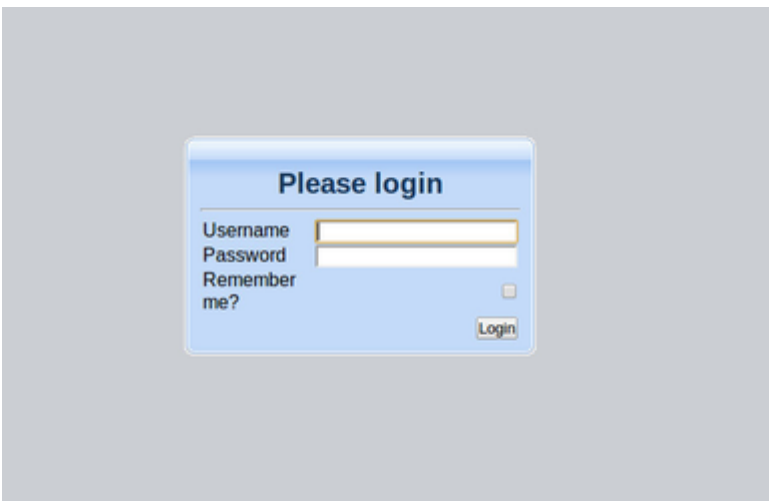
Finally, if they have no client configs accessible then they'll be directed to a login page.

### Client config only available to users that pass the 'admin' acl

```
<client:config id="test">
  <acl>admin</acl>
  <title>My Test Client</title>
  <!-- other config content goes here -->
</client:config>
```

**⚠** Just because a client config has an ACL attached to it does not mean that a user will not be able to access that client config, either if they're logged on or not, since the ACL may specify that 'anonymous' users are acceptable.

In addition to displaying the list of available client the list page also includes a button that allows the user to login (this button will allow the user to logout if they're already logged in). This button will direct the user to the `login.html` page, this is the same login page that the user would be directed to if they did not have access to any client configurations because of ACL's.



This page provides the user with an opportunity to login to gain access to the system if all clients are locked down and the user has not already been authenticated (for example by using NTLM authentication which generally occurs before the `index.html` page is displayed).

## Advanced access control with security.xml

### Restricting anonymous access

There are two ways to restrict access to Weave so that a user has to be authenticated before they can access the system (well three if you count automatic login mechanisms like NTLM) the first is to use the information above and set an ACL for each client config that disallows access to anonymous users

#### Restricting access to unauthenticated users

```
<acl:acl id="authenticated">
  <entry type="deny">anonymous</entry>
  <entry type="allow">*</entry>
</acl:acl>

<client:config id="main">
  <acl>authenticated</acl>
  <title>My Super Client</title>
  <!-- other config content goes here -->
</client:config>

<client:config id="test">
  <acl>authenticated</acl>
  <title>My Test Client</title>
  <!-- other config content goes here -->
</client:config>
```

The other method is to use `security.xml` to ensure that the user is authenticated before they can even access the system.

This is done using the `FilterSecurityInterceptor` in `security.xml`. By default the `FilterSecurityInterceptor` supplied with Weave is configured to ensure that each user is at least an anonymous user, this is the lowest level of authentication and basically ensures that we at least have a user object to work with. Changing the `objectDefinitionSource` in the `FilterSecurityInterceptor` from `IS_AUTHENTICATED_ANONYMOUSLY` to `IS_AUTHENTICATED_FULLY` you can ensure that the user is forced to authenticate (either via the login page or using non-interactive methods if they're setup) before they can get any further into the system.

#### Default authentication level in security.xml

```
<property name="objectDefinitionSource">
  <value>
    CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
    PATTERN_TYPE_APACHE_ANT
    /**=IS_AUTHENTICATED_ANONYMOUSLY
  </value>
</property>
```

This section can even be extended to provide different levels of default authentication for different client configurations, for example to ensure the user logs in to access `main.html` but allow anonymous access to everything else (this doesn't mean an anonymous user can access every other client config, just that that decision will be handled by the ACL's and not by `security.xml`) you could do the following.

#### Specific client access in security.xml

```

<property name="objectDefinitionSource">
  <value>
    CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
    PATTERN_TYPE_APACHE_ANT
    /main.html=IS_AUTHENTICATED_FULLY
    /**=IS_AUTHENTICATED_ANONYMOUSLY
  </value>
</property>

```

Client access with roles in security.xml

In addition to `IS_AUTHENTICATED_FULLY` and `IS_AUTHENTICATED_ANONYMOUSLY` you can also include roles that the user must have before they can access a given client configuration.

### Role based access via security.xml

```

<property name="objectDefinitionSource">
  <value>
    CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
    PATTERN_TYPE_APACHE_ANT
    /external.html=ROLE_EXTERNAL
    /main.html=IS_AUTHENTICATED_FULLY
    /**=IS_AUTHENTICATED_ANONYMOUSLY
  </value>
</property>

```

Here it's assumed that when logging in the authenticated user will be granted the role `ROLE_EXTERNAL`, which would then allow them to access the `/weave/external.html` URL.

Note that the equivalent can be done in using ACL's attached to a client configuration.

### Role based access via acl

```

<acl:acl id="external">
  <entry type="allow">ROLE_EXTERNAL</entry>
  <entry type="deny">*</entry>
</acl:acl>

<client:config id="external">
  <acl>external</acl>
  <title>My External Client</title>
  <!-- other config content goes here -->
</client:config>

```

There's one difference between using these above two methods that restrict access to a client configuration (assuming that in the `security.xml` version the client config doesn't also have an ACL attached, if it did then they're exactly the same), and that's when it comes to the display of the client list page.

The `security.xml` version will display the external client as being available to the user (because it doesn't have an ACL attached,

meaning that the user won't fail to pass the ACL checks for that particular client) but when the user attempts to open the page they'll either be asked to login, if they're an anonymous user or be denied access if they're not.

So this generally means that the ACL methods is more user friendly in an environment where the most of the authentication is being handled by Weave, whereas the `security.xml` methods can be more useful in environments where security is also being handled elsewhere and they user is never intended to see the listing page.

## Filter chains

In addition to the `FilterSecurityInterceptor` `objectDefinitionSource` there's another location where the client URL can play a part in authentication in `security.xml`, and that's in the `FilterChainProxy`.

The filter chain proxy determines what steps will be taken when an browser request passes through the security system. By default Weave is configured with two different filter chains, one for requests from the JavaScript code in the client and one for every other request. This is primarily done for error handling, where we want any errors to be presented to the user using a nice HTML page, but error in response to requests made by the JavaScript code should be in a format that it can interpret.

### Default security filters

```
<bean id="filterChainProxy" class="org.acegisecurity.util.
FilterChainProxy">
  <property name="filterInvocationDefinitionSource">
    <value>
      CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
      PATTERN_TYPE_APACHE_ANT
      /server/**=httpSessionContextIntegrationFilter,
authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,jsonExceptionTranslationFilter,
filterInvocationInterceptor
      /**=httpSessionContextIntegrationFilter,logoutFilter,
authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,exceptionTranslationFilter,
filterInvocationInterceptor
    </value>
  </property>
</bean>
```

Here we can see the default filter chains, with the `/server/*` handling requests from the JavaScript code, and the `/*` chain handling everything else.

This list can be supplemented to perform different actions depending upon what filters a request should be processed by based on the URL of the request, and in this case we're talking about client configuration access URL's, so we can adjust the list to include a different filter chain for a different client

### Default security filters

```
<bean id="filterChainProxy" class="org.acegisecurity.util.
FilterChainProxy">
  <property name="filterInvocationDefinitionSource">
    <value>
      CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
      PATTERN_TYPE_APACHE_ANT
      /server/**=httpSessionContextIntegrationFilter,
```

```

authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,jsonExceptionTranslationFilter,
filterInvocationInterceptor
    /internal.html=httpSessionContextIntegrationFilter,
ntlmProcessingFilter,authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,exceptionTranslationFilter,
filterInvocationInterceptor
    /**=httpSessionContextIntegrationFilter,logoutFilter,
authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,exceptionTranslationFilter,
filterInvocationInterceptor
    </value>
</property>
</bean>

```

Here we've added the `ntlmProcessingFilter` to the filter chain for any requests to access `/weave/internal.html`.


Additional filter chains can be added for other client configurations.

Windows Security

### Windows Authentication

Implementing Windows security generally involves two processes, determining who the user is using their Windows userid then using information in Active Directory to determine what roles a user has.

The first process can be implemented independently of the second, that is a user can be identified by their Windows userid but their roles can be read from a database or the user.properties file rather than from Active Directory.

-  The authentication of the Windows user information is performed using the SMB protocol. Access to the role information in Active Directory is performed using the LDAP protocol.

The configuration for Windows and Active Directory authentication is done via the `security.xml` file as all the authentication is.

First up we'll look at how to provide the user with access to the system without having to enter a username/password via Windows integrated authentication. Then we'll look at extending this to also obtain the access levels for the users from the domain.

-  The latest windows authentication bundle is [downloadable here](#)

## Debugging

You may want to turn on the logging of the security processing during the setting up of the authentication, since it'd disabled by default.

To do this ensure that the following two lines appear in `logging.properties` and any others referencing security are removed (in case they reduce the logging)

### Enable security process logging

```

log4j.logger.org.acegisecurity=DEBUG
log4j.logger.com.cohga.server.security=DEBUG

```

Then when someone is logging in the log file will show the progress and what roles they were granted, which should help you to understand what role names must be used in the access control lists

## Integrated Authentication



To implement Windows integrated authentication and allow internal users to login to Weave automatically using their Windows userid involves editing the `security.xml` file to replace the default login form with handling from the NTLM processor.

Looking at the default `security.xml` file it contains the following near the top:

### Default request filter chain

```

    <bean id="filterChainProxy" class="org.acegisecurity.util.
FilterChainProxy">
        <property name="filterInvocationDefinitionSource">
            <value>

CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
                PATTERN_TYPE_APACHE_ANT
                /server

/**=httpSessionContextIntegrationFilter,
authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,jsonExceptionTranslationFilter,
filterInvocationInterceptor

/**=httpSessionContextIntegrationFilter,logoutFilter,
authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,exceptionTranslationFilter,
filterInvocationInterceptor
            </value>
        </property>
    </bean>

```

What this section does is to determine which filters are applied to any incoming requests, passing everything that matched `/server/**` through the first list, and everything else through the second.

What we want to do to enable Windows authentication is add an additional filter to perform the NTLM authentication steps when required.

That should be the final new section we need to add, since the sections that it references should already exist. So all that remains is to add the first section we added to the list of filters:

### Adding NTLM filter to request filter chain

```

    <bean id="filterChainProxy" class="org.acegisecurity.util.
FilterChainProxy">
        <property name="filterInvocationDefinitionSource">
            <value>

CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
                PATTERN_TYPE_APACHE_ANT
                /server

/**=httpSessionContextIntegrationFilter,ntlmProcessingFilter,
authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,jsonExceptionTranslationFilter,
filterInvocationInterceptor
            </value>
        </property>
    </bean>

```

```

/**=httpSessionContextIntegrationFilter,ntlmProcessingFilter,
logoutFilter,authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,exceptionTranslationFilter,
filterInvocationInterceptor
        </value>
    </property>
</bean>

```

Here we've added the `ntlmProcessingFilter` to the list of filters that will be applied to the incoming requests.

Now we need to create the `ntlmProcessingFilter` filter and configure it to use the local domain.

To do this we add a new section to the `security.xml` file to define a new `NtlmProcessingFilter`

### The root NTLM filter definition

```

<bean id="ntlmProcessingFilter" class="org.acegisecurity.ui.
ntlm.NtlmProcessingFilter">
    <property name="defaultDomain"><value>DOMAINNAME<
/value></property>
    <property name="domainController"><value>172.16.0.30<
/value></property>
    <property name="authenticationEntryPoint" ref="
ntlmEntryPoint"/>
    <property name="authenticationManager" ref="
ntlmAuthenticationManager"/>
</bean>

```

The two values in there `DOMAINNAME` and `172.16.0.30` need to be replaced with values that are appropriate for your environment.

This is the filter that we added to the filter list, now we need to create two more sections that this filter references, the `ntlmEntryPoint` and the `ntlmAuthenticationManager`.

The `ntlmEntryPoint` takes care of the communication for the server to obtain the Windows userid from the browser.

The `ntlmAuthenticationManager` then sends that information through a list of `AuthenticationProviders` to validate that it's correct.

### NTLM entry point for obtaining the user information from browser

```

<bean id="ntlmEntryPoint" class="org.acegisecurity.ui.ntlm.
NtlmProcessingFilterEntryPoint"/>

```

### NTLM authentication manager for verifying the user information from browser

```

<bean id="ntlmAuthenticationManager" class="org.acegisecurity.
providers.ProviderManager">
    <property name="providers">
        <list>

```


```

        <ref local="smbAuthenticationProvider"
/>
        <bean class="org.acegisecurity.
providers.anonymous.AnonymousAuthenticationProvider">
            <property name="key" value="
changeThis" />
        </bean>
        <bean class="org.acegisecurity.
providers.rememberme.RememberMeAuthenticationProvider">
            <property name="key" value="
changeThis" />
        </bean>
    </list>
</property>
</bean>

```

The `ntlmEntryPoint` does not require any configuration, but we can see that the `ntlmAuthenticationManager` references yet another item that we need to add, the `smbAuthenticationManager`.

The `smbAuthenticationManager` provides authentication via the SMB protocol of the authentication information extracted by the `NtlmProcessingFilter`.

 The `AnonymousAuthenticationProvider` and `RememberMeAuthenticationProvider` ensure that anonymous users can connect and users that have indicated that they want to be remembered (if the login form is used) can be logged in from the cookie that was previously set. If neither of these things are applicable then it's possible to remove these 2 authentication providers and rely on the `smbAuthenticationProvider`.

Setting up the `smbAuthenticationProvider` is just a matter of configuring the `SmbNtlmAuthenticationProvider` with the `authorizationProvider` provider to be used.

### A SMB NTLM aware authentication provider

```

<bean id="smbAuthenticationProvider" class="org.acegisecurity.
providers.smb.SmbNtlmAuthenticationProvider">
    <property name="authorizationProvider">
        <ref local="nullDaoAuthenticationProvider" />
    </property>
</bean>

```

In this case we're referencing yet another item, the `nullDaoAuthenticationProvider` authentication provider.

The `nullDaoAuthenticationProvider` is a simple authentication provider that uses a separate `UserDetailsService` to retrieve the information about what roles a user has, and if you're using the default `security.xml` file for this that will be the `users.properties` file. Alternatively the `UserDetailsService` could be accessing a database to retrieve the users roles, and later we'll be looking at changing this to use Active Directory (via LDAP) to determine the users roles.

### A SMB simple password authenticator

```

<bean id="nullDaoAuthenticationProvider" class="org.
acegisecurity.providers.smb.NullPasswordDaoAuthenticationProvider">
    <property name="userDetailsService" ref="
userDetailsService" />
    <property name="userCache">
        <bean class="org.acegisecurity.providers.dao.

```

```

cache.EhCacheBasedUserCache">
    <property name="cache">
        <bean class="org.
springframework.cache.ehcache.EhCacheFactoryBean">
            <property name="
cacheManager">
                <bean class="
org.springframework.cache.ehcache.EhCacheManagerFactoryBean"/>
            </property>
            <property name="
cacheName" value="userCache"/>
        </bean>
    </property>
</bean>
</property>
</bean>

```

If you're starting with the default `security.xml` file that should be the final new section we need to add, since the `userDetailsService` that it references should already exist. And you could restart the server and assuming that the `users.properties` file has an entry for each user they should be able to log in without having to enter a username/password.

#### Customising the the SMB authentication process

Depending upon the version of active directory you're running you may need to specify a username/password for the `ntlmProcessingFilter`, so if you find authentication errors in the `weave.log` file after enabling integrated authentication then change the `ntlmProcessingFilter` to the following and set the appropriate username/password.

#### Setting username/password for domain access

```

<bean id="ntlmProcessingFilter" class="org.acegisecurity.ui.
ntlm.NtlmProcessingFilter">
    <property name="defaultDomain"><value>DOMAINNAME<
/value></property>
    <property name="domainController"><value>172.16.0.30<
/value></property>
    <property name="authenticationEntryPoint" ref="
ntlmEntryPoint"/>
    <property name="authenticationManager" ref="
ntlmAuthenticationManager"/>
    <property name = "JCifsProperties">
        <map>
            <entry key="jcifs.smb.client.username"
>
                <value>username</value>
            </entry>
            <entry key="jcifs.smb.client.password"
>
                <value>password</value>
            </entry>
        </map>
    </property>
</bean>


```

Additional properties that can effect the SMB authentication process can be found [here](#).

#### Selectively applying NTLM authentication

You can specify what IP addresses you want NTLM authentication to apply to, or not apply to, allowing you to support NTLM authentication for internal users and bypass it for external ones, for example (this prevents external users from being presented with a username/password dialogue box that they will probably not have valid values for).

To do this you need to replace the `ntlmProcessingFilter`, rather than using the `org.acegisecurity.ui.ntlm.NtlmProcessingFilter` class you should use the `org.acegisecurity.ui.ntlm.IPFilteredNtlmProcessingFilter`, this implementation of the `NtlmProcessingFilter` can then be provided with additional configuration items specifying which IP addresses should/shouldn't be provided with the option to authenticate using NTLM.

 All the previous configuration items still apply, and should still be set, for the `IPFilteredNtlmProcessingFilter`. This new version just provides additional configuration options.

The new configuration items that the `IPFilteredNtlmProcessingFilter` provides are `excludedIpAddresses` and `includedIpAddresses`, and are set as a list of IP addresses or address ranges.

#### Selectively applying NTLM authentication

```
<bean id="ntlmProcessingFilter" class="org.acegisecurity.ui.
ntlm.IPFilteredNtlmProcessingFilter">
    <property name="defaultDomain"><value>DOMAINNAME<
/value></property>
    <property name="domainController"><value>172.16.0.30<
/value></property>
    <property name="authenticationEntryPoint" ref="
ntlmEntryPoint"/>
    <property name="authenticationManager" ref="
ntlmAuthenticationManager"/>
    <property name="excludedIpAddresses">
        <list>
            <value>192.168.2.0/24</value>
            <value>138.19.19.50</value>
        </list>
    </property>
    <property name="includedIpAddresses">
        <list>
            <value>172.16.0.0/16</value>
        </list>
    </property>
</bean>
```

You don't need to provide both `excludedIpAddresses` and `includedIpAddresses`, in fact it's more than likely that you'll only want to provide one, either listing those addresses that should be NTLM authenticated, and everyone else isn't, or listing those addresses that should not be NTLM authenticated and everyone else should. But, if you do provide both then the exclude list is checked first. Also, if the include list is set then the IP address must appear in the list for NTLM authentication to be attempted.

 The `IPFilteredNtlmProcessingFilter` class is provided in version 1.0.7 or later of the `org.acegisecurity.ntlm` bundle

As of version 1.3.4 of the `org.acegisecurity.ntlm` bundle there's an additional property that can be set for the `IPFilteredNtlmProcessingFilter`, and that's `defaultRole`, which when set will add the role (exactly as it appears in the `security.xml` file) to the list of roles the user has. This allows you to utilise multiple Active Directory domain to authenticate user and provide access control based on what domain the user was authenticated against.

Note: If you're using LDAP to provide the users roles then it's also possible to set a `defaultRole` in the LDAP populator.

## Using multiple domain for authentication

```

    <bean id="ntlmProcessingFilterInternal" class="org.
acegisecurity.ui.ntlm.IPFilteredNtlmProcessingFilter">
        <property name="defaultDomain"><value>INTERNAL<
/value></property>
        <property name="domainController"><value>172.16.0.30<
/value></property>
        <property name="authenticationEntryPoint" ref="
ntlmEntryPoint"/>
        <property name="authenticationManager" ref="
ntlmAuthenticationManager"/>
        <property name="includedIpAddresses">
            <list>
                <value>172.16.0.0/16</value>
            </list>
        </property>
        <property name="defaultRole"><value>ROLE_INTERNAL<
/value></property>
    </bean>

    <bean id="ntlmProcessingFilterExternal" class="org.
acegisecurity.ui.ntlm.IPFilteredNtlmProcessingFilter">
        <property name="defaultDomain"><value>EXTERNAL<
/value></property>
        <property name="domainController"><value>201.20.109.76
</value></property>
        <property name="authenticationEntryPoint" ref="
ntlmEntryPoint"/>
        <property name="authenticationManager" ref="
ntlmAuthenticationManager"/>
        <property name="includedIpAddresses">
            <list>
                <value>201.20.0.0/16</value>
            </list>
        </property>
        <property name="defaultRole"><value>ROLE_EXTERNAL<
/value></property>
    </bean>

```

Not that to enable this both filters need to be added to the filter chain:

## Adding NTLM filter to request filter chain

```

    <bean id="filterChainProxy" class="org.acegisecurity.util.
FilterChainProxy">
        <property name="filterInvocationDefinitionSource">
            <value>
CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON

```

```

        PATTERN_TYPE_APACHE_ANT
        /server
        /**=httpSessionContextIntegrationFilter,ntlmProcessingFilter1,
        ntlmProcessingFilter2,authenticationProcessingFilter,
        securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
        anonymousProcessingFilter,jsonExceptionTranslationFilter,
        filterInvocationInterceptor

        /**=httpSessionContextIntegrationFilter,ntlmProcessingFilter1,
        ntlmProcessingFilter2,logoutFilter,authenticationProcessingFilter,
        securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
        anonymousProcessingFilter,exceptionTranslationFilter,
        filterInvocationInterceptor
        </value>
    </property>
</bean>

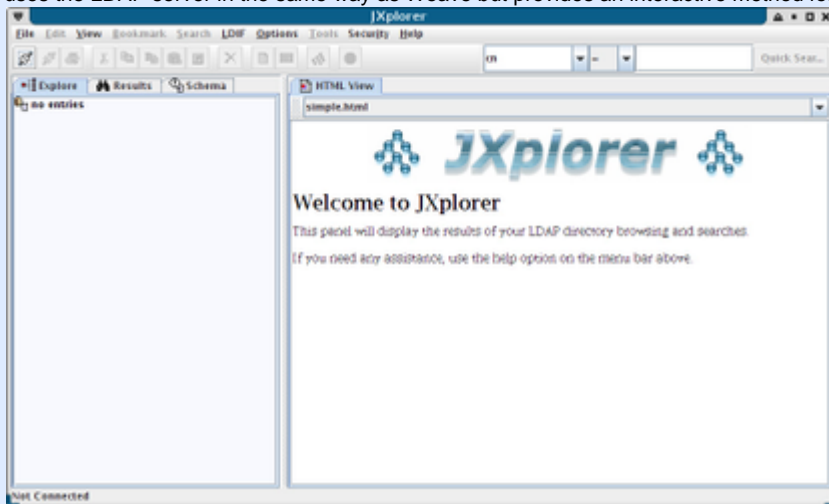
```

## Active Directory Groups

Information about what active directory groups a user belongs to can be used to provide role information to Weave for the users that are authenticated using Windows integrated authentication, removing the need to utilise the `users.properties` file.

This information is obtained from an AD domain controller using the LDAP protocol.

- i** When setting up the LDAP integration it's recommended that the [JXplorer](#) tool be used to test the settings, because the JXplorer uses the LDAP server in the same way as Weave but provides an interactive method for verifying the settings.



To enable LDAP as a source of authentication information the `ntlmAuthenticationManager` we created earlier needs to be altered to use an `LdapAuthenticationProvider` rather than the `SmbAuthenticationProvider`.

So we first need to change the `ntlmAuthenticationManager` to

### Enabling LDAP support for authenticating users

```

    <bean id="ntlmAuthenticationManager" class="org.acegisecurity.
    providers.ProviderManager">
        <property name="providers">
            <list>
                <ref local="
    ldapAuthenticationProvider"/>

```

```

                <bean class="org.acegisecurity.
providers.anonymous.AnonymousAuthenticationProvider">
                    <property name="key" value="
changeThis" />
                </bean>
                <bean class="org.acegisecurity.
providers.rememberme.RememberMeAuthenticationProvider">
                    <property name="key" value="
changeThis" />
                </bean>
            </list>
        </property>
    </bean>

```

And then setup the new `LdapAuthenticationProvider` as follows:


### LDAP authentication provider that can handle NTLM authenticated users

```

    <bean id="ldapAuthenticationProvider" class="org.
acegisecurity.ui.ntlm.ldap.authenticator.
NtlmAwareLdapAuthenticationProvider">
        <constructor-arg>
            <ref local="authenticatorLdap" />
        </constructor-arg>
        <constructor-arg>
            <ref local="populatorLdap" />
        </constructor-arg>
    </bean>

```

This provider uses two other item to provide information, the `authenticatorLdap` bean and the `populatorLdap` bean.

 We use the `NtlmAwareLdapAuthenticationProvider` here because the user has been authenticated using their windows userid. If you're not using windows integrated authentication you can still LDAP to provide the roles for a user, but in that case you'd use the `LdapUserDetailsService` which isn't covered here.

The authentication would be configured as follows:

### The authenticator that will search an LDAP directory for the user

```

    <bean id="authenticatorLdap" class="org.acegisecurity.ui.ntlm.
ldap.authenticator.NtlmAwareLdapAuthenticatorImpl">
        <constructor-arg>
            <ref local="initialDirContextFactory" />
        </constructor-arg>
        <property name="userSearch">
            <ref local="userSearchLdap" />
        </property>
    </bean>

```



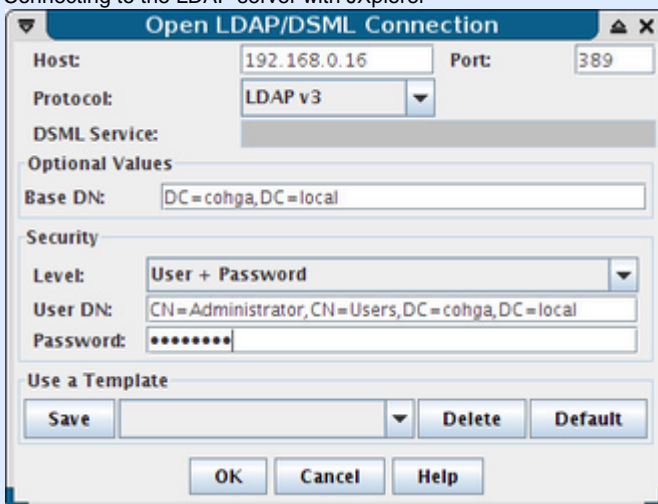
Which as we can see requires two other items, the `initialDirContextFactory` and the `userSearchLdap`.

The `initialDirContextFactory` is also used by the `userSearchLdap` and the `populaterLdap` beans so we'll look at that first

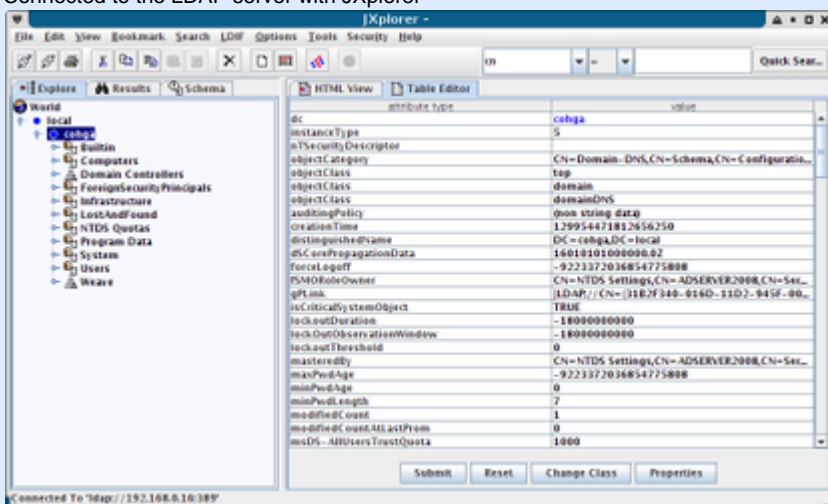
### Setting up a connection to the LDAP server

```
<bean id="initialDirContextFactory" class="org.acegisecurity.
ldap.DefaultInitialDirContextFactory">
  <constructor-arg value="ldap://192.168.0.16:389/" />
  <property name="managerDn">
    <value>CN=Administrator,OU=Users,DC=cohga,
DC=local</value>
  </property>
  <property name="managerPassword">
    <value>password</value>
  </property>
</bean>
```

#### Connecting to the LDAP server with JXplorer




#### Connected to the LDAP server with JXplorer

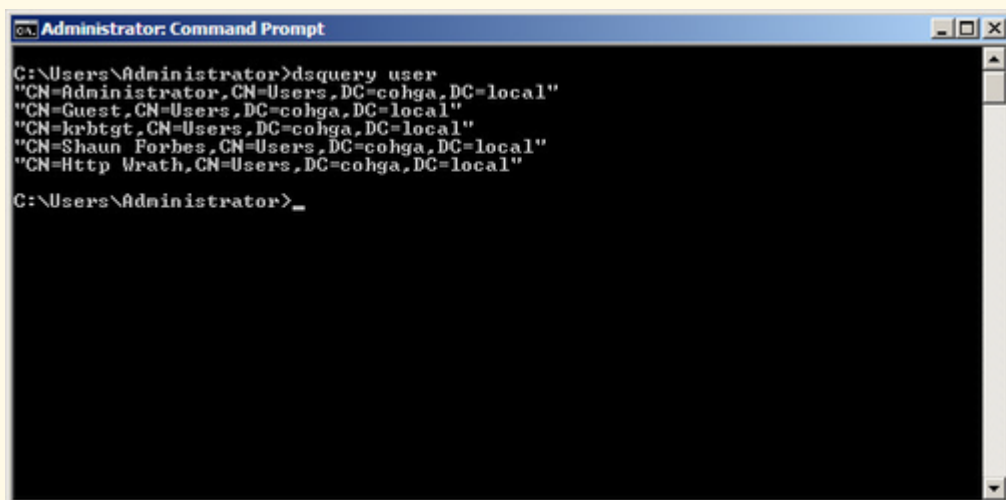


Here the ip address, manager distinguished name and manager passwords must be set to appropriate values for a user that can read for the active directory server.

I've used the Administrator user to connect to LDAP in the example above but you do not want to do that on a production system (and

you should be worried if your IT department gives you the password for this user just to do this). Instead you should have IT create a limited user account that you can connect with.

 The `dsquery.exe` program can be used to find the distinguished name of the user: `dsquery user`



```

CA, Administrator: Command Prompt
C:\Users\Administrator>dsquery user
"CN=Administrator,CN=Users,DC=cohga,DC=local"
"CN=Guest,CN=Users,DC=cohga,DC=local"
"CN=krbtgt,CN=Users,DC=cohga,DC=local"
"CN=Shaun Forbes,CN=Users,DC=cohga,DC=local"
"CN=Http Wrath,CN=Users,DC=cohga,DC=local"
C:\Users\Administrator>_

```

The two final beans, the `userSearchLdap` and `populatorLdap` also require information that is specific to the environment you're running within, the `userSearchLdap` beans would be something like the following:

#### Setting up an LDAP search for a user

```

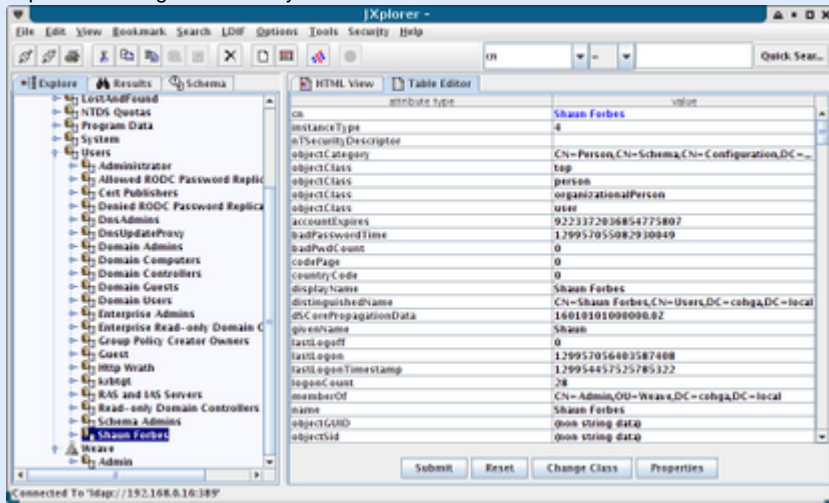
<bean id="userSearchLdap" class="org.acegisecurity.ldap.
search.FilterBasedLdapUserSearch">
  <constructor-arg>
    <value>OU=Users,DC=cohga,DC=local</value>
  </constructor-arg>
  <constructor-arg>
    <value>(sAMAccountName={0})</value>
  </constructor-arg>
  <constructor-arg>
    <ref local="initialDirContextFactory" />
  </constructor-arg>
  <property name="searchSubtree">
    <value>>true</value>
  </property>
</bean>

```

This configuration assumes that there is a branch in the tree matching the first constructor arg and that the `sAMAccountName` value of any users found there will match the username they logged into Windows with.



JXplorer showing a users entry

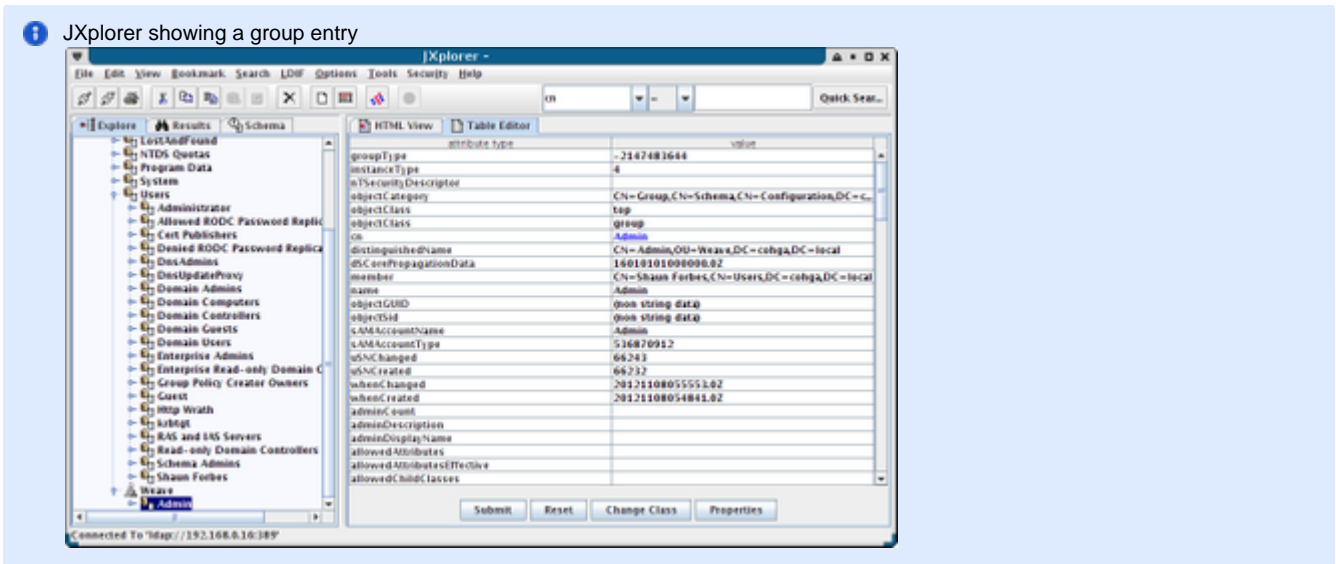


Finally the `populatorLdap` is responsible for mapping the username to the roles and would be configured as follows

### Setting up an LDAP search for groups

```
<bean id="populatorLdap" class="org.acegisecurity.providers.
ldap.populator.DefaultLdapAuthoritiesPopulator">
  <constructor-arg>
    <ref local="initialDirContextFactory"/>
  </constructor-arg>
  <constructor-arg>
    <value>OU=Weave,DC=cohga,DC=local</value>
  </constructor-arg>
  <property name="groupRoleAttribute">
    <value>cn</value>
  </property>
  <property name="searchSubtree">
    <value>true</value>
  </property>
  <property name="rolePrefix">
    <value>ROLE_</value>
  </property>
  <property name="convertToUpperCase">
    <value>true</value>
  </property>
  <property name="groupSearchFilter">
    <value>(member={0})</value>
  </property>
  <property name="defaultRole">
    <value>ROLE_USERS</value>
  </property>
</bean>
```

This configuration is setup to search for groups and use the `member` attribute of the group to determine if the user belongs. This setup is assuming that there has been a security group created in the Active Directory server called `Weave` and that the roles/groups for `Weave` are created under there. This "should" take the active directory groups that the user belongs to and convert them to a format that's usable in `Weave`, and also assigns a default `ROLE_USERS` role to all users (which you can remove if it's not appropriate).



You will then need to create Weave Access Control Lists utilising the roles that users will be assigned.

#### Supplementing LDAP Roles

As of version 1.4.4 of the `org.acegisecurity.ntlm` bundle it's possible to supplement the roles set via LDAP from a separate users details service (i.e. for example from a `user.properties` type file).

In this updated bundle there is a new `SupplementedLdapAuthoritiesPopulator` that can replace the `DefaultLdapAuthoritiesPopulator` in your `security.xml` and this authorities populator can be configured with an additional user details service, which if set will be used to look up the user being authenticated and if they're found the roles provided by the user details service will be added to the current user.

To implement the new authorities populator you should change:

#### Old DefaultLdapAuthoritiesPopulator

```
<bean id="populator" class="org.acegisecurity.providers.ldap.
populator.DefaultLdapAuthoritiesPopulator">
    ...
</bean>
```

to

#### New SupplementedLdapAuthoritiesPopulator with reference to user details service

```
<bean id="populator" class="org.acegisecurity.providers.ldap.
populator.SupplementedLdapAuthoritiesPopulator">
    <property name="userDetailsService" ref="ldapDetailsService"/>
    ...
</bean>
```

This will change it from `DefaultLdapAuthoritiesPopulator` to `SupplementedLdapAuthoritiesPopulator` and add a reference to the service that will provide the additional role information, which can be included by adding:

#### New user details service reading roles from a file

```

<bean id="ldapDetailsService" class="org.acegisecurity.userdetails.
memory.InMemoryDaoImpl">
    <property name="userProperties">
        <bean class="org.springframework.beans.factory.config.
PropertiesFactoryBean">
            <property name="location" value="ldap.
properties"/>
        </bean>
    </property>
</bean>

```

Then you can create the `ldap.properties`, which has the same format as the existing `users.properties` file, in the workspace directory and any user listed there will have those roles added. e.g.

```
shaun=password,ROLE_TEST
```

Note the password is ignored, we're just mapping the userid to the roles and adding those roles to the current user. There is currently no capability to remove roles or disable the users access using the `SupplementedLdapAuthoritiesPopulator`.

Alternatively you could also use the following if you just had a couple of users:

### New user details service reading roles from its configuration

```

<bean id="ldapDetailsService" class="org.acegisecurity.userdetails.
memory.InMemoryDaoImpl">
    <property name="userMap">
        <value>
            shaun=password,ROLE_TEST
        </value>
    </property>
</bean>

```

You could even use the `JdbcDaoSupport` class if you wanted to get this information from a database.

### Alternate Windows Security Setup

#### Description

Windows Server 2008 can have issues with the NTLM authentication method used with Weave. This page outlines a method of configuring the Weave authentication to work around this until the issue is resolved.

- ✘ The reason that this method works and the other doesn't is that this setup bypasses the checks on the credentials that the browser has submitted, since the new method used by Windows 2008 to validate the credentials is not yet supported. Obviously this is a security risk, and this method should be viewed as a convenience for the user rather than a method of securing access to sensitive information.

### Alternate Integrated Authentication

To implement Windows integrated authentication and allow internal users to login to Weave automatically using their Windows username involves editing the `security.xml` file to replace the login form with handling with the NTLM processor.

So looking at the default `security.xml` file it contains the following near the top

```

    <bean id="filterChainProxy" class="org.acegisecurity.util.
FilterChainProxy">
        <property name="filterInvocationDefinitionSource">

```

```

        <value>

CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
                PATTERN_TYPE_APACHE_ANT
                /server
/**=httpSessionContextIntegrationFilter,
authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,jsonExceptionTranslationFilter,
filterInvocationInterceptor

/**=httpSessionContextIntegrationFilter,logoutFilter,
authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,exceptionTranslationFilter,
filterInvocationInterceptor
                </value>
        </property>
</bean>

```

What this section does is to determine which filters are applied to any incoming requests, passing everything that matched `/server/**` through the first list, and everything else through the second.

What we want to do here is add an additional filter to perform the NTLM authentication steps when required. But before we add the new filter to the list we need to add the new definition for the filter object itself.

To do this we add a new section to the `security.xml` file to define a new `NtlmProcessingFilter`


```

        <bean id="ntlmProcessingFilter" class="org.acegisecurity.ui.
ntlm.NtlmProcessingFilter">
                <property name="defaultDomain"><value>DOMAINNAME<
/value></property>
                <property name="domainController"><value>172.16.0.30<
/value></property>
                <property name="authenticationEntryPoint" ref="
ntlmEntryPoint"/>
                <property name="authenticationManager" ref="
ntlmAuthenticationManager"/>
        </bean>

```

The two values in there `DOMAINNAME` and `172.16.0.30` need to be replaced with values that are appropriate for your environment.

This is the filter that we will eventually add to the filter list, but we first need to create two more sections that this references, the `ntlmEntryPoint` and the `ntlmAuthenticationManager`.

 This is the first place where this configuration is different from the original NTLM setup, namely the `nullDaoAuthentication` provider is used instead of the `smbAuthenticationProvider`.

```

        <bean id="ntlmEntryPoint" class="org.acegisecurity.ui.ntlm.
NtlmProcessingFilterEntryPoint"/>

        <bean id="ntlmAuthenticationManager" class="org.acegisecurity.

```

```

providers.ProviderManager">
    <property name="providers">
        <list>
            <ref local="
nullDaoAuthenticationProvider"/>
            <bean class="org.acegisecurity.
providers.anonymous.AnonymousAuthenticationProvider">
                <property name="key" value="
changeThis"/>
            </bean>
            <bean class="org.acegisecurity.
providers.rememberme.RememberMeAuthenticationProvider">
                <property name="key" value="
changeThis"/>
            </bean>
        </list>
    </property>
</bean>

```

The `ntlmEntryPoint` is pretty straight forward, but we can see that the `ntlmAuthenticationManager` references yet another section that we need to add, the `nullDaoAuthenticationManager`.

```

    <bean id="nullDaoAuthenticationProvider" class="org.
acegisecurity.providers.smb.NullPasswordDaoAuthenticationProvider">
        <property name="userDetailsService" ref="
userDetailsService"/>
        <property name="userCache">
            <bean class="org.acegisecurity.providers.dao.
cache.EhCacheBasedUserCache">
                <property name="cache">
                    <bean class="org.
springframework.cache.ehcache.EhCacheFactoryBean">
                        <property name="
cacheManager">
                            <bean class="
org.springframework.cache.ehcache.EhCacheManagerFactoryBean"/>
                        </property>
                        <property name="
cacheName" value="userCache"/>
                    </bean>
                </property>
            </bean>
        </property>
    </bean>

```

That should be the final new section we need to add, since the sections that it references should already exist. So all that remains is to add the first section we added to the list of filters

```

<bean id="filterChainProxy" class="org.acegisecurity.util.

```

```

FilterChainProxy">
    <property name="filterInvocationDefinitionSource">
        <value>

CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
        PATTERN_TYPE_APACHE_ANT
        /server

/**=httpSessionContextIntegrationFilter,ntlmProcessingFilter,
authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,jsonExceptionTranslationFilter,
filterInvocationInterceptor

/**=httpSessionContextIntegrationFilter,ntlmProcessingFilter,
logoutFilter,authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,
anonymousProcessingFilter,exceptionTranslationFilter,
filterInvocationInterceptor
        </value>
    </property>
</bean>

```

Here you can see that we've added the `ntlmProcessingFilter` to the two filter lists, this should allow used that are logged into Windows to have that username automatically recognized by Weave.

But that doesn't take care of determining what the user has access to once they're logged in. That will still need to be done via the `users.properties` file (to map usernames to roles), unless you setup `security.xml` to utilise information stored in Active Directory.

Depending upon the version of active directory you're running you may need to specify a username/password for the `ntlmProcessingFilter`, so if you find authentication errors in the `weave.log` file after enabling integrated authentication then change the `ntlmProcessingFilter` to the following and set the appropriate username.password.

```

    <bean id="ntlmProcessingFilter" class="org.acegisecurity.ui.
ntlm.NtlmProcessingFilter">
        <property name="defaultDomain"><value>DOMAINNAME<
/value></property>
        <property name="domainController"><value>172.16.0.30<
/value></property>
        <property name="authenticationEntryPoint" ref="
ntlmEntryPoint"/>
        <property name="authenticationManager" ref="
ntlmAuthenticationManager"/>
        <property name = "JCifsProperties">
            <map>
                <entry key="jcifs.smb.client.username"
>
                    <value>username</value>
                </entry>
                <entry key="jcifs.smb.client.password"
>
                    <value>password</value>
                </entry>
            </map>

```




```

        </property>
    </bean>

```

## Windows Authentication Examples

 The latest windows authentication bundle is [available here](#)

### First step to enable integrated login

**Download:** [security\\_ntlm\\_step\\_1.xml](#)

This example contains the bare minimum to enable Windows Integrated Authentication and should be used as a first step to test the authentication.

It does not provide access to Weave at all if the user is not part of the domain or if they are not listed in the `users.properties` file.

It requires that each user that will be given access to the system be listed in the `users.properties` file, but since this is for testing that should only be one or two users.

It will require editing of the `ntlmProcessingFilter` bean to at least set the correct `defaultDomain` and `domainController`. It may also require changing the `loadBalance` property to set it to `true` (`false` is the default if it's not set).

Depending upon the version of active directory you're running you may need to specify a `username/password` for the `ntlmProcessingFilter`.

If you find authentication errors in the `weave.log` file after enabling integrated authentication then change the `ntlmProcessingFilter` to the following and set the appropriate `username/password`.

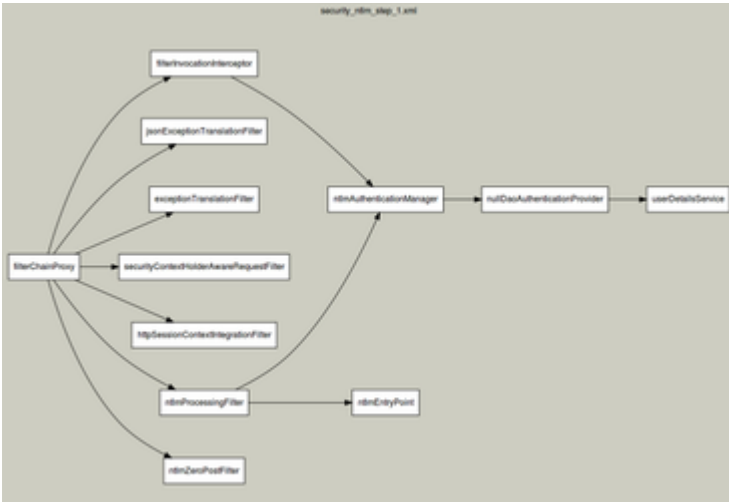
#### Setting username/password for domain access

```

    <bean id="ntlmProcessingFilter" class="org.acegisecurity.ui.
ntlm.NtlmProcessingFilter">
        <property name="authenticationEntryPoint" ref="
ntlmEntryPoint"/>
        <property name="authenticationManager" ref="
ntlmAuthenticationManager"/>
        <property name="defaultDomain">
            <value>cohga.local</value>
        </property>
        <property name="domainController">
            <value>192.168.0.80</value>
        </property>
        <property name = "JCifsProperties">
            <map>
                <entry key="jcifs.smb.client.username"
>
                    <value>username</value>
                </entry>
                <entry key="jcifs.smb.client.password"
>
                    <value>password</value>
                </entry>
            </map>
        </property>
    </bean>

```

Additional properties that can effect the authentication process [can be found here](#), where they can be set in the `JCifsProperties` section to alter the authentication process.



### Extended authentication example

Download: [security\\_ntlm\\_step\\_2.xml](#)

This example expands on the original bare minimum example but includes anonymous authentication plus form login.

It will also require editing of the ntlmProcessingFilter bean to at least set the correct defaultDomain and domainController. It may also require changing the loadBalance property to set it to true (false is the default if it's not set).



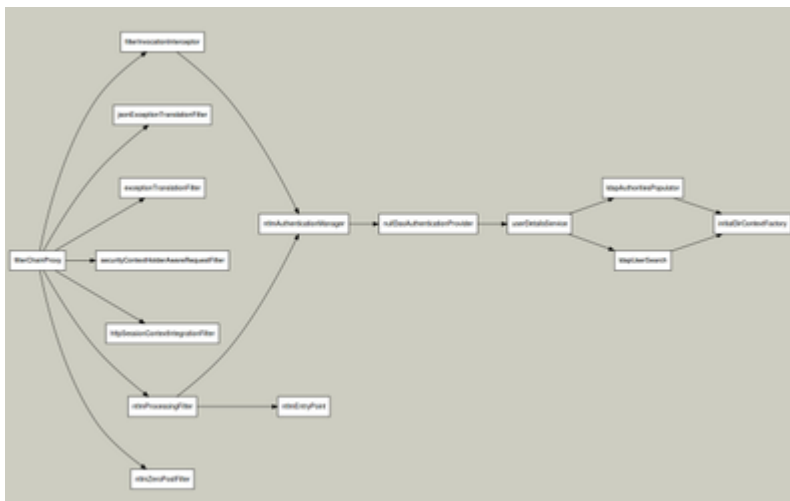
### Getting roles from Active Directory (via LDAP)

Download: [security\\_ntlm\\_step\\_3.xml](#)

In this example we're going back to step 1 but instead of obtaining the user information from the users.properties file, via the org.acegisecurity.userdetails.memory.InMemoryDaoImpl user details service, we'll access the information from Active Directory (via the LDAP protocol).

This example is exactly the same as step one except we've swapped out the user details service that accesses users.properties for the one that accesses that information LDAP server.

The information setup in ldapUserSearch, ldapAuthoritiesPopulator and initialDirContextFactory will at least need to be updated to reflect your local settings.



## Database Security

This will explain how to authenticate users based on tables in a database.

Note: These examples require Weave 2.6.5 or later, but this can be done on earlier releases but it's not as easy.

```
// these are the default table names but can be changed via config
drop table authorities;
drop table users;

create table users (
    username varchar2(64) primary key,
    password varchar2(64) not null,
    enabled integer not null
);

create table authorities (
    username varchar2(64) not null,
    authority varchar2(64) not null,
    foreign key(username) references users(username)
);

// the password here is SHA-256 encoded
insert into users values
('sforbes', '680cef309f7c44fe9c7c615247c796165da38e21c5611db6555c2a6d64
f9e3a8', 1);
insert into authorities values('sforbes', 'ROLE_USER');
insert into authorities values('sforbes', 'ROLE_ADMIN');

insert into users values
('demo', '452bae0ece336e40ee13c59b96227f862041cee4ab9232cd9ed82dd84b008
818', 1);
insert into authorities values('demo', 'ROLE_USER');

commit;
```

The above shows table creation and adding a couple of example users

The following shows a user details configuration from security.xml



```

        <bean id="daoAuthenticationProvider" class="org.acegisecurity.
providers.dao.DaoAuthenticationProvider">
            <property name="userDetailsService" ref="
userDetailsService"/>
            <property name="passwordEncoder" ref="passwordEncoder"
/>
            <property name="saltSource" ref="saltSource"/>
            <property name="userCache">
                <bean class="org.acegisecurity.providers.dao.
cache.EhCacheBasedUserCache">
                    <property name="cache">
                        <bean class="org.
springframework.cache.ehcache.EhCacheFactoryBean">
                            <property name="
cacheManager">
                                <bean class="
org.springframework.cache.ehcache.EhCacheManagerFactoryBean"/>
                            </property>
                            <property name="
cacheName" value="userCache"/>
                        </bean>
                    </property>
                </bean>
            </property>
        </bean>

        <bean id="passwordEncoder" class="org.acegisecurity.providers.
encoding.ShaPasswordEncoder">
            <constructor-arg value="256"/>
        </bean>

        <bean id="saltSource" class="org.acegisecurity.providers.dao.
salt.SystemWideSaltSource">
            <property name="systemWideSalt">
                <value>Ha0%@ra2nA^ad</value>
            </property>
        </bean>

        <bean id="userDetailsService" class="org.acegisecurity.
userdetails.jdbc.JdbcDaoImpl">
            <property name="dataSource">
                <ref bean="authDataSource"/>
            </property>
        </bean>

        <bean id="authDataSource" class="com.cohga.beans.jdbc.
datasource.DriverManagerDataSource">
            <property name="driverClassName">
                <value>oracle.jdbc.OracleDriver</value>
            </property>
            <property name="url">
                <value>jdbc:oracle:thin:@devdb:1521:orcl<
/property>
            </property>
        </bean>

```

```
        <property name="username">
            <value>gis</value>
        </property>
        <property name="password">
            <value>ENCAHFUYADASHFDA</value>
        </property>
    </bean>

    <!-- alternatively you can use a datasource defined in config.xml
        <bean id="authDataSource" class="com.cohga.beans.jdbc.
datasource.WeaveDataSource">
            <property name="dataSource">
                <value>postgres</value>
            </property>
        </bean>
-->
```

## Logging

Logging in Weave is done using a third party library. Prior to Weave 2.6.7 [Log4j 1.2](#) was used. From 2.6.7 onwards [Logback](#) is used. It is beyond the scope of this document to explain how the logging infrastructure works with Log4j and Logback so the reader is encouraged to visit the Log4j or Logback websites to find out more information about the inner workings of these libraries.

The following links provide details about Logback.

*Main Site:*

<https://logback.qos.ch/>

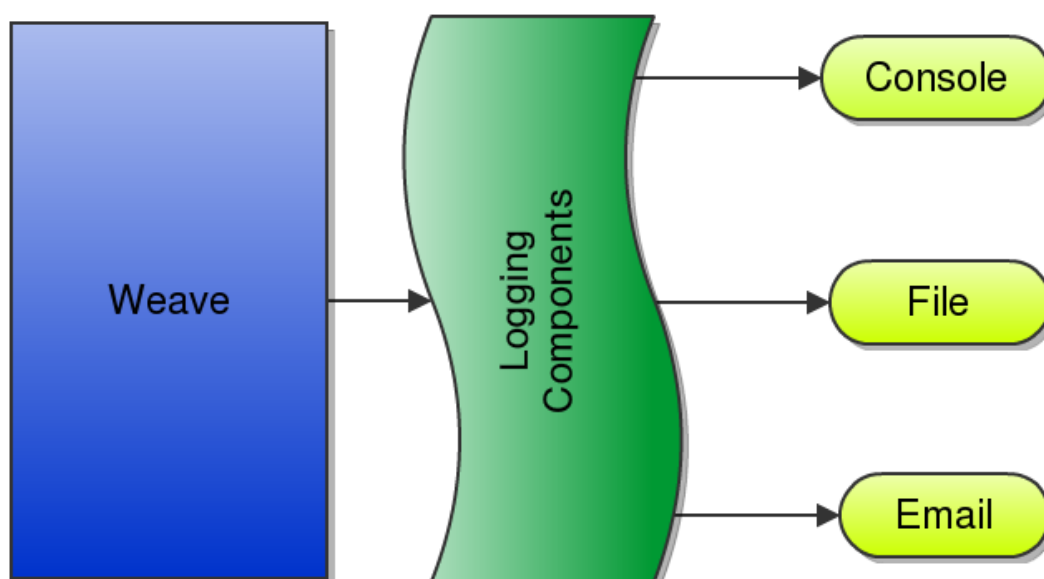
*Documentation:*

<https://logback.qos.ch/manual/index.html>

## Architecture

From an architectural point of view, software application modules and logging components reside in two separate layers. The application makes a call to the logging components in the logging layer and delegates the logging responsibility to those components. The logging components receive the logging request and publish the logging information at preferred destinations.

The figure below represents the collaboration of a software module and its logging components.



The architecture of the Logback API is layered. Each layer consists of different objects performing different tasks. The top layer captures the logging information, the middle layer is involved in analyzing and authorizing the logging information, and the bottom layer is responsible for formatting and publishing the logging information to a destination. In essence, Logback consists of three types of primary objects:

- **Logger:** The *Logger* object is responsible for capturing logging information. Logger objects are stored in a namespace hierarchy.
- **Appender:** The *Appender* object is responsible for publishing logging information to various preferred destinations. Each Appender object will have at least one target destination attached to it. For example, a `ConsoleAppender` object is capable of printing logging information to a console. <https://logback.qos.ch/manual/appenders.html>
- **Layout:** The *Layout* object is used to format logging information in different styles. Appender objects utilize Layout objects before publishing logging information. Layout objects play an important role in publishing logging information in a way that is human-readable and reusable. <https://logback.qos.ch/manual/layouts.html>

The preceding core objects are central to the architecture of Logback. Apart from them, there are several auxiliary objects that can plug and play to any layer of the API. These objects help manage the different Logger objects active within an application and fine-tune the logging process.

The principal auxiliary components in the Logback framework that play a vital role in the logging framework are:

- **Level:** The *Level* object defines the granularity and priority of any logging information. Each piece of logging information is accompanied by its appropriate Level object, which tells the Logger object about the priority of the information. The levels of logging defined within the API are: TRACE, DEBUG, INFO, WARN and ERROR. The Level values can be arranged in an ascending manner as:  
TRACE < DEBUG < INFO < WARN < ERROR  
During normal operation of Weave you're likely to only require log messages with a level of INFO or higher (WARN and ERROR) which will tell you what Weave is doing and if there are any problems. But when you're trying to figure out a problem, or are actively configuring the system, then seeing the log level of DEBUG can also be useful. It's unlikely that a Weave instance would require a log level of TRACE. This log level produces a large amount of low level detail and is mainly intended to help developers troubleshoot issues with the Weave code.  
Additionally, when filtering log output, you can specify OFF as a log level to disable all log output from a specific logger. For example a number of third party libraries used by Weave also produce log output but this can be unnecessary noise when trying to view the Weave log output, so the default Weave log configuration disables the log output from a number of these libraries by setting their log level to OFF (or in some cases ERROR or WARN).
- **Filter:** The *Filter* object is used to analyze logging information and make further decisions on whether that information should be logged or not. In the Logback context, Appender objects can have several Filter objects associated with them. If logging information is passed to a particular Appender object, all the Filter objects associated with that Appender need to approve the logging information before it can be published to the preferred destination attached to the Appender. Filter objects are very helpful in filtering out unwanted logging information based on any application-specific criteria.
- **Pattern:** The *Pattern* object is specialized in providing a string representation of different objects passed to the logging framework. More precisely, when the application passes a custom Object to the logging framework, the logging framework will use the corresponding Pattern to obtain a string representation of the passed Object. This is used by Layout objects to prepare the final logging information.

## Configuring Logging

The configuration below defines the level of the root logger as `DEBUG` and specifies the appender to use as `testAppender`. Next, the appender `testAppender` is defined as referencing the `ch.qos.logback.core.ConsoleAppender` object and the layout of the appender as the pattern is specified. A pattern requires that a conversion pattern or a format be supplied. In this case the conversion pattern of `%m%n` is supplied, which means the logging message will be printed followed by a newline character. More information about these patterns are available at <https://logback.qos.ch/manual/layouts.html#conversionWord>

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <appender name="testAppender" class="ch.qos.logback.core.
ConsoleAppender">
        <encoder>
            <pattern>%message%n</pattern>
        </encoder>
    </appender>

    <root level="DEBUG">
        <appender-ref ref="testAppender" />
    </root>

</configuration>
```

A more complex configuration can attach multiple appenders to a particular logger. Each appender, in turn, can have a different layout, and that layout can have a conversion pattern associated with it.

Below is an example of a more complex configuration file. This configuration file defines the root logger as having level `DEBUG` and attaches two appenders named `console` and `file` to it.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

    <appender name="console" class="ch.qos.logback.core.
ConsoleAppender">
        <encoder>
            <pattern>%message%n</pattern>
        </encoder>
    </appender>

    <appender name="file" class="ch.qos.logback.core.rolling.
RollingFileAppender">
        <file>logs/weave.log</file>
        <encoder>
            <pattern>%date{yyyy-MM-dd HH:mm:ss.SSS} %
level [%thread] %logger: "%message"%n</pattern>
        </encoder>
        <rollingPolicy class="ch.qos.logback.core.rolling.
TimeBasedRollingPolicy">
            <fileNamePattern>logs/weave-%d{yyyy-MM-dd}.
log</fileNamePattern>
            <maxHistory>10</maxHistory>
        </rollingPolicy>
    </appender>
```

```

<root level="DEBUG">
    <appender-ref ref="console"/>
    <appender-ref ref="file"/>
</root>

</configuration>

```

The default `logging.xml` file included with Weave is similar to the above example but has some additional configuration that enables the different types of log output depending upon how Weave was started. For example, if you use `startup.cmd` to start Weave you only get the console output, but if you started Weave as a service then you only get the `weave.log` file output. This is done by including conditions around the `appender-ref` tags in the `root` section of the file, but if you want to explicitly determine which log output is generated you can remove those additional condition tags and have the same output regardless of how Weave was started.

Note that the default log level set in `logging.xml` does not effect the log information recorded in a Support Dump. A Support Dump generated from Weave will now always include all log messages generated during startup of Weave and all log messages generated (up to a certain number) before the dump was produced regardless of the log level set in `logging.xml`.

## Reading Log Files

When Weave has been installed without any modifications to the `logging.xml` file, Weave logs the following information to the `weave.log` file within the `$WEAVE_HOME/logs/` directory when started as a service/daemon. The `weave.log` file contains the current log messages that Weave has produced in the following format.

```
TIME LOGGING_LEVEL THREAD USER_ID SESSION_ID MESSAGE_ORIGIN MESSAGE
```

Note that you can also use the `dump logs OSGi` command or the Admin UI to see additional log output if Weave was not started as a service/daemon.

## Examples

The following examples help explain each element in each item.

### Example 1

```

2002-01-06 00:06:45.405 DEBUG [pool-1-thread-1] system 00000000-0000-
0000-0000-000000000000 com.cohga.client.weave.Activator "Removed
script source from ..."

```

Given the above example, the following items can be determined:

Item	Value	Description
TIME	2002-01-06 00:06:45.405	States the time the message was logged to the file. Format is YEAR-MONTH-DAY HOURS:MINUTES:SECONDS.MILLISECONDS
LOGGING_LEVEL	DEBUG	The Level of the message logged. Possible values are TRACE, DEBUG, WARN, INFO or ERROR
THREAD	[pool1-thread-1]	The thread that the message was logged in (See below for more information about reading logs).
USER_ID	system	The username of the user who made the request that generated the log message, or "system" if it wasn't generated by a user
SESSION_ID	00000000-0000-0000-0000-000000000000	This is a GUID that remains constant across a single Weave session for each client, or all zeros if the request log wasn't generated on behalf of a user. This can be used to group all log messages related to a single user session.
MESSAGE_ORIGIN	com.cohga.client.weave.Activator	The class that the message was logged against. e.g. The class Activator in the package com.cohga.client.weave
MESSAGE	"Removed script source from ..."	The message that is being set to the log. Typically this is in human readable form, however sometimes the messages may not be that intuitive to the user.



## Example 2

```
00:06:45,405 DEBUG [OSGi Console] com.cohga.search.indexer.internal.
Activator "Stopping Weave Index ..."
```

Item	Value	Description
TIME	00:06:45,405	States the time the message was logged to the file. Format is HOURS:MINUTES:SECONDS,MILLISECONDS
LOGGING_LEVEL	DEBUG	The Level of the message logged. Possible values are DEBUG, WARN, INFO or ERROR
THREAD	[OSGi Console]	The thread that the message was logged in (See below for more information about reading logs).
MESSAGE_ORIGIN	com.cohga.search.indexer.internal.Activator	The class that the message was logged against. e.g. The class Activator in the package com.cohga.search.indexer.internal
MESSAGE	"Stopping Weave Index ..."	The message that is being set to the log. Typically this is in human readable form, however sometimes the messages may not be that intuitive to the user.

The main items that change, and that should be noted are the Thread, Message Origin and the Message. These three items define when and where the message was logged from. The Thread component can cause some confusion to many first time readers if they have not familiar with multithreaded applications. A key example is that the same message may be logged three times in succession. This happens due to the WebService, that Weave runs under, which is handling requests from multiple different locations at the same time. Each thread is being executed concurrently and working in isolation within the Weave application. An example of what you would see in the log file for this is:

```
10:09:54,304 DEBUG [btpool0-0] com.cohga.server.processor.script.
ScriptProcessor "Finished script execution (in 91ms)"
10:09:54,305 DEBUG [btpool0-0] com.cohga.server.processor.script.
ScriptProcessor "Starting sending script result"
10:09:54,305 DEBUG [btpool0-0] com.cohga.server.processor.script.
ScriptProcessor "Finished sending script result (in 0ms)"
10:09:54,305 DEBUG [btpool0-0] com.cohga.server.processor.script.
ScriptRequestProcessor "Finished script request processing (in 178ms)"
10:09:54,306 DEBUG [btpool0-4] com.cohga.server.processor.script.
Activator "JSON canonical: {"map":{"extent":{"crs":"EPSG:3111","...}}"
10:09:54,315 DEBUG [btpool0-7] com.cohga.server.map.wms.WmsMapEngine
"Wrote image/png image in 30ms"
10:09:54,315 DEBUG [btpool0-7] com.cohga.server.processor.script.
ScriptProcessor "Finished script execution (in 34ms)"
10:09:54,315 DEBUG [btpool0-7] com.cohga.server.processor.script.
ScriptProcessor "Starting sending script result"
10:09:54,316 DEBUG [btpool0-7] com.cohga.server.processor.script.
ScriptProcessor "Finished sending script result (in 1ms)"
10:09:54,316 DEBUG [btpool0-7] com.cohga.server.processor.script.
ScriptRequestProcessor "Finished script request processing (in 183ms)"
10:09:54,316 DEBUG [btpool0-4] com.cohga.server.processor.script.
ScriptProcessor "Starting script execution"
10:09:54,321 DEBUG [btpool0-4] com.cohga.server.map.base.
BaseMapEngine "Checking scale ranges against 20133.7824"
10:09:54,323 DEBUG [btpool0-4] com.cohga.server.map.wms.WmsMapEngine
"Wrote image/png image in 2ms"
10:09:54,324 DEBUG [btpool0-4] com.cohga.server.processor.script.
```

```

ScriptProcessor "Finished script execution (in 8ms)"
10:09:54,324 DEBUG [btpool0-4] com.cohga.server.processor.script.
ScriptProcessor "Starting sending script result"
10:09:54,324 DEBUG [btpool0-4] com.cohga.server.processor.script.
ScriptProcessor "Finished sending script result (in 0ms)"
10:09:54,324 DEBUG [btpool0-4] com.cohga.server.processor.script.
ScriptRequestProcessor "Finished script request processing (in 179ms)"
10:09:54,325 DEBUG [btpool0-9] com.cohga.server.map.base.
BaseMapEngine "Checking scale ranges against 20133.7824"
10:09:54,332 DEBUG [btpool0-9] com.cohga.server.map.wms.WmsMapEngine
"Wrote image/png image in 6ms"
10:09:54,332 DEBUG [btpool0-9] com.cohga.server.processor.script.
ScriptProcessor "Finished script execution (in 39ms)"
10:09:54,332 DEBUG [btpool0-9] com.cohga.server.processor.script.
ScriptProcessor "Starting sending script result"
10:09:54,333 DEBUG [btpool0-9] com.cohga.server.processor.script.
ScriptProcessor "Finished sending script result (in 1ms)"
10:09:54,333 DEBUG [btpool0-9] com.cohga.server.processor.script.
ScriptRequestProcessor "Finished script request processing (in 241ms)"

```

**i** The above example illustrates that there were four threads being executed. btpool0-0, btpool0-4, btpool0-7 and btpool0-9. It shows that btpool0-4 has added a logging message between a message logged by btpool0-0 and btpool0-7. When looking for a particular sequence messages that have been logged by a particular user, make sure that the thread name is the same throughout the request.

### Common Issues

Some of the underlying libraries used within Weave also produce logging information. Each library determines what it sees as being relevant to log at various debugging levels (i.e. ERROR, INFO, etc). These messages can sometimes confuse the reader about where and when an error has occurred. Also, an error message may be in server parts, this is called a nested exception. Typically the exact error location and message will be the first in

e.g. The error below was caused by the Database closing the connection. The first exception is the most detailed as it explains that the error's original location was in the class `oracle.jdbc.driver.SQLStateMapping` which is core to the Oracle communications libraries back to the database. The next error nested exception is the code that called the Oracle class `com.cohga.server.stats.export.internal.ExportEvents`. This class re-threw the exception which was logged in the logging file.

**w** When reading a log file you will typically work your way from the bottom to the top. If you do and you encounter an error, please go to the first location in the log file that it occurs. It will generally be the most detailed.

```

03:40:32,622 ERROR [PushToDatasource-0] com.cohga.server.stats.
export.internal.ExportEvents "There was an exception during the
rollback."
java.sql.SQLException: Closed Connection
    at oracle.jdbc.driver.SQLStateMapping.newSQLException
(SQLStateMapping.java:70)
    at oracle.jdbc.driver.DatabaseError.newSQLException
(DatabaseError.java:110)
    at oracle.jdbc.driver.DatabaseError.throwSQLException
(DatabaseError.java:171)
    at oracle.jdbc.driver.DatabaseError.throwSQLException
(DatabaseError.java:227)
    at oracle.jdbc.driver.DatabaseError.throwSQLException
(DatabaseError.java:439)
    at oracle.jdbc.driver.PhysicalConnection.rollback
(PhysicalConnection.java:3369)

```

```

    at com.cohga.server.datasource.jdbc.internal.
PooledJdbcDataSource$PooledConnection.rollback(PooledJdbcDataSource.
java:468)
    at com.cohga.server.stats.export.internal.ExportEvents.
doAnExport(ExportEvents.java:118)
    at com.cohga.server.stats.export.internal.ExportEvents$1.run
(ExportEvents.java:46)
    at java.util.concurrent.Executors$RunnableAdapter.call
(Executors.java:441)
    at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.
java:303)
    at java.util.concurrent.FutureTask.run(FutureTask.java:138)
    at java.util.concurrent.ThreadPoolExecutor$Worker.runTask
(ThreadPoolExecutor.java:886)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run
(ThreadPoolExecutor.java:908)
    at java.lang.Thread.run(Thread.java:619)
03:40:32,622 WARN [PushToDatasource-0] com.cohga.server.stats.
export.internal.ExportEvents "Failed to push events to Datasource
sqlsvr1"
java.lang.RuntimeException: java.sql.SQLException: Closed Connection
    at com.cohga.server.stats.export.internal.ExportEvents.
doAnExport(ExportEvents.java:125)
    at com.cohga.server.stats.export.internal.ExportEvents$1.run
(ExportEvents.java:46)
    at java.util.concurrent.Executors$RunnableAdapter.call
(Executors.java:441)
    at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.
java:303)
    at java.util.concurrent.FutureTask.run(FutureTask.java:138)
    at java.util.concurrent.ThreadPoolExecutor$Worker.runTask
(ThreadPoolExecutor.java:886)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run
(ThreadPoolExecutor.java:908)
    at java.lang.Thread.run(Thread.java:619)
Caused by: java.sql.SQLException: Closed Connection
    at oracle.jdbc.driver.SQLStateMapping.newSQLException
(SQLStateMapping.java:70)
    at oracle.jdbc.driver.DatabaseError.newSQLException
(DatabaseError.java:110)
    at oracle.jdbc.driver.DatabaseError.throwSQLException
(DatabaseError.java:171)
    at oracle.jdbc.driver.DatabaseError.throwSQLException
(DatabaseError.java:227)
    at oracle.jdbc.driver.DatabaseError.throwSQLException
(DatabaseError.java:439)
    at oracle.jdbc.driver.PhysicalConnection.prepareStatement
(PhysicalConnection.java:3046)
    at oracle.jdbc.driver.PhysicalConnection.prepareStatement
(PhysicalConnection.java:2961)
    at com.cohga.server.datasource.jdbc.internal.
PooledJdbcDataSource$PooledConnection.prepareStatement
(PooledJdbcDataSource.java:458)
    at com.cohga.server.stats.export.internal.ExportEvents.

```

```
doAnExport (ExportEvents.java:101)
... 7 more
```

## Logging for Legacy Weave Versions

### **i** Applicable for Weave versions prior to, and including, 2.6.6

Logging in Weave is done using the third party library named Log4j. It is beyond the scope of this document to explain the how the logging infrastructure works with Log4J however the reader is encouraged to visit the Log4J website to find out more information about inner workings of the library.

The following links will help the reader understand more about Log4J.

Main Site:

<http://logging.apache.org/log4j/1.2/>

Documentation:

<http://logging.apache.org/log4j/docs/documentation.html>

Wiki:

<http://wiki.apache.org/logging-log4j/>

Books:

<http://www.qos.ch/shop/products/log4jManual>

<http://www.amazon.com/Apache-Log4j-Second-Samudra-Gupta/dp/1590594991>

#### Architecture

From the architectural point of view, software application modules and logging components reside in two separate layers. The application makes a call to the logging components in the logging layer and delegates the logging responsibility to those components. The logging components receive the logging request and publish the logging information at preferred destinations.

The figure below represents the collaboration of a software module and its logging components.

The architecture of the log4j API is layered. Each layer consists of different objects performing different tasks. The top layer captures the logging information, the middle layer is involved in analyzing and authorizing the logging information, and the bottom layer is responsible for formatting and publishing the logging information to a destination. In essence, log4j consists of three types of primary objects:

- **Logger:** The Logger object (known as the Category object in releases prior to log4j 1.2) is responsible for capturing logging information. Logger objects are stored in a namespace hierarchy. I'll discuss this topic further in the "Logger Object" section of this chapter.
- **Appender:** The Appender object is responsible for publishing logging information to various preferred destinations. Each Appender object will have at least one target destination attached to it. For example, a ConsoleAppender object is capable of printing logging information to a console.
- **Layout:** The Layout object is used to format logging information in different styles. Appender objects utilize Layout objects before publishing logging information. Layout objects play an important role in publishing logging information in a way that is human-readable and reusable.

The preceding core objects are central to the architecture of log4j. Apart from them, there are several auxiliary objects that can plug and play to any layer of the API. These objects help manage the different Logger objects active within an application and fine-tune the logging process.

The principal auxiliary components in the log4j framework that play a vital role in the logging framework are:

- **Level:** The Level object, previously referred to as the Priority object, defines the granularity and priority of any logging information. Each piece of logging information is accompanied by its appropriate Level object, which tells the Logger object about the priority of the information. There are seven levels of logging defined within the API: OFF, DEBUG, INFO, ERROR, WARN, FATAL, and ALL. Each level has a unique integer value. The Level values can be arranged in an ascending manner:

ALL<DEBUG<INFO<WARN<ERROR<FATAL<OFF

where ALL means most of the logging information will be published and OFF means none of the logging information will be published. For more on this topic, see the "Level Object"

- **Filter:** The Filter object is used to analyze logging information and make further decisions on whether that information should be logged or not. In the log4j context, Appender objects can have several Filter objects associated with them. If logging information is passed to a particular Appender object, all the Filter objects associated with that Appender need to approve the logging information before it can be published to the preferred destination attached to the Appender. Filter objects are very helpful in filtering out unwanted logging information based on any application-specific criteria.
- **ObjectRenderer:** The ObjectRenderer object is specialized in providing a String representation of different objects passed to the logging framework. More precisely, when the application passes a custom Object to the logging framework, the logging framework

will use the corresponding `ObjectRenderer` to obtain a `String` representation of the passed `Object`. This is used by `Layout` objects to prepare the final logging information.

- `LogManager`: The `LogManager` object manages the logging framework. It is responsible for reading the initial configuration parameters from a system-wide configuration file or a configuration class. The `LogManager` stores in a namespace hierarchy each `Logger` instance created with a namespace within an application. When we try to obtain the reference named logger, the `LogManager` class returns the already created instance of it, or creates a new instance of the named logger, stores it in a repository for future reference, and returns the new instance to the caller application.

#### Configuring logging

```
#set the level of the root logger to DEBUG and set its appender
#as an appender named testAppender
log4j.rootLogger = DEBUG, testAppender
#define a named logger
log4j.logger.dataAccessLogger = com.apress.logging.logger
#set the appender named testAppender to be a console appender
log4j.appender.testAppender=org.apache.log4j.ConsoleAppender
#set the layout for the appender testAppender
log4j.appender.testAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.testAppender.layout.conversionPattern=%m%n
```

The preceding configuration defines the level of the root logger as `DEBUG` and specifies the appender to use as `testAppender`. Next, we define the appender `testAppender` as referencing the `org.apache.log4j.ConsoleAppender` object and specify the layout of the appender as `org.apache.log4j.PatternLayout`. A `PatternLayout` also requires that a conversion pattern or a format be supplied. We supply the conversion pattern `%m%n` in this case, which means the logging message will be printed followed by a newline character.

A more complex configuration can attach multiple appenders to a particular logger. Each appender, in turn, can have a different layout, and that layout can have a conversion pattern associated with it. Below is an example of a more complex configuration file.

```
# define the root logger with two appenders writing to console and
file
log4j.rootLogger = DEBUG, CONSOLE, FILE

#define your own logger named com.foo
#and assign level and appender to your own logger
log4j.logger.com.foo=DEBUG,FILE

#define the appender named FILE
log4j.appender.FILE=org.apache.log4j.FileAppender
log4j.appender.FILE.File=${user.home}/log.out

#define the appender named CONSOLE
log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
log4j.appender.CONSOLE.conversionPattern=%m%n
```

This configuration file defines the root logger as having level `DEBUG` and attaches two appenders named `CONSOLE` and `FILE` to it. We define one of our own custom loggers with the name `com.foo` and the level `DEBUG` and attach an appender named `FILE` to the custom logger. The appender `FILE` is defined as `org.apache.log4j.FileAppender`. The `FILE` appender writes to a file named `log.out` located in the `user.home` system path. It is important to note that `log4j` supports UNIX-style variable substitution such as `${variableName}`. The variable name defined is considered as the key and searched first in the system properties. If the `log4j` framework does not find the name, it then looks for the value for the variable in the properties file being parsed. The `CONSOLE` appender is then assigned to the `org.apache.log4j.ConsoleAppender` and the conversion pattern defined is `%m%n`, which means the printed logging message will be followed by a newline character.

Given the above simple examples the default logging setup for Weave should now look fairly self explanatory.

```

# Logging settings, see Log4J documentation for further help
# Web address http://jakarta.apache.org/log4j/docs/api/index.html

# Threshold can be set to DEBUG, INFO, WARN, ERROR or FATAL

# Output to stdout and log file, use during instial installation
# The following should be used when debugging Weave
log4j.rootCategory=DEBUG, stdout, file

# Uncomment the following to reduce the amount of information
written to the logfile
# Note: we have removed the stdout to not have any messages being
written to the standard out
#log4j.rootCategory=ERROR, file

# Define a logger that sends it's outout to the console
# only usfull when Weave isn't installed as a service
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Threshold=DEBUG
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%m <%c> [%t] %n

# Define a logger that sends it's output to a rolling log file
log4j.appender.file=org.apache.log4j.DailyRollingFileAppender
log4j.appender.file.Threshold=DEBUG
log4j.appender.file.File=../logs/weave.log
log4j.appender.file.Encoding=UTF-8
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%d{ABSOLUTE} %-5p [%t]
%x %c "%m"%n

# If you want email for problems then add this to the rooCategory
# e.g. alter the above to log4j.rootCategory=ERROR, email, file
log4j.appender.email=org.apache.log4j.net.SMTPAppender
log4j.appender.email.Threshold=ERROR
log4j.appender.email.SMTPHost=mail
log4j.appender.email.SMTPPort=22
# Uncomment the following lines if you have to authenticate when
sending mail
#log4j.appender.email.SMTPUsername=mail
#log4j.appender.email.SMTPPassword=mail

# Set the following to true to enable debugging information about
sending email to the stdout.
# Useful if you find you are having issues send mail.
log4j.appender.email.SMTPDebug=false

log4j.appender.email.From=weave@localdomain
log4j.appender.email.To=admin@localdomain

```

```
log4j.appender.email.Subject=[Weave] Application error
log4j.appender.email.BufferSize=512
log4j.appender.email.layout=org.apache.log4j.PatternLayout
log4j.appender.email.layout.ConversionPattern=%d{ABSOLUTE} %-5p [%
t] %x %c "%m"%n

# Reduce the messages from some of the apache classes
# Uncomment the following if you would like to see more detailed
messages from some
# of the third party libraries used. Please note some of these
packages produce vast
# amounts of logging information and will most likely reduce the
speed of Weave.
log4j.logger.org.apache=WARN
log4j.logger.org.mortbay=ERROR
log4j.logger.org.geotools=ERROR
log4j.logger.net.refrations=ERROR
log4j.logger.org.springframework=WARN
log4j.logger.org.acegisecurity=WARN
log4j.logger.net.sf.ehcache=ERROR

# Comment out the following line to get AXL requests and responses
logged
log4j.logger.com.cohga.server.map.arcims.log.ArcImsLog=ERROR
```

Reading Log files

When Weave has been installed without any modifications to the logging.properties file, the Weave logs the following information to both the console and the weave.log file within the \$WEAVE\_HOME/logs directory. The weave.log file contains the current log messages that Weave has produced in the following format

TIME LOGGING\_LEVEL THREAD MESSAGE\_ORIGIN MESSAGE

Examples

The following examples help explain each element in the each item.

Example 1

```
00:06:45,405 DEBUG [pool-1-thread-1] com.cohga.client.weave.
Activator "Removed script source from ..."
```

Give the above example the following items can be determined.

Item	Value	Description
TIME	00:06:45,405	States the time the message was logged to the file. Format is HOURS:MINUTES:SECONDS,MILLISECONDS
LOGGING_LEVEL	DEBUG	The Level of the message logged. Possible values are DEBUG, WARN, INFO or ERROR
THREAD	[pool1-thread-1]	The thread that the message was logged in (See below for more information about reading logs).

MESSAGE_ORIGIN	com.cohga.client.weave.Activator	The class that the message was logged against. e.g. The class Activator in the package com.cohga.client.weave
MESSAGE	"Removed script source from ..."	The message that is being set to the log. Typically this is in human readable form, however sometimes the messages may not be that intuitive to the user.

*Example 2*

```
00:06:45,405 DEBUG [OSGi Console] com.cohga.search.indexer.internal.Activator "Stopping Weave Index ..."
```

Item	Value	Description
TIME	00:06:45,405	States the time the message was logged to the file. Format is HOURS:MINUTES:SECONDS,MILLISECONDS
LOGGING_LEVEL	DEBUG	The Level of the message logged. Possible values are DEBUG, WARN, INFO or ERROR
THREAD	[OSGi Console]	The thread that the message was logged in (See below for more information about reading logs).
MESSAGE_ORIGIN	com.cohga.search.indexer.internal.Activator	The class that the message was logged against. e.g. The class Activator in the package com.cohga.search.indexer.internal
MESSAGE	"Stopping Weave Index ..."	The message that is being set to the log. Typically this is in human readable form, however sometimes the messages may not be that intuitive to the user.

The main items that change and you should be aware of are the Thread, Message Origin and the Message. These three items define when and where the message was logged from. The Thread component can cause some confusion to many first time readers if they have not familiar with multithreaded applications. A key example is that you may see the same message three times in succession. This happens due the the WebService that Weave runs under, is handling requests from multiple different locations at the same time. Each thread is being executed concurrently and working in isolation within the Weave application. An example of what you would see in the log file is

```
10:09:54,304 DEBUG [btpool0-0] com.cohga.server.processor.script.ScriptProcessor "Finished script execution (in 91ms)"
10:09:54,305 DEBUG [btpool0-0] com.cohga.server.processor.script.ScriptProcessor "Starting sending script result"
10:09:54,305 DEBUG [btpool0-0] com.cohga.server.processor.script.ScriptProcessor "Finished sending script result (in 0ms)"
10:09:54,305 DEBUG [btpool0-0] com.cohga.server.processor.script.ScriptRequestProcessor "Finished script request processing (in 178ms)"
10:09:54,306 DEBUG [btpool0-4] com.cohga.server.processor.script.Activator "JSON canonical: {"map":{"extent":{"crs":"EPSG:3111","...}}"
```



```

ScriptProcessor "Starting sending script result"
10:09:54,316 DEBUG [btpool0-7] com.cohga.server.processor.script.
ScriptProcessor "Finished sending script result (in 1ms)"
10:09:54,316 DEBUG [btpool0-7] com.cohga.server.processor.script.
ScriptRequestProcessor "Finished script request processing (in
183ms)"
10:09:54,316 DEBUG [btpool0-4] com.cohga.server.processor.script.
ScriptProcessor "Starting script execution"
10:09:54,321 DEBUG [btpool0-4] com.cohga.server.map.base.
BaseMapEngine "Checking scale ranges against 20133.7824"
10:09:54,323 DEBUG [btpool0-4] com.cohga.server.map.wms.
WmsMapEngine "Wrote image/png image in 2ms"
10:09:54,324 DEBUG [btpool0-4] com.cohga.server.processor.script.
ScriptProcessor "Finished script execution (in 8ms)"
10:09:54,324 DEBUG [btpool0-4] com.cohga.server.processor.script.
ScriptProcessor "Starting sending script result"
10:09:54,324 DEBUG [btpool0-4] com.cohga.server.processor.script.
ScriptProcessor "Finished sending script result (in 0ms)"
10:09:54,324 DEBUG [btpool0-4] com.cohga.server.processor.script.
ScriptRequestProcessor "Finished script request processing (in
179ms)"
10:09:54,325 DEBUG [btpool0-9] com.cohga.server.map.base.
BaseMapEngine "Checking scale ranges against 20133.7824"
10:09:54,332 DEBUG [btpool0-9] com.cohga.server.map.wms.
WmsMapEngine "Wrote image/png image in 6ms"
10:09:54,332 DEBUG [btpool0-9] com.cohga.server.processor.script.
ScriptProcessor "Finished script execution (in 39ms)"
10:09:54,332 DEBUG [btpool0-9] com.cohga.server.processor.script.
ScriptProcessor "Starting sending script result"
10:09:54,333 DEBUG [btpool0-9] com.cohga.server.processor.script.
ScriptProcessor "Finished sending script result (in 1ms)"
10:09:54,333 DEBUG [btpool0-9] com.cohga.server.processor.script.
ScriptRequestProcessor "Finished script request processing (in
241ms)"

```

The above example illustrates that there were four threads being executed. btpool0-0, btpool0-4, btpool0-7 and btpool0-9. You can see that btpool0-4 has added a logging message between a message logged by btpool0-0 and btpool0-7. When looking for a particular sequence messages that have been logged by a particular user, make sure that the thread name is the same throughout the request.

## Common issues

Some of the underlying libraries used within Weave also produce logging information. Each library determines what it sees as being relevant to log at various debugging levels (i.e. ERROR, INFO, etc). These messages can sometime confuse the reader about where and when an error has occurred. Also an error message may be in sever parts, this is called a nested exception. Typically the exact error location and message will be the first in

e.g. The error below was caused by the Database closing the connection. The first exception is the most detailed as it explains that the error's original location was in the class oracle.jdbc.driver.SQLStateMapping which is core to the oracle communications libraries back to the database. The next error nested exception is the code that called the oracle class com.cohga.server.stats.export.internal.ExportEvents. This classes re threw the exception which was logged in the logging file.

When reading a log file you will typically work your way from the bottom to the top. If you do and you encounter an error, please go to the first location in the log file that it occurs. It will generally be the most detailed.

```

03:40:32,622 ERROR [PushToDatasource-0] com.cohga.server.stats.
export.internal.ExportEvents "There was an exception during the

```

```

rollback.
Connection problems: {}"
java.sql.SQLException: Closed Connection
    at oracle.jdbc.driver.SQLStateMapping.newSQLException
(SQLStateMapping.java:70)
    at oracle.jdbc.driver.DatabaseError.newSQLException
(DatabaseError.java:110)
    at oracle.jdbc.driver.DatabaseError.throwSQLException
(DatabaseError.java:171)
    at oracle.jdbc.driver.DatabaseError.throwSQLException
(DatabaseError.java:227)
    at oracle.jdbc.driver.DatabaseError.throwSQLException
(DatabaseError.java:439)
    at oracle.jdbc.driver.PhysicalConnection.rollback
(PhysicalConnection.java:3369)
    at com.cohga.server.datasource.jdbc.internal.
PooledJdbcDataSource$PooledConnection.rollback(PooledJdbcDataSource.
java:468)
    at com.cohga.server.stats.export.internal.ExportEvents.
doAnExport(ExportEvents.java:118)
    at com.cohga.server.stats.export.internal.ExportEvents$1.run
(ExportEvents.java:46)
    at java.util.concurrent.Executors$RunnableAdapter.call
(Executors.java:441)
    at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.
java:303)
    at java.util.concurrent.FutureTask.run(FutureTask.java:138)
    at java.util.concurrent.ThreadPoolExecutor$Worker.runTask
(ThreadPoolExecutor.java:886)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run
(ThreadPoolExecutor.java:908)
    at java.lang.Thread.run(Thread.java:619)
03:40:32,622 WARN [PushToDatasource-0] com.cohga.server.stats.
export.internal.ExportEvents "Failed to push events to Datasource
{}"
java.lang.RuntimeException: java.sql.SQLException: Closed Connection
    at com.cohga.server.stats.export.internal.ExportEvents.
doAnExport(ExportEvents.java:125)
    at com.cohga.server.stats.export.internal.ExportEvents$1.run
(ExportEvents.java:46)
    at java.util.concurrent.Executors$RunnableAdapter.call
(Executors.java:441)
    at java.util.concurrent.FutureTask$Sync.innerRun(FutureTask.
java:303)
    at java.util.concurrent.FutureTask.run(FutureTask.java:138)
    at java.util.concurrent.ThreadPoolExecutor$Worker.runTask
(ThreadPoolExecutor.java:886)
    at java.util.concurrent.ThreadPoolExecutor$Worker.run
(ThreadPoolExecutor.java:908)
    at java.lang.Thread.run(Thread.java:619)
Caused by: java.sql.SQLException: Closed Connection
    at oracle.jdbc.driver.SQLStateMapping.newSQLException
(SQLStateMapping.java:70)
    at oracle.jdbc.driver.DatabaseError.newSQLException

```

```
(DatabaseError.java:110)
    at oracle.jdbc.driver.DatabaseError.throwSQLException
(DatabaseError.java:171)
    at oracle.jdbc.driver.DatabaseError.throwSQLException
(DatabaseError.java:227)
    at oracle.jdbc.driver.DatabaseError.throwSQLException
(DatabaseError.java:439)
    at oracle.jdbc.driver.PhysicalConnection.prepareStatement
(PhysicalConnection.java:3046)
    at oracle.jdbc.driver.PhysicalConnection.prepareStatement
(PhysicalConnection.java:2961)
    at com.cohga.server.datasource.jdbc.internal.
PooledJdbcDataSource$PooledConnection.prepareStatement
(PooledJdbcDataSource.java:458)
    at com.cohga.server.stats.export.internal.ExportEvents.
doAnExport(ExportEvents.java:101)
    ... 7 more
```

## Mapping

When setting up maps in Weave you need to understand that you are configuring 2 completely different things when you're setting up map engines.

### Server Map Engine

Firstly there are server side map engines, these are the things that talk directly to ArcGIS Server, ArcIMS or a WMS server and this is what the Weave server uses to generate map images (both for the browser and for reports, plus anything else that requires a map image, but those things aren't important for this discussion).

These are top level configuration items, and therefore require the namespace in their configuration (e.g. <arcims:mapengine id="...">, <arcgis:mapengine id="..."> or <wms:mapengine id="...">), because it's the namespace that directs the chunk of configuration xml to the correct bundle to process it, and each of these types of map engines is handled by a different bundle.

This way we can add new types of objects just by adding a new bundle and giving it a new namespace (this applies to any top level configuration item, and not just map engines).

These setup the basic settings that Weave requires in order to know how to talk to the respective map service (e.g. userid, url, service names, connection pooling), think of something like a database connection, where different databases require different settings depending on the manufacturer (Oracle, Microsoft), but in the end to Weave it's still just a database.

### Client Map Engine

The second part are the client side map engines which describe to the browser what map layers you want to display in the map panel. These are not top level configuration items, and they only make sense within the context of a map view inside of a client config, so they don't require a namespace (actually they already have a namespace because they're always inside of a client config). Basically they're used to configure an OpenLayers map component.

At their simplest a client side map engine only needs to be provided with the id of a server side map engine <mapengine id="server\_mapengine\_id"/>, in which case the rest of the stuff that's needed for OpenLayers is figured out by Weave based on what was configured in the server side map engine that has the same id.

Additionally you can provide a few options in the client map engine config to alter slightly the layer that OpenLayers creates (things like the transition effect and opacity), these optional settings are applied to the OpenLayers component effect the way the map image is displayed /generated from the client perspective.

### Map Image Generation

If you do nothing more than this then each time the browser wants to update the map display it'll send a request to the Weave server asking to update the image related to a particular map engine (as part of the request the map engine id is sent, along with the map extent, screen size, visible layers, etc), it doesn't care what type of map engine it is on the other end (ArcIMS, AGS or WMS) it just wants a picture back. When the request gets to the server it looks up the corresponding server side map engine and uses the information in the request to generate a new bitmap image using the appropriate API (generating AXL for ArcIMS, constructing a URL for WMS or using the AGS SOAP API for ArcGIS server).

The image is generated and returned to the browser which then updates the display.

All of the above happens because the default type for a layer added to the OpenLayers component in the Weave client is set to "weave". That is, if you don't set the type in a client side map engine explicitly Weave sets it to "weave", which is an OpenLayers layer implementation that we created for OpenLayers which knows how to talk to a Weave server to generate map images.

## Map Layer Types

From that you can gather that OpenLayers supports other layer types, for example "wms" where the browser can talk directly to a WMS server to generate map images, not needing to communicate to the Weave server at all.

There are actually a handful of these other OpenLayers layer types including one for talking to ArcIMS, one for ArcGIS and a handful for directly accessing map tiles among others.

Each one has their own configuration options requires parameters, as defined by OpenLayers.

But there's one small issue with these other OpenLayers layer types, the Weave server doesn't know anything about them. This isn't really a problem until you try and generate a report with a map in it (or do anything else that requires that the server generate a map image that represents what the user is seeing on their screen).

Since the Weave server doesn't know anything about the layers the OpenLayers was accessing directly it can't draw the layers on the map that it's trying to create and the resultant map image won't match what the client was viewing.

## Linking Client Layer to Server Map Engine

Fortunately there's a way around this, allowing you to gain the performance of having the client directly access the map engine whilst still allowing the server to generate a map image that matches the client.

And that's by configuring the client map engine with information about a server side map engine that can be used to replicate the map image on the server.

For example in the case of the client side "wms" layer type you would setup a corresponding server side map engine pointing to the same WMS server using the same base URL that the client was configured with.

Then you'd add a <server> section to the client map engine configuration telling it which server side map engine the server should use if it's ever asked to draw a map that includes that particular OpenLayer layer.

## ArcGIS Server

All of that can get a little messy, or a lot messy when trying to do this with tile caches, so to make life easier for sites wanting to use ArcGIS Server tile caches, which are inherently a client side mapping option, Weave does a bunch of extra stuff to take care of a lot of the manual configuration that would normally be required (believe me you do not want to have to manually setup a client side ArcGIS tile cache).

Basically if Weave sees that you're referencing a server side ArcGIS map engine in your client config, and that map service supports tile caching, then it'll query the ArcGIS server to obtain the information that describes the tiles and use that to alter the client side map engines config, it will also change its type from "weave" to "agstiled" (for a single fused map cache) or "agsrest" (for a multi-layer map cache), and finally add the required server tags so the server knows what server side map engine to talk to if required.

So generally to make use of an ArcGIS tile cache nothing explicitly has to be done, either in the server side map engine or the client side configuration (in fact you have to explicitly include a setting to disable use of the cache if it's available).

## Directly accessing ArcGIS tiles

Both of the ArcGIS layer types referenced above, "agstiled" and "agsrest", still talk to an ArcGIS Server instance, they just do it from the browser rather than the Weave server. So, while they're quicker than going via the Weave server they're still not the quickest option, which is going to the tiles directly (well, via a web server, but that happens with the other too as well). Of course accessing the tiles directly has the down side that the tiles must already exist, where as these 2 option can dynamically generate the tiles when they're asked for.

Accessing the tiles directly it turns out is almost exactly the same as accessing them via the "agstiled" method. The only difference is the base URL that the tiles are accessed from, for the "agstiled" method the base URL points to the ArcGIS server, e.g. <http://hostname:8399/arcgis/services/imagery/> but for directly accessing the tiles it's something like <http://hostname:80/tiles/imagery/>, and the rest of the URL, the part that actually describes the tile to retrieve, is exactly the same (it includes the level, row, column and format).

One other difference is in the level, row and column are formatted when talking to ArcGIS verses accessing the tiles directly, which is why the <useArcGISServer> option must be set to false in the client map engine configuration if you want to alter the map engine to access the tiles directly.

So to directly access pre-generated ArcGIS tiles you just need to setup things so that an "agstiled" client side map engine is used and change its URL parameter. The first part can be done by ensuring that the client map engine points to a server side map engine that is setup to serve a fused map cache (so that Weave will perform the automatic processing required to set all the other parameters that are needed) and the second part can be accomplished by setting the <url> parameter for the client side map engine (thereby overriding the one the URL Weave would generate to access the tiles via the ArcGIS server).

### Other Layer Types

At the moment ArcGIS tile caches are the only client side map engine that Weave performs the extra processing on, but it may be possible to add similar functionality for ArcIMS and WMS (not that it's add support for tile caches to these 2 map engine types) or to the other tile caching formats that Weave/OpenLayers support (e.g. TileCache, TMS and OpenStreetMap) but that's a problem for another day and at the moment you have to manually figure out what values are required for the various layer types.

### WMS Server

This update exposes Weave map engines as a WMS service. The bundle can also be used to create URL's that can be called from external applications to create custom map images. The bundle supports additional parameters beyond those required by WMS and allows for more customised map output.

A new configuration describes a map context that is similar to that which is setup for a map view in a client configuration. It is generated in a similar way to the maps that are included in a BIRT report.

There can be multiple map context setups, that each describe different map setups.

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:wmsserver="urn:com.cohga.server.wms#1.0">

    <wmsserver:context id="test">
        <mapengine id="mapengine.raster">
            <layer>satellite</layer>
        </mapengine>
        <mapengine id="mapengine.vector">
            <layer>property</layer>
            <layer>road</layer>
            <layer>suburb</layer>
            <layer>mccboud</layer>
        </mapengine>
        <mapengine id="mapengine.parks" opacity="0.75">
            <layer>Reserve Images</layer>
        </mapengine>
    </wmsserver:context>

</config>
```

The above example exposes a new WMS service at `/weave/wms/test` that will contain the layers listed (and only those layers). When the WMS client tries to draw the map, Weave will use the three map engines to generate the individual images and combine them into a single map image to return to the client. This service should be supported in any WMS compliant client.

### Complete option list

Name	Description	WMS Compliant
srs	Coordinate reference system of the coordinates specified in the URL	Y
width	Width of generated image in pixels	Y
height	Height of generated image in pixels	Y
format	Output format of the generated image	Y
bgcolor	Background colour	Y
transparent	true if transparent images should be created, false if opaque images should be generated	Y
bbox	Comma separated minx, miny, maxx and maxy of the generated image	Y
layers	List of layers to draw, all layers will be drawn if not set at all, and no layers drawn if layers is set to no value	Y
exceptions	Method used to return errors to the caller	Y
size	Comma separated width and height of generated image in pixels	
dpi	DPI of generated image, to ensure scale dependencies are calculated correctly	
minx	Left X coordinate	
miny	Bottom Y coordinate	

maxx	Right X Coordinate	
maxy	Top Y coordinate	
x	Center X coordinate	
y	Center Y coordinate	
center	Comma separated X and Y coordinate. Both <code>center</code> and <code>centre</code> are supported	
scale	Scale to generate map at	
entity	Entity to use to determine map extent and draw highlight	
ids	Identifiers of entities to zoom to and highlight	
filter	Filters to be applied to 'ids' before calculating the extent. For use if external system uses different id's for the underlying GIS data	
buffer	Amount to scale entity extent by to calculate extent, e.g. 1.2 to place a 20% buffer around the entity bounds	
minscale	Minimum scale to zoom to when zooming to an entity	
highlight	false if the entity highlight should not be drawn, true if it should be drawn	
hlcolor	Vector highlight colour	
mcolor	Marker highlight colour. Should be blue, brown, dark, green, orange, purple, red, silver or yellow. The default is green. The mcolor 'dark' is a dark grey.	



**You must include a parameter or parameters in the URL to set or calculate the map bounding box, this can not be set in the context.**

When specifying the map bounding box setting minx, miny, maxx and maxy is the same as specifying just bbox, and bbox will take precedence if both are present.

If a bounding box is set it will take precedence over setting the map extent based on a center x,y and scale.

If a center x, y and scale are set they will take precedence over the entity and ids.

Setting size will take precedence over setting width and height if both are present.

The above settings will generally be supplied by the WMS client, and defaults can be set in the `context`.

Generally these will all be set by the WMS client when sending WMS requests to the Weave server and do not need to be specified in the configuration, however, you may want to set these values in the configuration if you're intending on constructing and using the URL's from another application and you want to simplify the URL that the other application needs to use.

For example, if you have an application that you want to always generate an 800x800 PNG image with a light blue background then you could use the URL:

```
/weave/wms/test?width=800&height=800&bgcolor=0x7f7fff&format=png&bbox=140,-36,141,-35
```

or you could set width, height, format and bgcolor in the context and use the following URL instead

```
/weave/wms/test?bbox=140,-36,141,-35
```

This allows you to provide a simplified URL to the application provider while specifying the fixed settings yourself, it all depends upon the requirements of the other application and how much of it can be pre-set.

#### Basic Context Example Requests

These example assume the above example map context

#### Generate a map at a given extent

```
/weave/wms/test?
maxx=355971&maxy=5831675&minx=323098&miny=5807358&width=800&height=600
&format=png32
/weave/wms/test?bbox=323098,580735,355971,5831675&size=800,
600&format=png32
```

#### Generate a map at a given location

```

/weave/wms/test?
x=355971&y=5831675&scale=5000&width=800&height=600&format=png32
/weave/wms/test?center=355971,5831675&scale=5000&size=800,
600&format=png32

```

### Generate a map at a given property

```

/weave/wms/test?entity=property&ids=242765&minscale=2500&buffer=1.
2&width=800&height=600&format=png32

```

### Generate a map at a given property and not highlighting the property

```

/weave/wms/test?entity=property&ids=242765&minscale=2500&buffer=1.
2&highlight=false&width=800&height=600&format=png32

```

### Generate a map at a given location and just highlight a property

```

/weave/wms/test?
x=355971&y=5831675&scale=5000&entity=property&ids=242765&minscale=2500
&buffer=1.2&width=800&height=600&format=png32

```

#### Extended Context Example Requests

The following examples assume the following map context and are the equivalent of the above.

```

<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:wmsserver="urn:
com.cohga.server.wms#1.0">
  <wmsserver:context id="test">
    <minscale>2500</minscale>
    <buffer>1.2</buffer>
    <width>800</width>
    <height>600</height>
    <format>png32</format>
    <entity>property</entity>
    <mapengine id="mapengine.raster">
      <layer>satellite</layer>
    </mapengine>
    <mapengine id="mapengine.vector">
      <layer>property</layer>
      <layer>road</layer>
      <layer>suburb</layer>
      <layer>mccbound</layer>
    </mapengine>
    <mapengine id="mapengine.parks" opacity="0.75">

```

```

        <layer>Reserve Images</layer>
    </mapengine>
</wmsserver:context>
</config>

```

### Generate a map at a given extent

```

/weave/wms/test?maxx=355971&maxy=5831675&minx=323098&miny=5807358
/weave/wms/test?bbox=323098,580735,355971,5831675

```

### Generate a map at a given location

```

/weave/wms/test?x=355971&y=5831675&scale=5000
/weave/wms/test?center=355971,5831675&scale=5000
/weave/wms/test?center=337649.188,5818856.678&scale=1000&srs=EPSG:
28355
/weave/wms/test?center=145.1568367,-37.7632155&scale=5000&srs=EPSG:
4283

```

### Generate a map at a given property

```

/weave/wms/test?entity=property&ids=242765

```

### Generate a map at a given road with a min scale of 5000

```

/weave/wms/test?entity=roads&ids=AS14124&minscale=5000


```

### Generate a map at a centred on given road with a min scale of 5000 showing just the roads and property layers

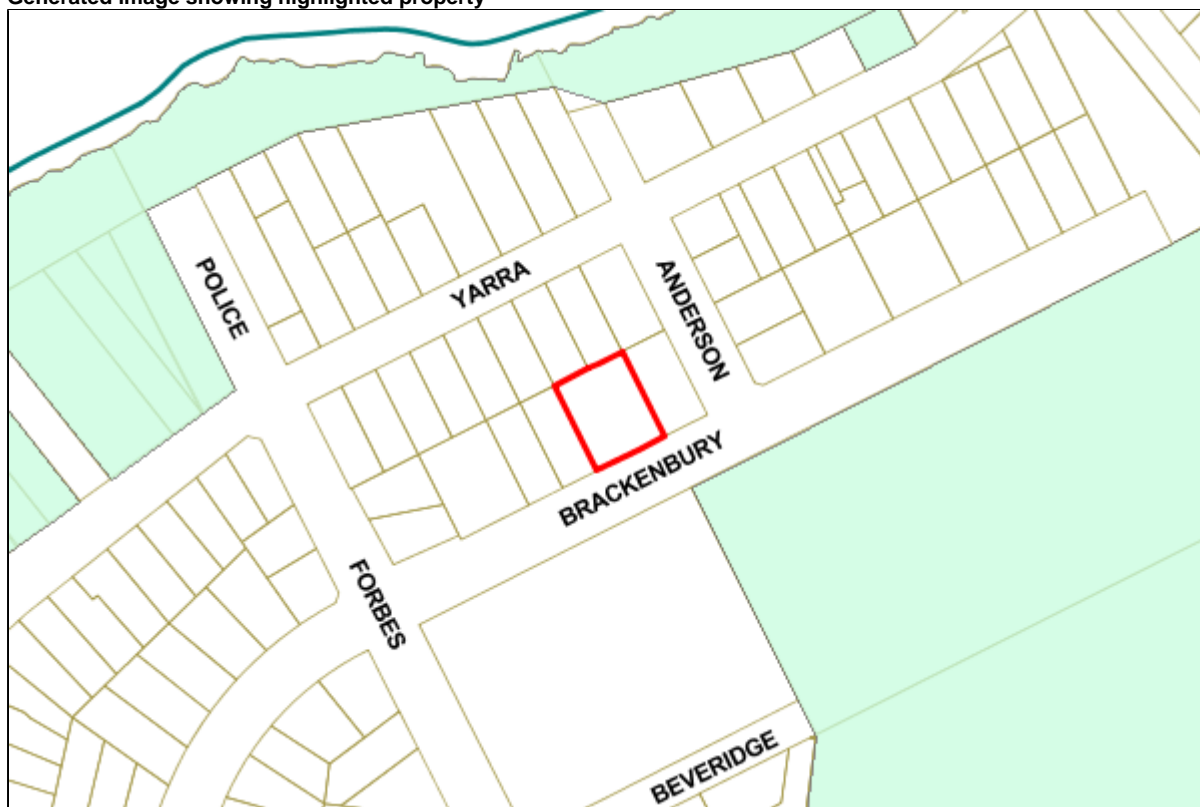
```

/weave/wms/test?entity=roads&ids=AS14124&minscale=5000&layers=road,
property

```

 The last example specifies the layer as `road` and `property` because there's only one `road` and `property` layer in all of the map engines. If the layer id was not unique then the layer can be specified by preceding it with the map engine id and a `|`, e.g. `mapengine.vector|roads`



**Generated image showing highlighted property**

**i** Using multiple contexts allows you to setup different maps depending on the target use.

## Reporting

### BIRT

Weave uses [BIRT](#) as its default reporting engine for producing printed reports. The reports are built using the BIRT Report Designer application that is part of BIRT and then the report designs produced by the report designer are registered with Weave to make them available for generation by users.

### BIRT Report Designer

**⚠** To start the report designer use the `birt.cmd` batch file which is installed in the weave directory, **do not** use the `birt.exe` that is installed under the birt directory.

Once started you use the File|New|New Report... menu to create a new report design.

You should change the default file location (which is under the BIRT directory by default) by un-checking the "Use default" check box and set the Directory to the Weave workspace directory (or a sub directory of the workspace directory).

**i** The Weave BIRT plugin will look in the Weave workspace directory for report designs by default but it's recommended that you place report designs in a sub folder of the workspace directory, in this case the reports directory, to keep the workspace directory clean and organized.

**New Report**  
create a new report

File name:

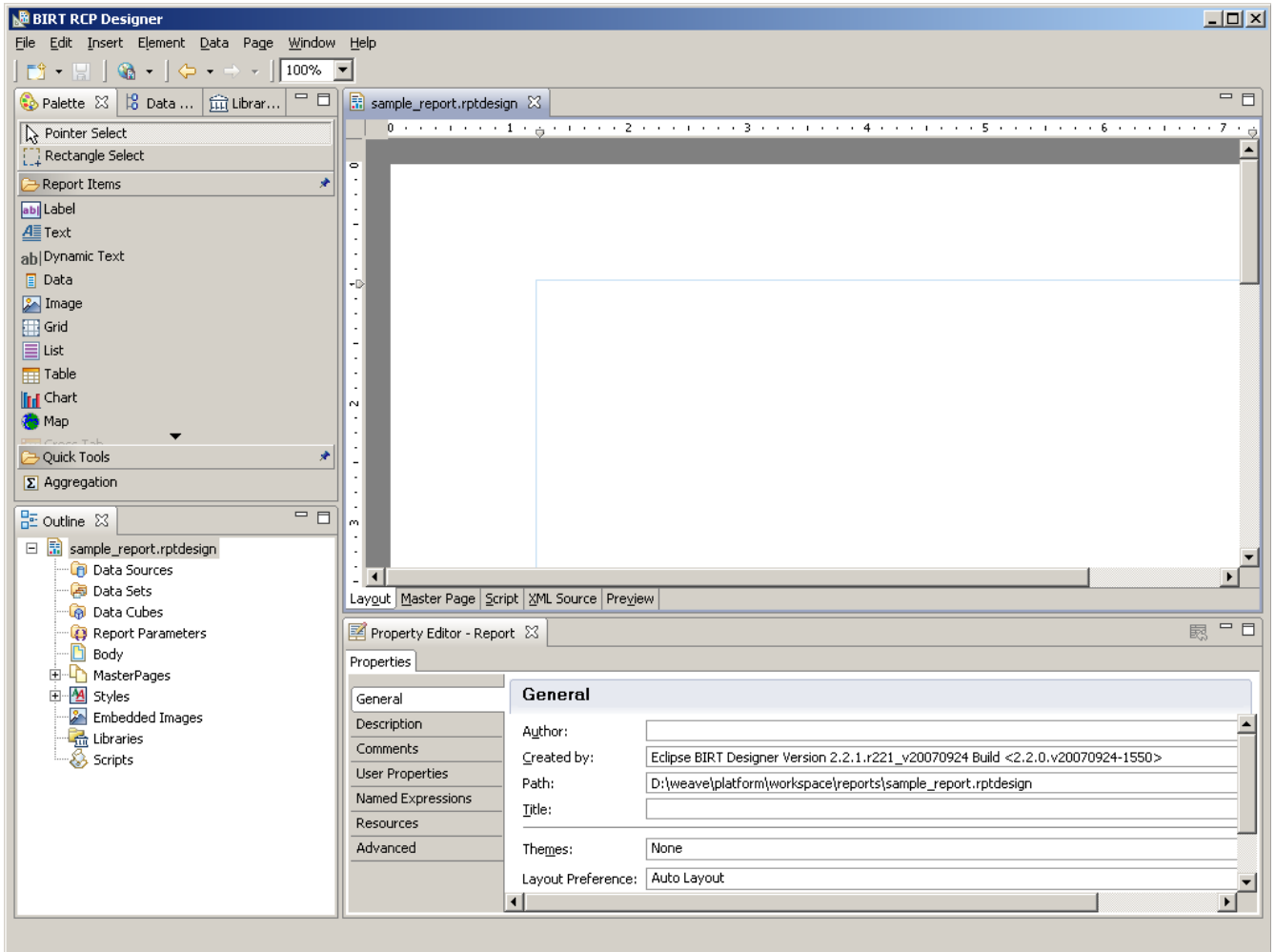
File location

Use default

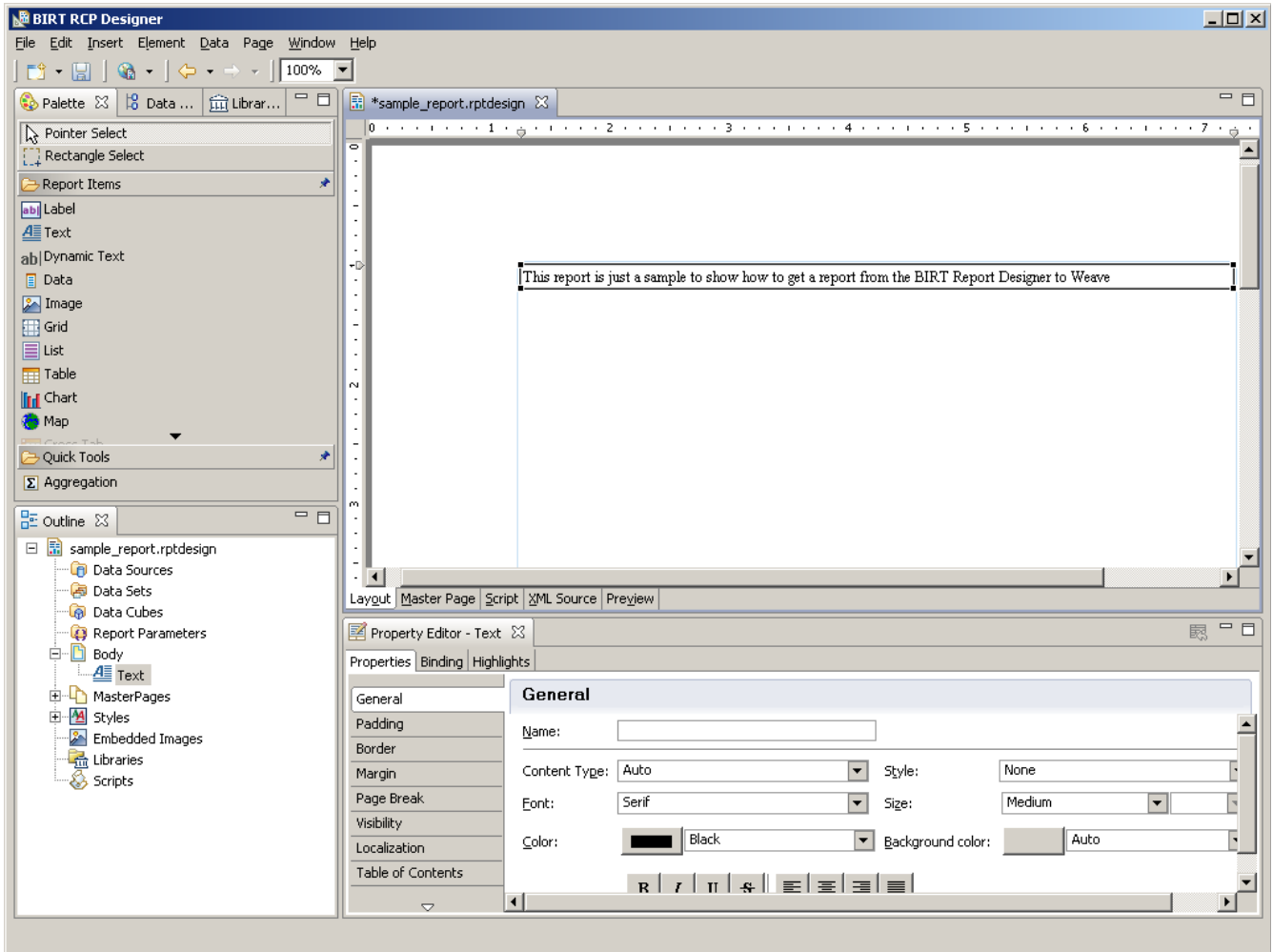
Directory:

If you click the Next button you'll be presented with a screen that allows you to choose a starting template to use for the report. Generally the Blank Report is a suitable starting point.

**i** The Blank Report template does include the current date as part of the Master Page, so it's not entirely blank.



For this sample I'll just add a simple text box to the report since the intent here is to show how to create a BIRT report design, not how to generate a full blown BIRT report.



Once this report is saved the design file will then be available in the ...\\weave\\platform\\workspace\\reports directory and should be ready to register in Weave.

## Report Registration

Once a report design is available it needs to be registered with Weave to make it available for users. The registration involves adding a new entry to the Weave config file for the report design, and there are two methods of performing the registration, either with a separate entry for each report design or as a single entry for a directory of report designs.

The later makes it easier to register a bunch of reports but the former provides more control over individual report registrations.

To create either type of report registration requires the setting of the appropriate namespace for the BIRT configuration items in the config.xml file, the namespace for BIRT reports is com.cohga.server.report.birt, so if we include BIRT reports in a separate config file called reports.xml and include that into our main config.xml file then the initial contents of the report.xml file should be:

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:birt="urn:com.cohga.server.report.birt#1.0">

</config>
```

Then within the config tags we can add a new BIRT report entry for our sample report

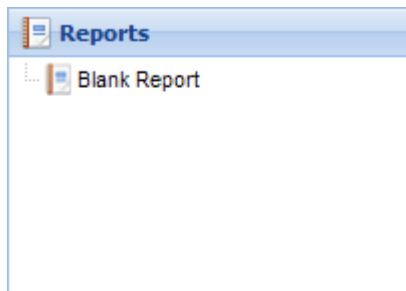
```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<config xmlns="urn:com.cohga.server.config#1.0" xmlns:birt="urn:com.cohga.server.report.birt#1.0">

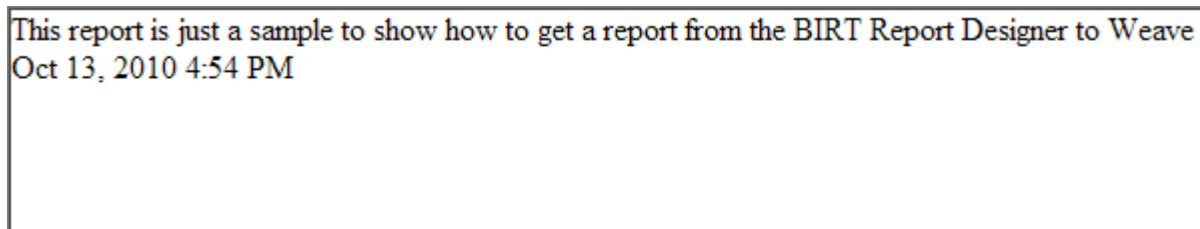
    <birt:report id="sample" report="reports\sample_report"/>

</config>
```

Then when we refresh the client the new report should appear in the Report view



and if we generate the report (in this case as HTML) we should get the following output



And that's the basics of how to get a BIRT report into Weave.

### Customising BIRT Reports

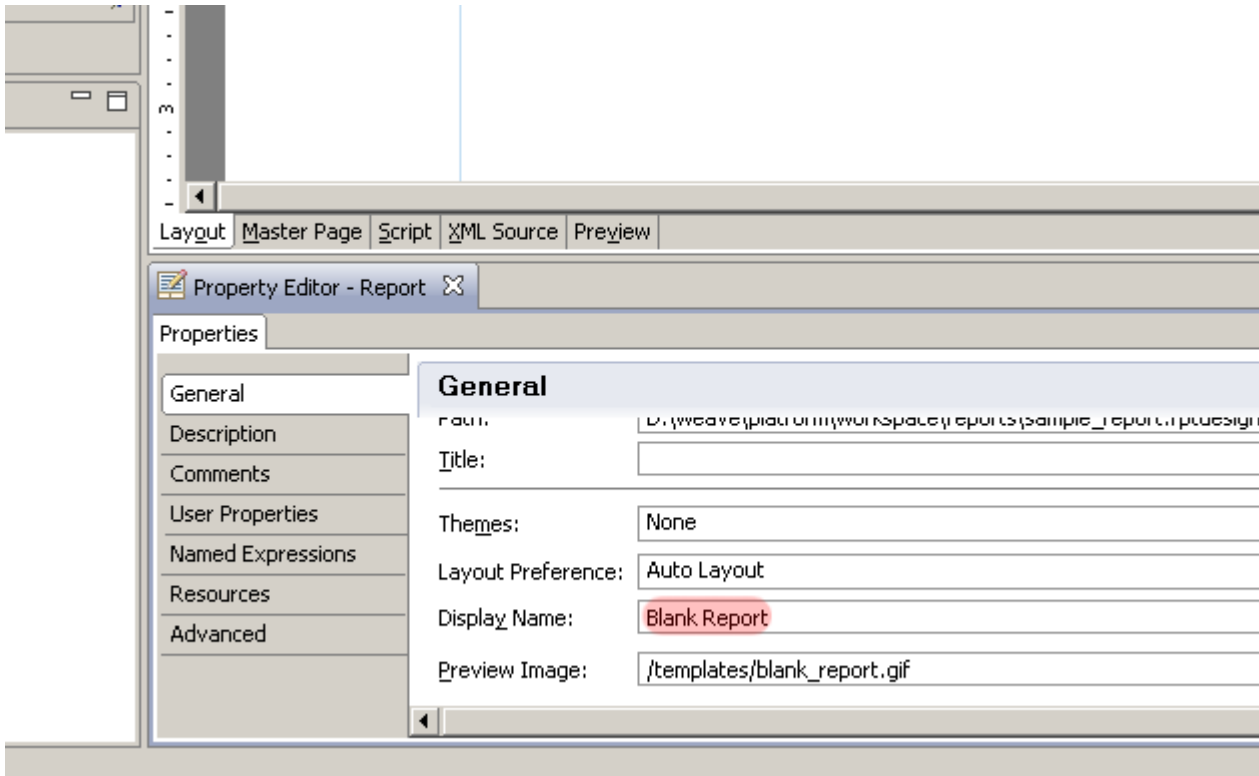
This sections details how to modify BIRT report designs and registrations to alter the report that the user can generate.

## Report Customisation

The first thing we need to look at the the label for the report in the report view.

### Report Label

When displaying an available report to the user Weave will use the Display Name property set in the report design to obtain the text for its label. For the Blank Report template the text for this property is "Blank Report", so that was the text that was displayed in the report panel in the client.



To alter the label text we could go back to the report design, change the Display Name property and save the report design, then the next time the client starts the report will have the new label in the report view.

Alternatively we can set the label directly in the report registration in the config file

```
<?xml version="1.0" encoding="UTF-8"?>

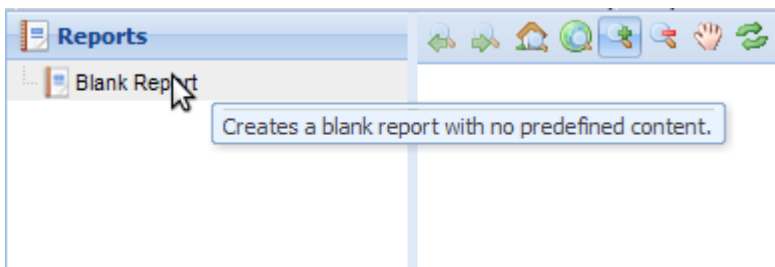
<config xmlns="urn:com.cohga.server.config#1.0" xmlns:birt="urn:com.cohga.server.report.birt#1.0">

    <birt:report id="sample" report="reports\sample_report" label="
Sample Report" />

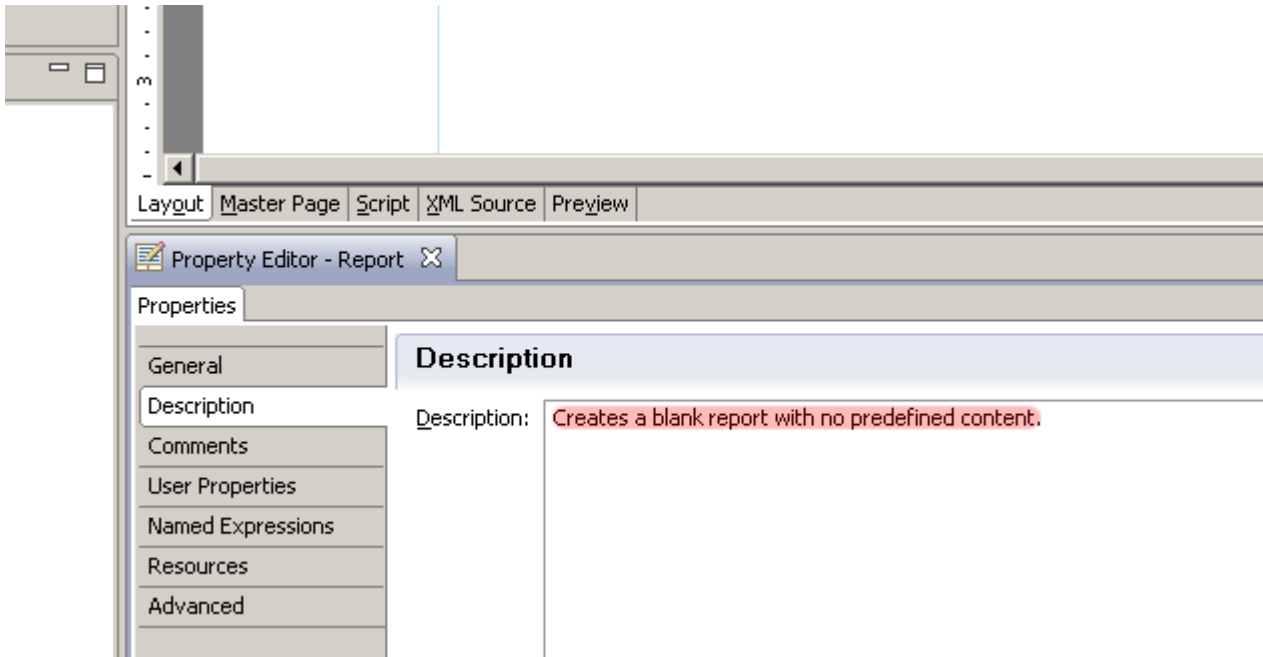
</config>
```

#### Report Description

If you hold the mouse cursor over the report label in the report view you'll notice a description of the report popup



Again, this can be changed by editing the report design and changing the Description in the report properties



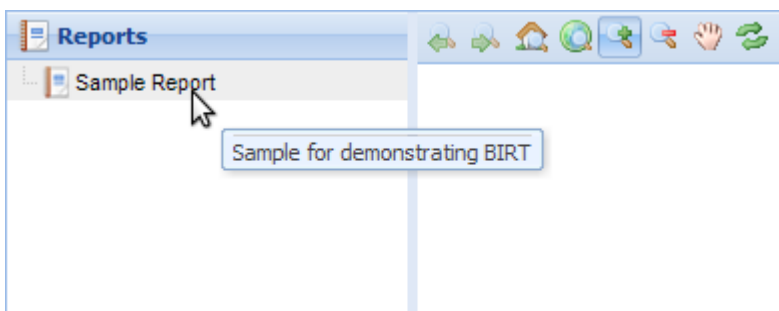
Or alternatively you can set a 'description' property in the report registration.

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:birt="urn:com.cohga.server.report.birt#1.0">

    <birt:report id="sample" report="reports\sample_report" label="Sample Report" description="Sample for demonstrating BIRT"/>

</config>
```



#### Report Parameters

Report parameters allow you to obtain input from the user when it's time to generate the report and that input can be used to alter the generated report.

This can be used simply to include custom text in a report or in more advanced ways by using the scripting environment within the BIRT reporting engine to make major changes to the report.

The report parameters can be seen in the Report Parameters section on the Outline panel in the BIRT report designer.

To create a new report parameter right click on the Report Parameters item and select New Parameter.

From there you can enter the detail for the new report parameter

**New Parameter**

Name:

Prompt text:

Data type:

Display type:

Display As

Help text:

Format as:

Preview with format:

List Limit:  values

Is Required  Do not echo input

Hidden  Allow Duplicate Values

List of value

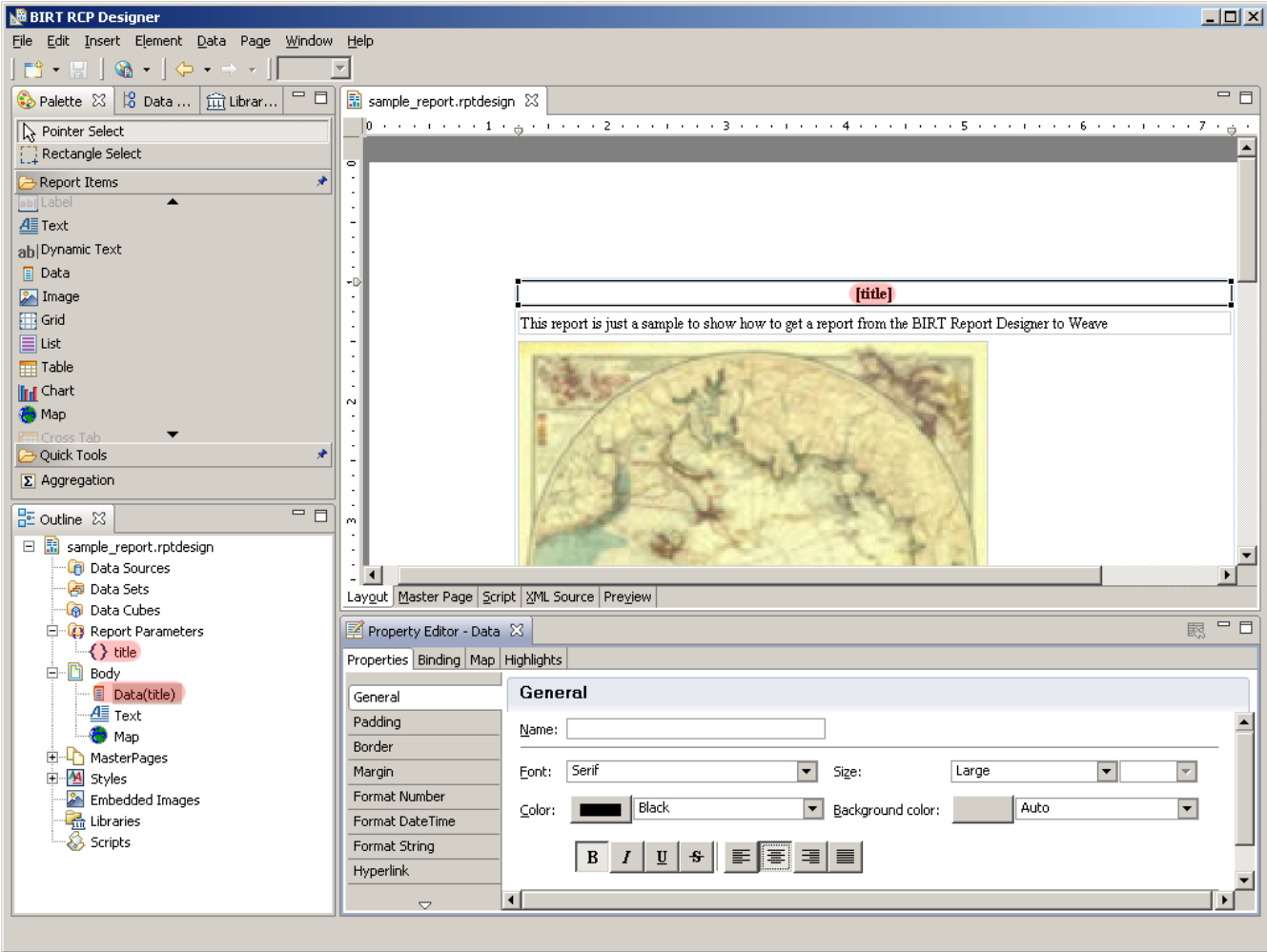
Static  Dynamic

Default value:

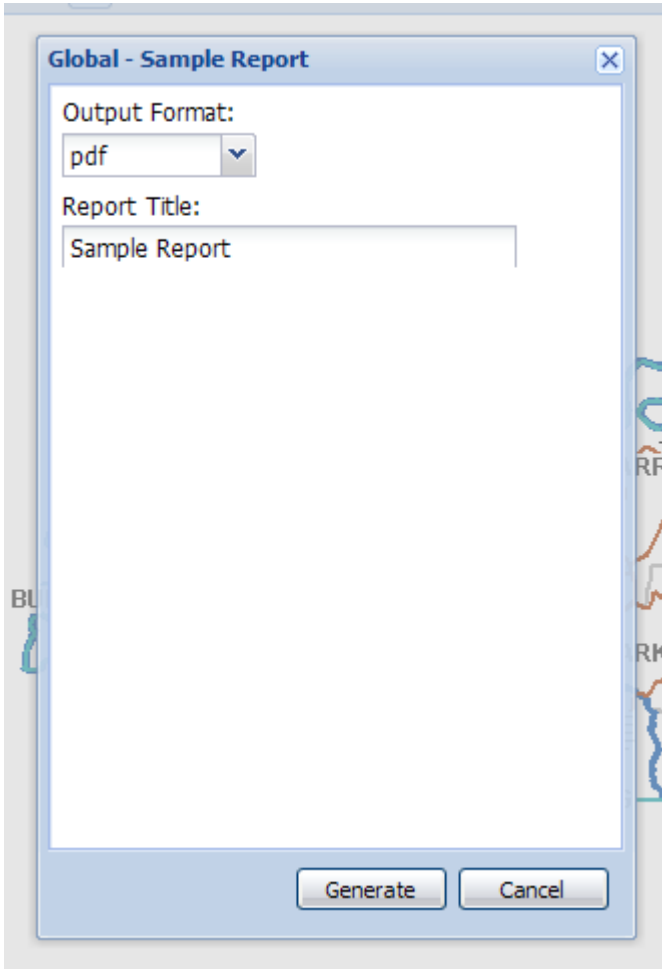
Then you can use the parameter in a report item

- i** A simple way to add a report parameter to a report design and have it display the text is to drag and drop the parameter from the Outline view to the report design

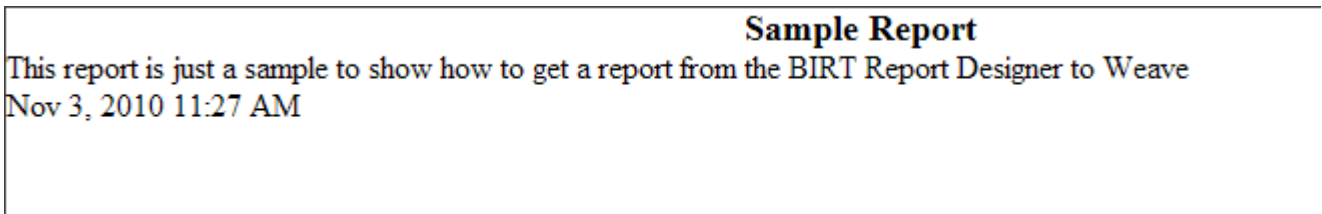





Then when the user tries to generate the report they will be asked for the report title



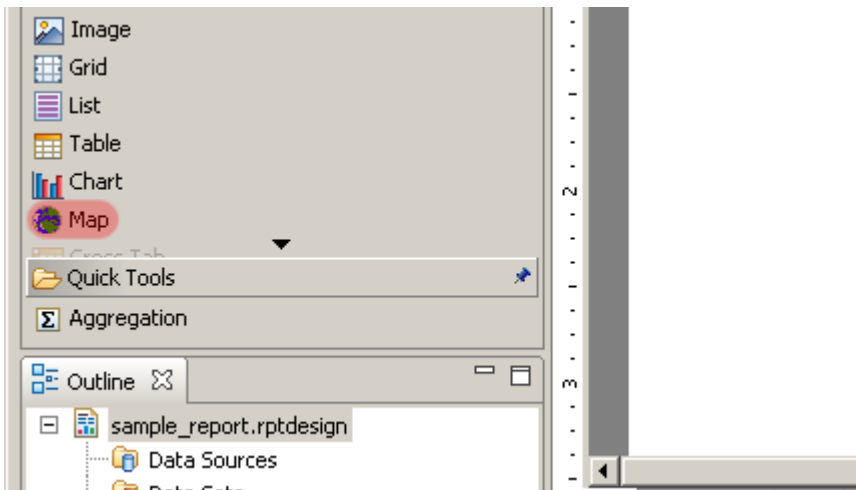
which will appear in the generated report



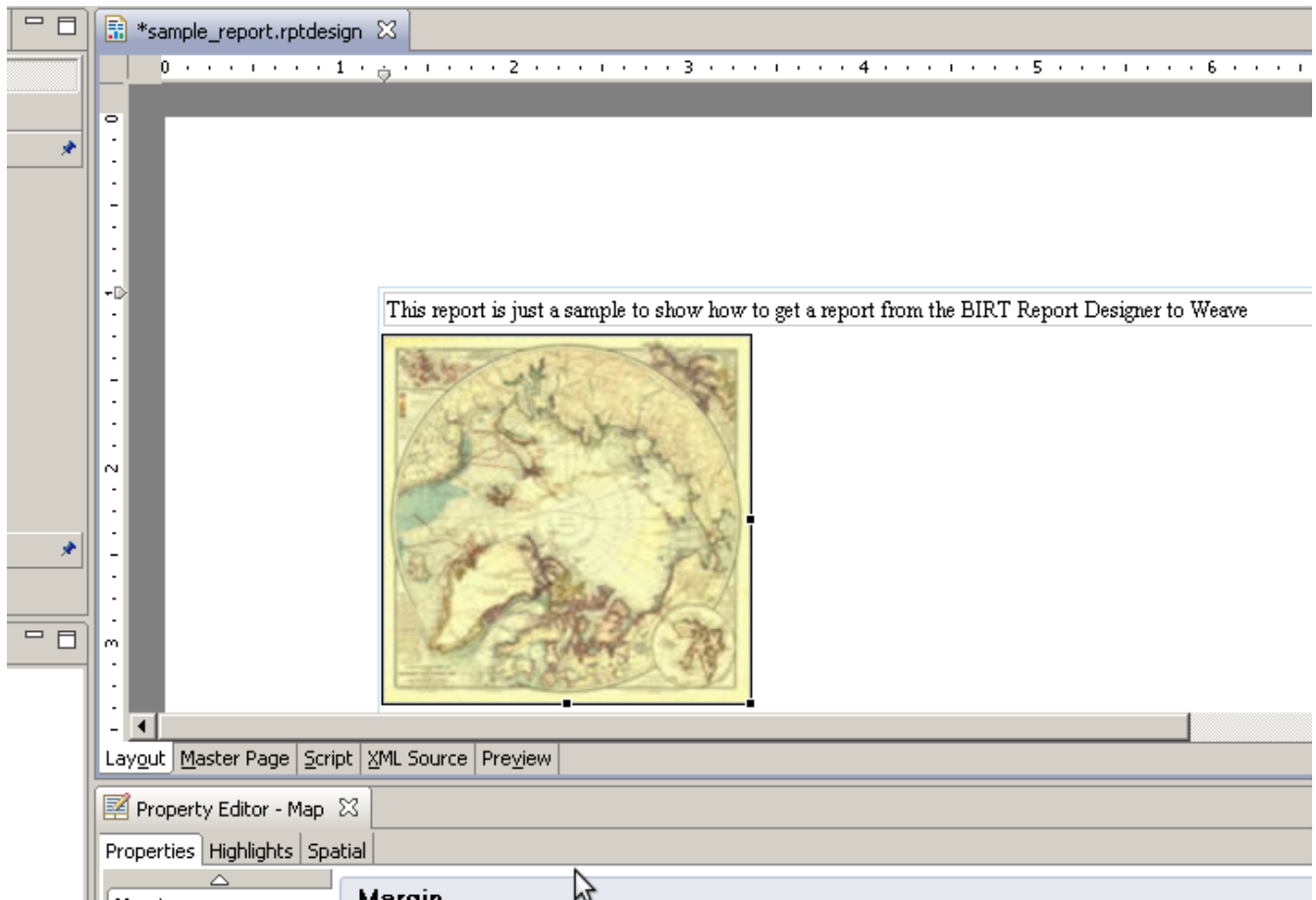
### Adding Maps to BIRT Reports

 This documentation describes the mapping components provided as part of Weave 2.3, for changes in Weave 2.4 see [BIRT Map Updates for Weave 2.4](#)

The BIRT designer installed along with Weave contains a map component in the tool palette for adding a map image to the generated report.



By selecting the tool and clicking in the report design you can add a new map place holder.

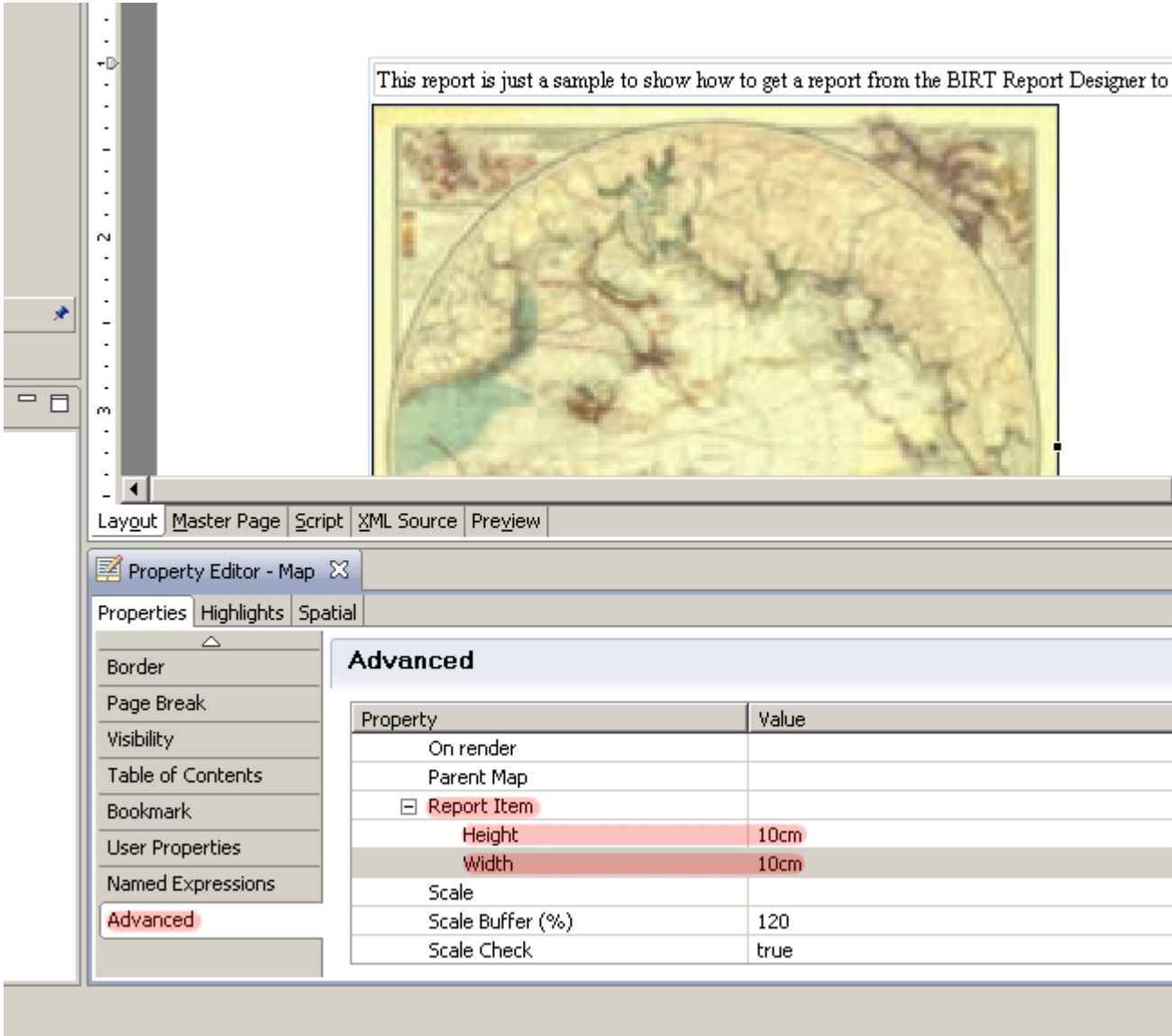


The map place holder will be replaced by a map when the report is generated.


### *Resizing the map*

You can resize the map using the handles directly within the design or you can set the width and height exactly by editing the advanced properties of the map

This report is just a sample to show how to get a report from the BIRT Report Designer to



Property	Value
On render	
Parent Map	
<input type="checkbox"/> Report Item	
Height	10cm
Width	10cm
Scale	
Scale Buffer (%)	120
Scale Check	true

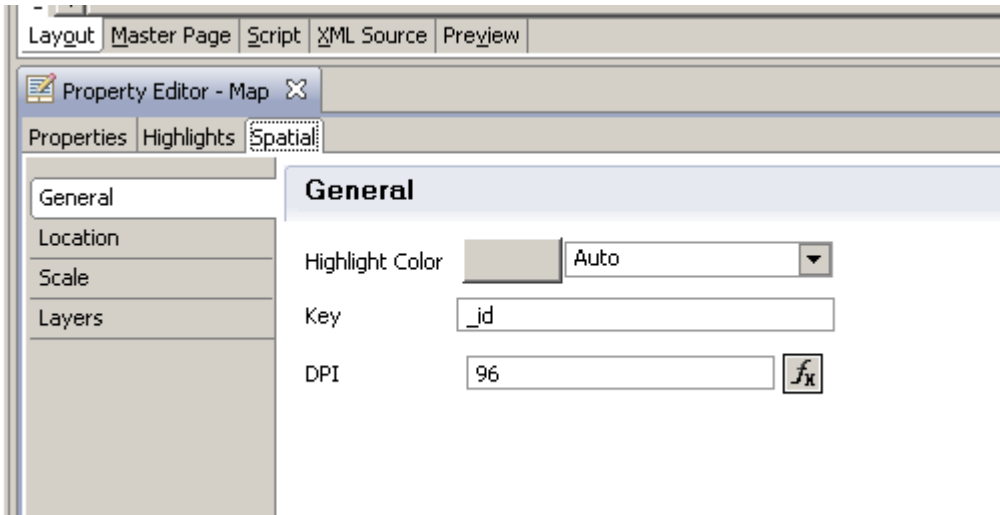
 Currently the map size must be set to specific units (cm, pixels, etc) and can not be set as a percentage.

### Map spatial settings

The Spatial tab in the map properties contains settings that adjust the display of the generated map.

General spatial settings

The general tab contains three setting.



The first is the colour to use to draw highlighted features (this is different from the colour used for drawing selections). This is used in the situation when a single feature within a selection needs to be highlighted in a BIRT report (i.e. when a map is contained within a report that will generate a map image for each selected feature).

The second is used when the map is embedded in a dataset and will be covered later.

And the third setting sets the DPI of the generated map image. The little button to the right of the DPI field allows the value to be set from a formula, which can be used for example to allow the user to choose between high, medium and low quality map output when they generate the report by associating the DPI value with a report parameter.

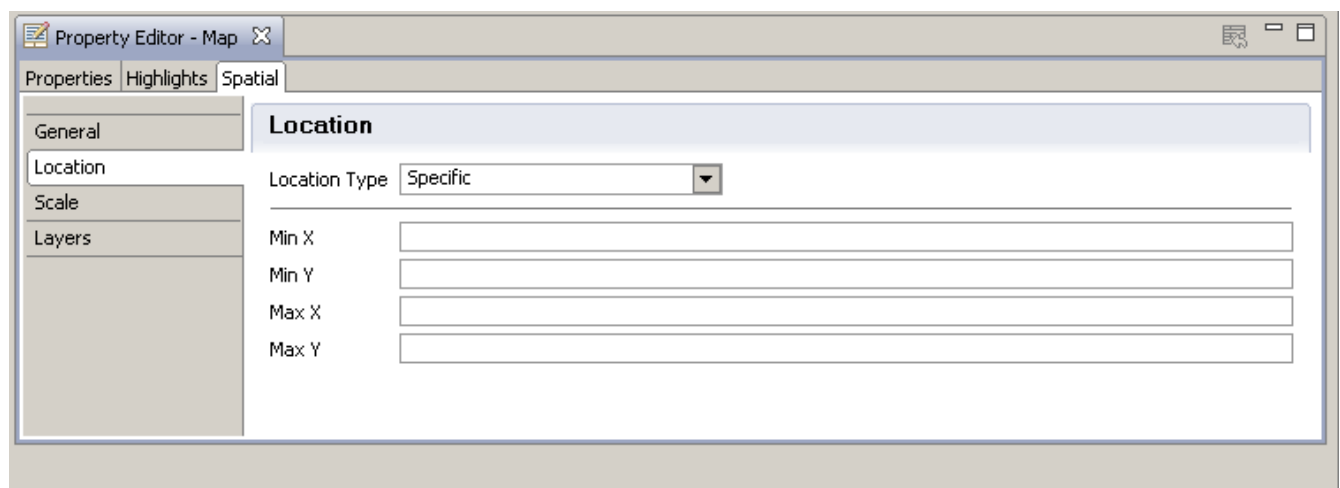
As of Weave 2.5.8 there is a fourth setting, `Show Highlight`, that allows you to turn off the highlighting defined by the `Highlight Color`.

#### Location settings

The Location setting for the map determines where the map will be centred and is used in conjunction with the Scale setting to determine what area of the world the map will cover.

There are three choices:

Value	Description
Original	Centre on the area that the user was viewing. This will ensure that the map covers <i>at least</i> the area the user was looking at, it may be more vertically <i>or</i> horizontally because the size of the map on the page may be different to that of the browser. There are no input parameters for this option.
Selection	Centre on the current selection. This will pan the map so that it is centred on the current selection. This would normally be used in conjunction with the Scale setting of the same name. There are no input parameters for this option.
Specific	Match an area entered directly. The report designer must enter the area of the map to be displayed directly.

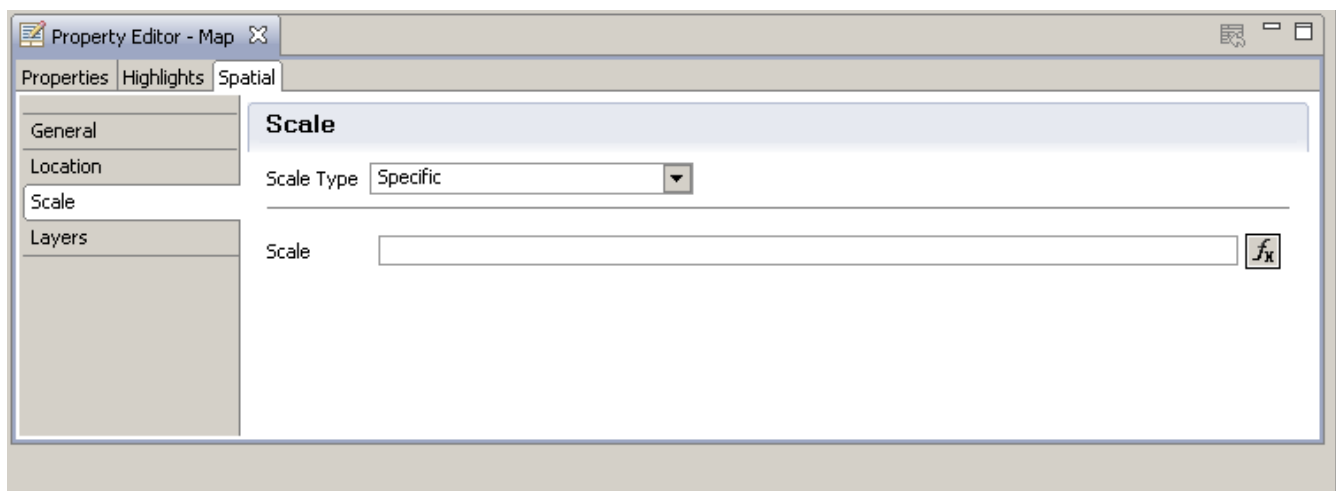


#### Scale settings

The Scale setting for the map determine what scale the map will be generated at and are used in conjunction with the Location settings to determine the actual area of the world the map will cover.

There are three choices:

V a l u e	Description
Or i g i n a l	Match the scale that the user was viewing. This basically does nothing to calculate a scale but instead uses the current location (as set by the Location setting) to determine the map scale. This may not be the exact scale the the user had selected on the client because of differences in page and screen sizes, if you want the map scale to be exactly what the user was viewing then the Specific value should be chosen for this option and a formula should be used to set the scale. There are no input parameters for this option.
Se l e c t i o n	Set the scale to ensure the map covers at least the current selection. This will scale the map so that it displays the current selection. This would normally be used in conjunction with the Location setting of the same name. You can also specify the size of a buffer around the selection so that the map covers more of the surrounding area
Sp e c i f i c	Set the scale directly. The report designer must enter the scale of the map to be displayed directly.



#### Setting a map scale

Like the DPI setting the scale setting, when set to Specific, allows its value to be set from a formula. Automatically matching the users current scale

One option when using a formula is to set the formulas value from a parameter that's automatically set by Weave each time a report is generated, the name of this parameter is 'client.scale' and the value for this parameters is always set to the users current map scale at the time of the report generation. So, by setting the Scale formula to

```
params['client.scale']
```

you can ensure that the map in the report will be generated at the same scale that the user was originally using.

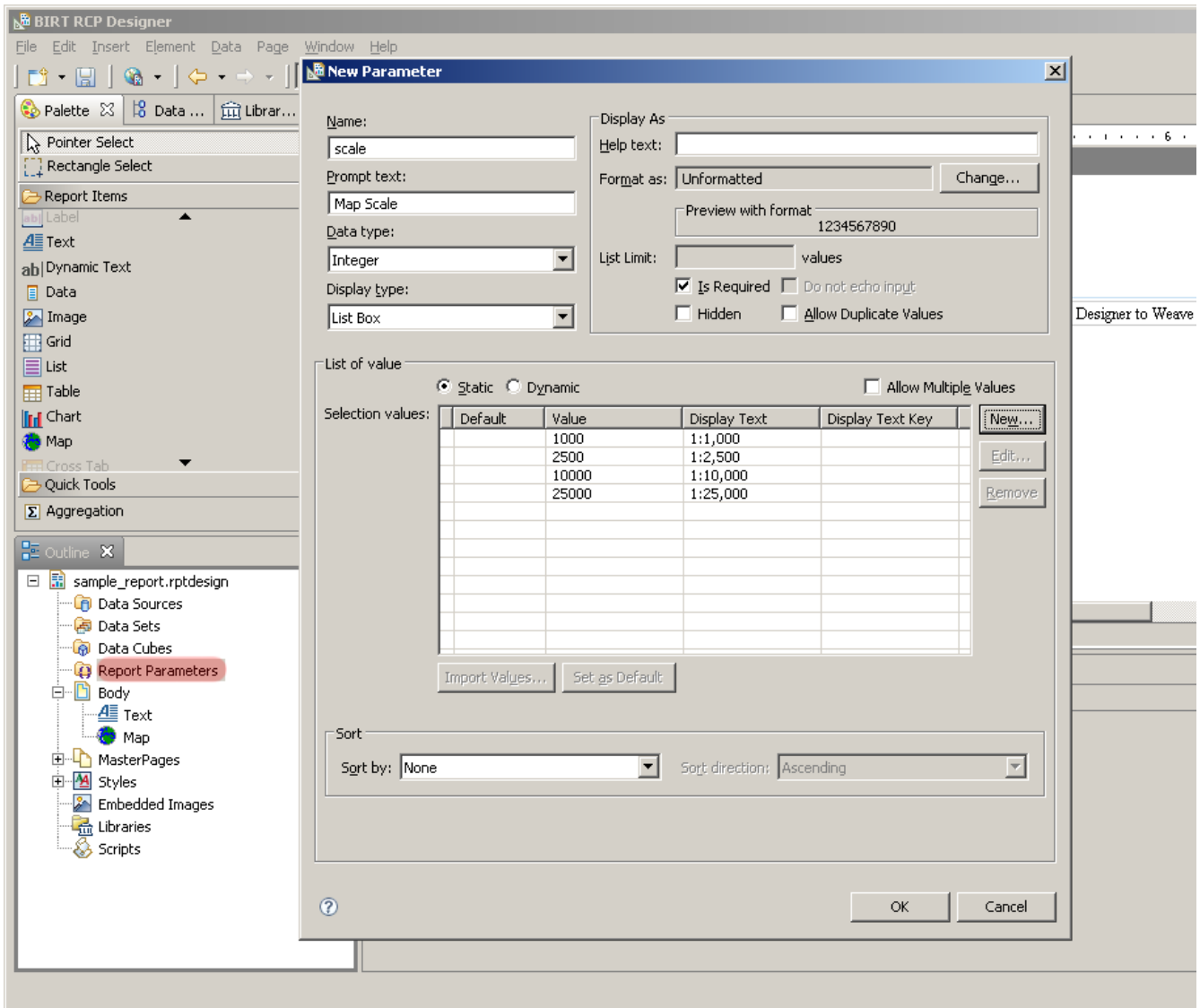
**⚠** This is different from setting the Scale Type to Original, which actually maintains the original extent rather than the original scale. Using 'client.scale' maintains the original scale which may result in a different map extent from what the client was viewing.

**i** This may be incorporated directly as a new Scale option in future versions of the BIRT mapping component

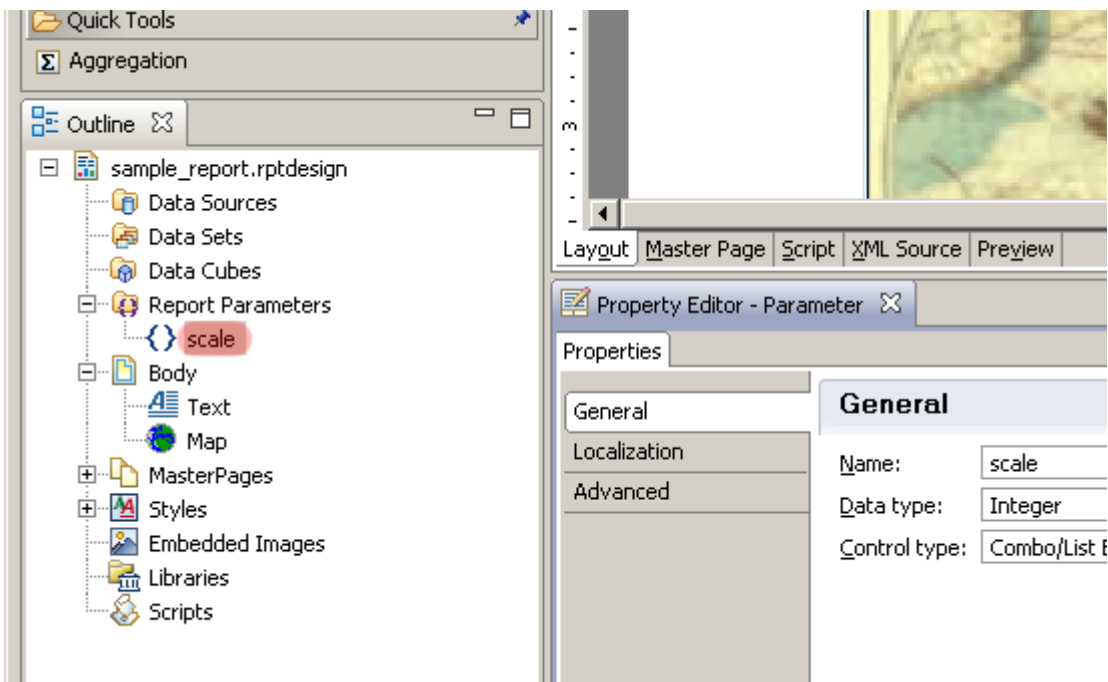
From a value entered by the user

Another option is to add a report parameter to the report and allow the user to choose a map scale when they generate the report (note that this is different from 'client.scale' which is automatically generated by Weave).

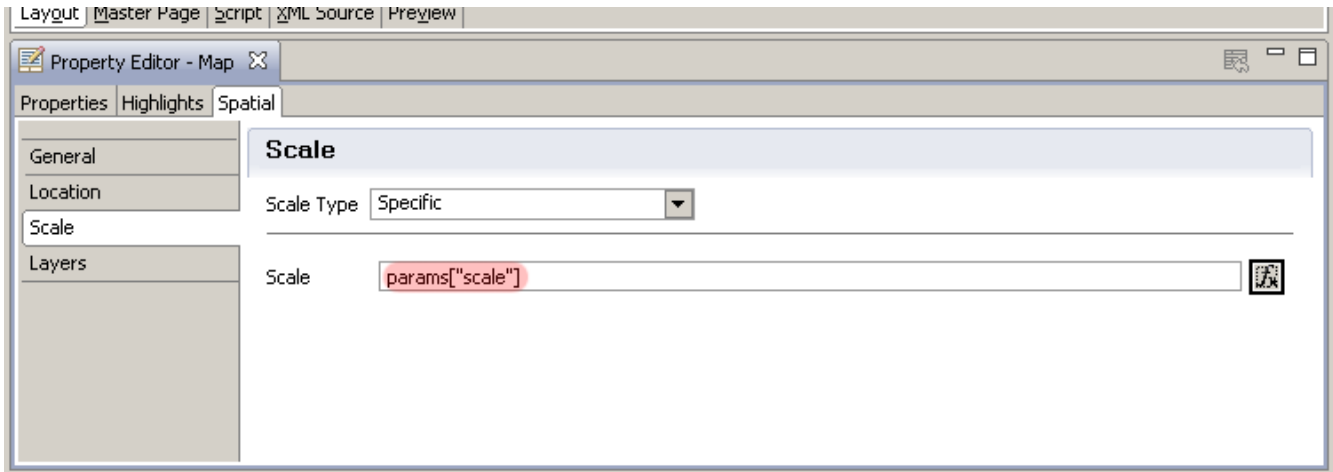
By right clicking on the Report Parameter item in the Outline you can create a new report parameter



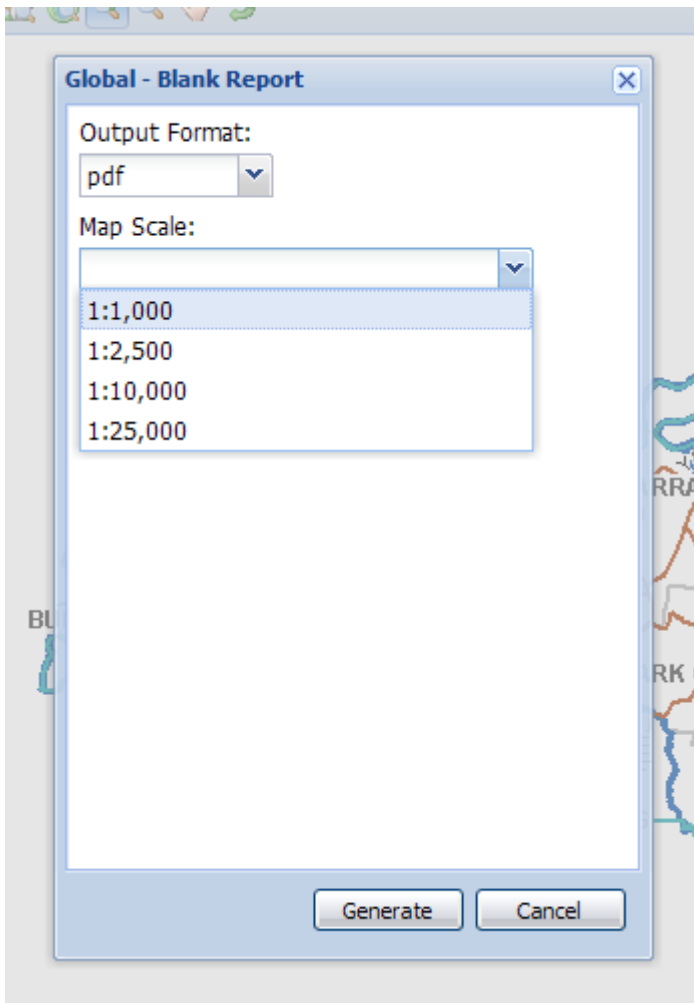
Here we've created a new report parameter called 'scale'



which we can then use in the formula for our report parameter (by using the value `params["scale"]` as our formula)



and it will appear when the user generates the report.



then the map scale will be taken from the value the user has selected.

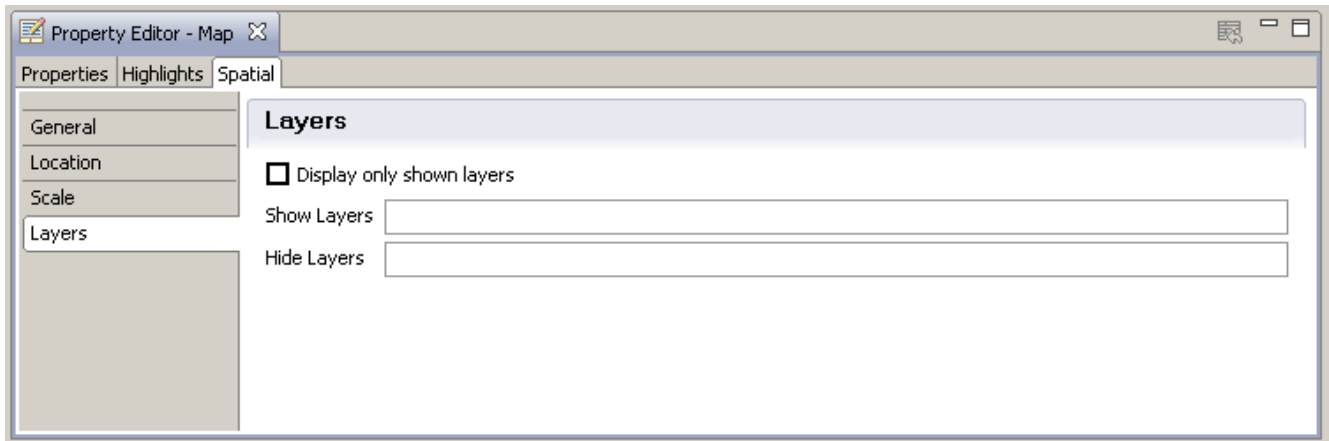
This is also a useful option if you wish to restrict the scales available for generating a report map.  
Displaying the map scale

All of these parameters can also be used to display the map scale using a dynamic label, but as of Weave 2.4.3 there is also an additional parameter available, after the map is generated, that contains the actual map scale that the map is generated at.

The parameter will be named `scale`, if the map doesn't have a name, or `<name>.scale` if the map does have a name. The numeric value is directly available as `params["scale"]` (or `params["<name>.scale"]`) or a formatted string value available using `params["scale"].displayText` (or `params["<name>.scale"].displayText`).

Layers Settings





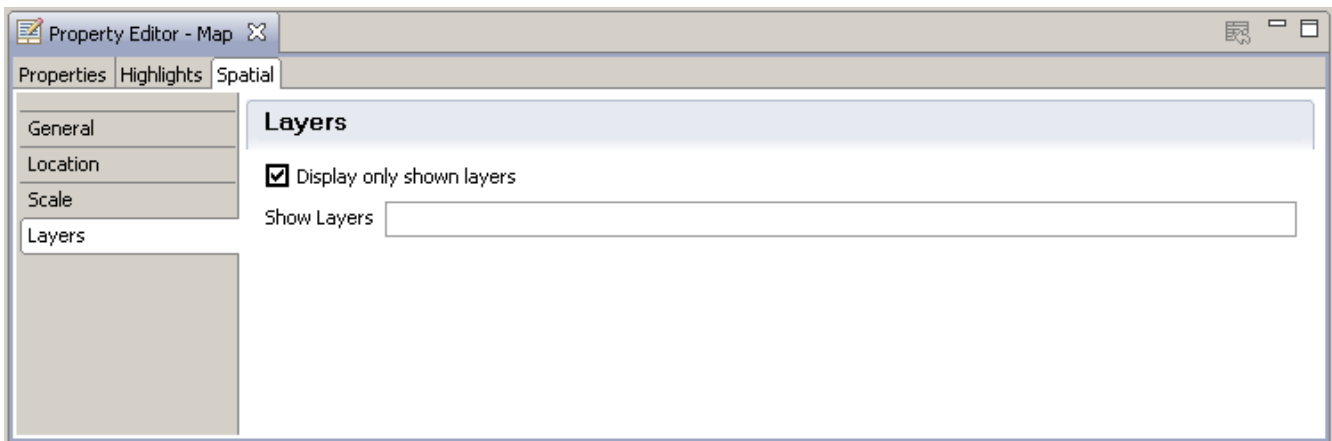
The layers settings allow you to alter the layers that are displayed on the map.

By entering a comma separated list of layer ids into the Show Layers value you can force layers to be turned on.

**⚠** This will just turn the layer on, if the map engine still won't show the layer, for example because it is outside of a certain scale range, then the layer may still not display.

Similarly by entering a comma separated list of layer ids into the Hide Layers value you can force layers to be turned off.

Finally by checking the Display only shown layers check box you can provide a list of exactly the layer ids that should be turned on.



#### *Layer ids for showing/hiding map layers*

The values for the layer ids are taken from the map engine that the layer belongs to, for example if the map engine is ArcIMS then the layer id will be the 'id' parameter for the layer in the AXL file, and are generally the unique identifier for the actual layer.

If the map is going to be created from multiple overlaid map engines and the layer id is not unique then you can specify which map engine the layer id corresponds to be suffixing the layer id with a "/" and the id of the map engine, for example "roads/me.vector" (the id of the map engine is the Weave map engine id).

**i** In reality unless you specify a map engine id explicitly each layer id is actually suffixed with "/\*", which will match any map engine id. This will effectively remove all layers matching the specified name from all map engines.

So using a layer id of "property" is the same as using "property/\*".

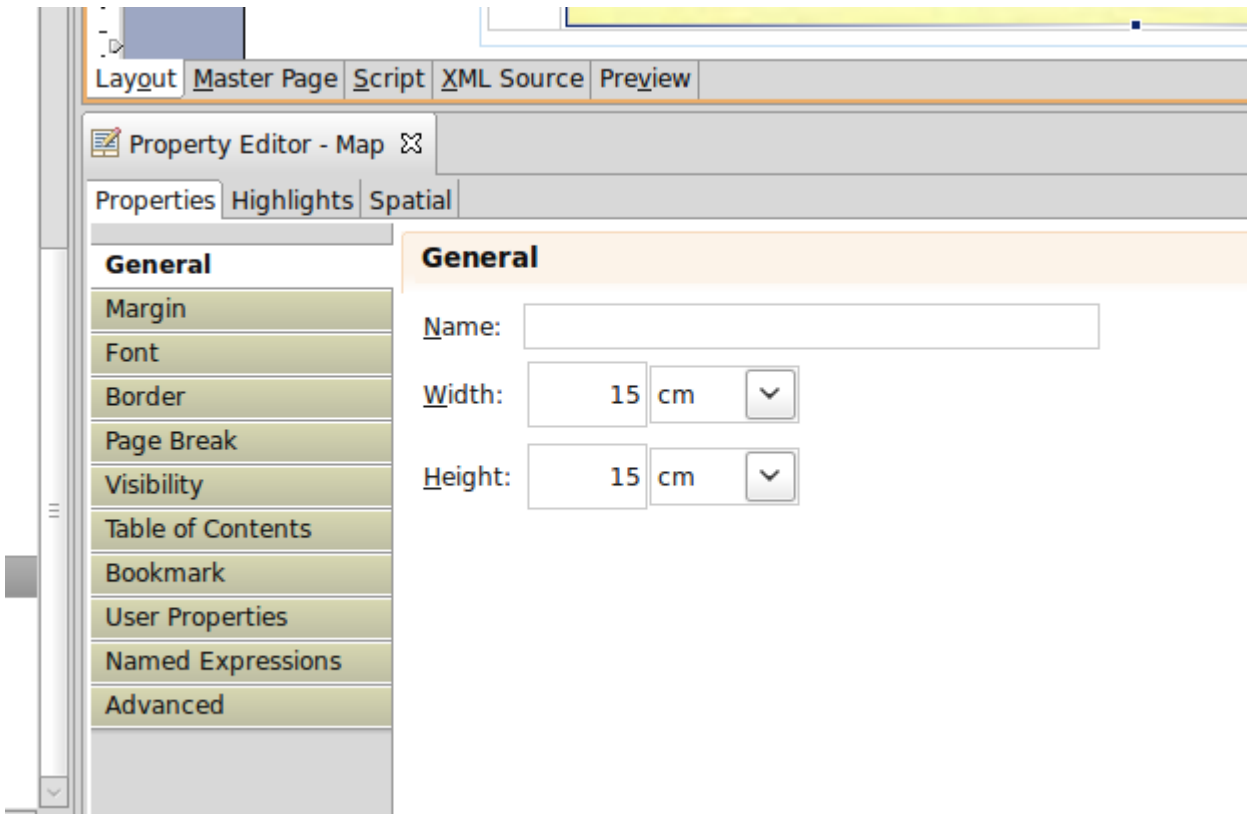
**⚠** It's not yet possible to specify "\*/mapengineid", that is indicate that all layers from a particular map engine should be on or off.

#### *BIRT Map Updates for Weave 2.4*

As part of the update in Weave 2.4 BIRT has been updated from version 2.2.1 to 2.6.1, which in itself provides a number of enhancements and bug fixes, but also allowed for some changes to the way that the mapping components works.

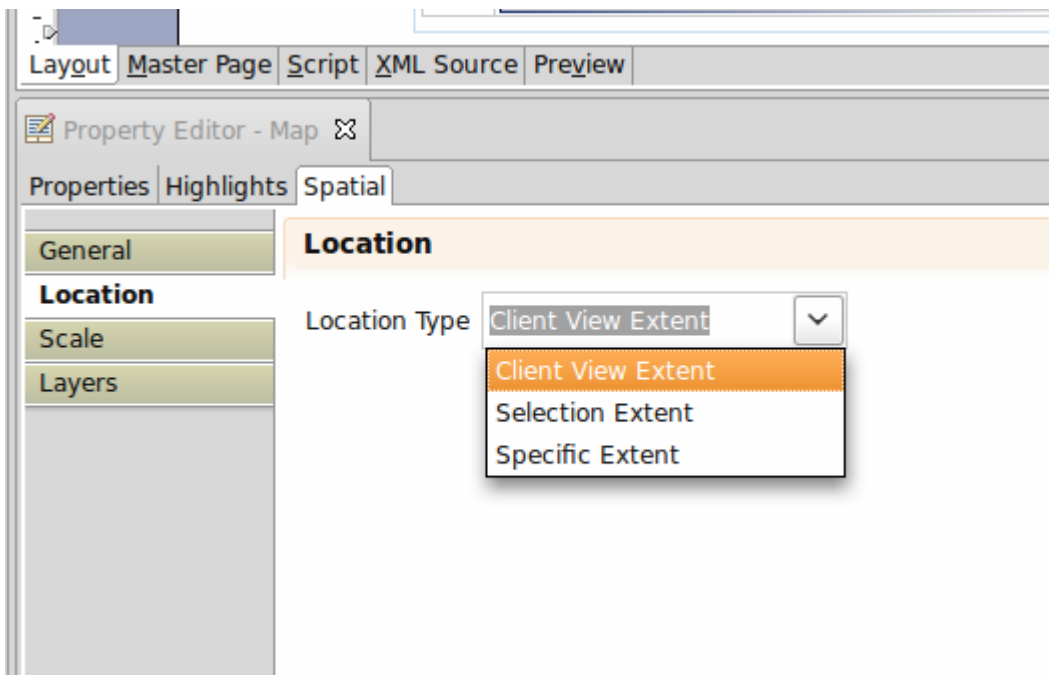
#### *Setting map size*

You can now set the size of the map component via the properties editor.



*Changed the wording for the location types*

The names of the options for determining the centre of the map have been changed to hopefully make it more obvious what they're doing



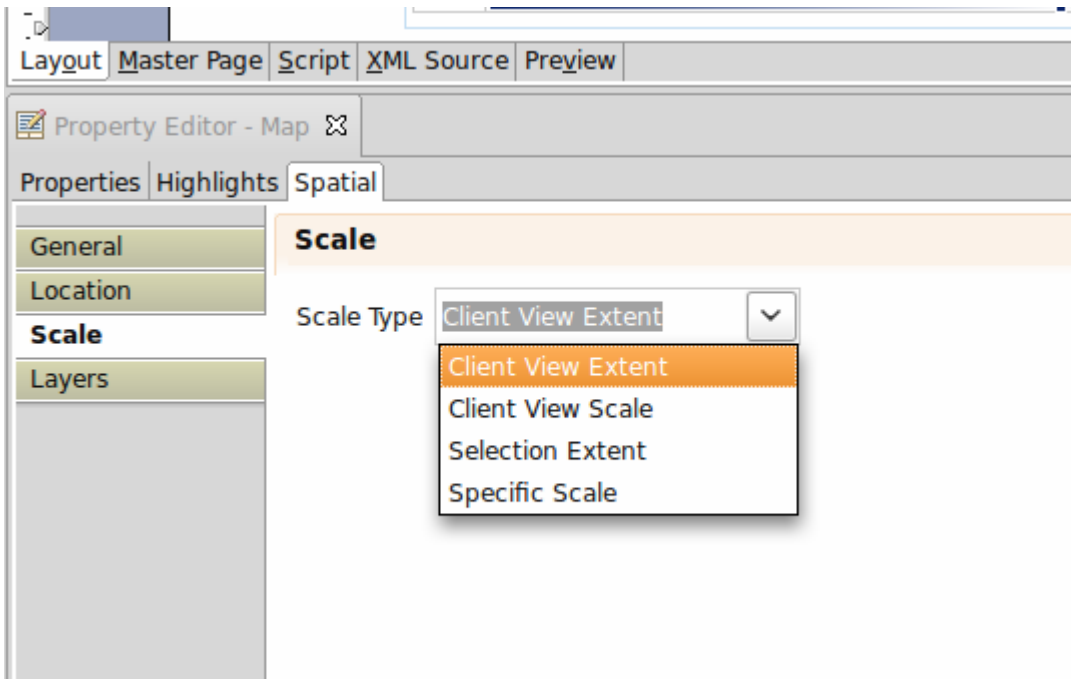
*Client View Extent* is was previously called *Original*, and matches the center of the map to be the same as what the user was viewing.

*Selection Extent* was previously called *Selection*, and matched the center of the map to be the center of the current selection.

*Specific Extent* was previously called *Specific*, and allows you to enter an extent directly (not that if this is chosen the scale type will be ignored).

*Changed the wording for the scale types and added a new one*

The names of the options for determining the scale of the map have been changed to hopefully make it more obvious what they're doing



*Client View Extent* was previously called *Original*, and matches the scale of the map to include at least the current map content that the user was viewing.

*Client View Scale* is new and matches the map scale to what the user was viewing.

*Selection Extent* was previously called *Selection*, and matches the scale of the map to include the currently selected features.

*Specific Scale* was previously called *Specific*, and allows you to enter a scale value directly, either as a scale or as a formula.

*Client View Scale* was introduced to make it easier to create a map at the same scale as the user was viewing, as opposed to *Client View Extent*, which tries to match the extent that the user was viewing which may result in a different map scale.

#### *Improved support for optional parameters when setting the scale*

Previously you could add a report parameter to allow the user to choose the map scale, and then use the parameter value with a *Specific Scale* scale type to set the map scale explicitly, but this required the user to always enter a scale value (and forced you to ensure the scale parameter was set as "required").

It's now possible to have the scale report parameter not be a required parameter and have the map fall back to either *Client View Extent* or *Client View Scale* if the user doesn't choose a value.

Assuming that you've created an optional report parameter called 'scale'

**Edit Parameter**

**Name:**

**Prompt text:**

**Data type:**

**Display type:**

**Display As**

**Help text:**

**Format as:**

**Preview with format:**

**List Limit:**  values

**Is Required**  Do not echo input

**Hidden**  **Allow Duplicate Values**

**Selection list values**

**Static**  **Dynamic**  **Allow Multiple Values**

Default	Value	Display Text	Display Text Key
	500	1:500	
	1000	1:1,000	
	2000	1:2,000	
	4000	1:4,000	
	8000	1:8,000	
	16000	1:16,000	
	32000	1:32,000	
	64000	1:64,000	
	128000	1:128,000	

**Sort**

**Sort by:**  **Sort direction:**

you can use the following formula

```

if ( params["scale"].value )
    params["scale"]
else
    null

```

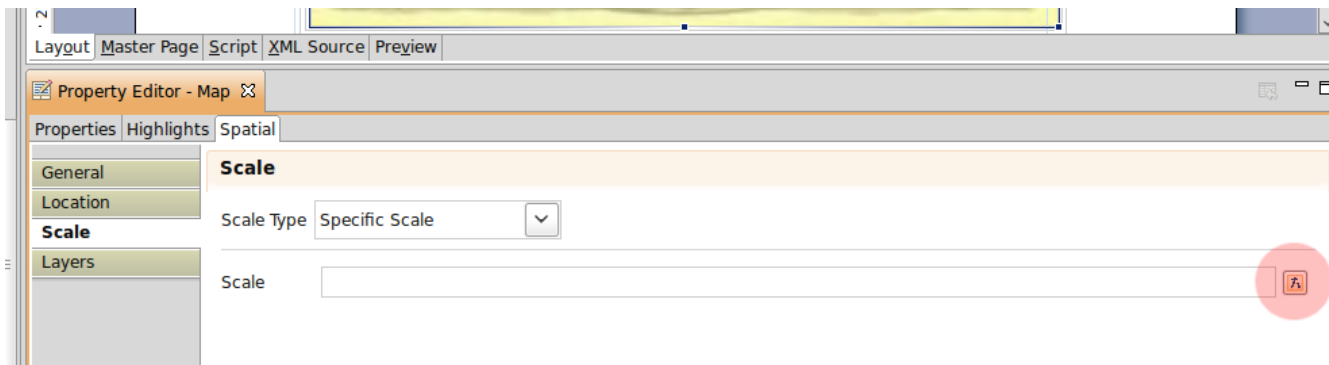
to use the current view extent, and the following formula

```

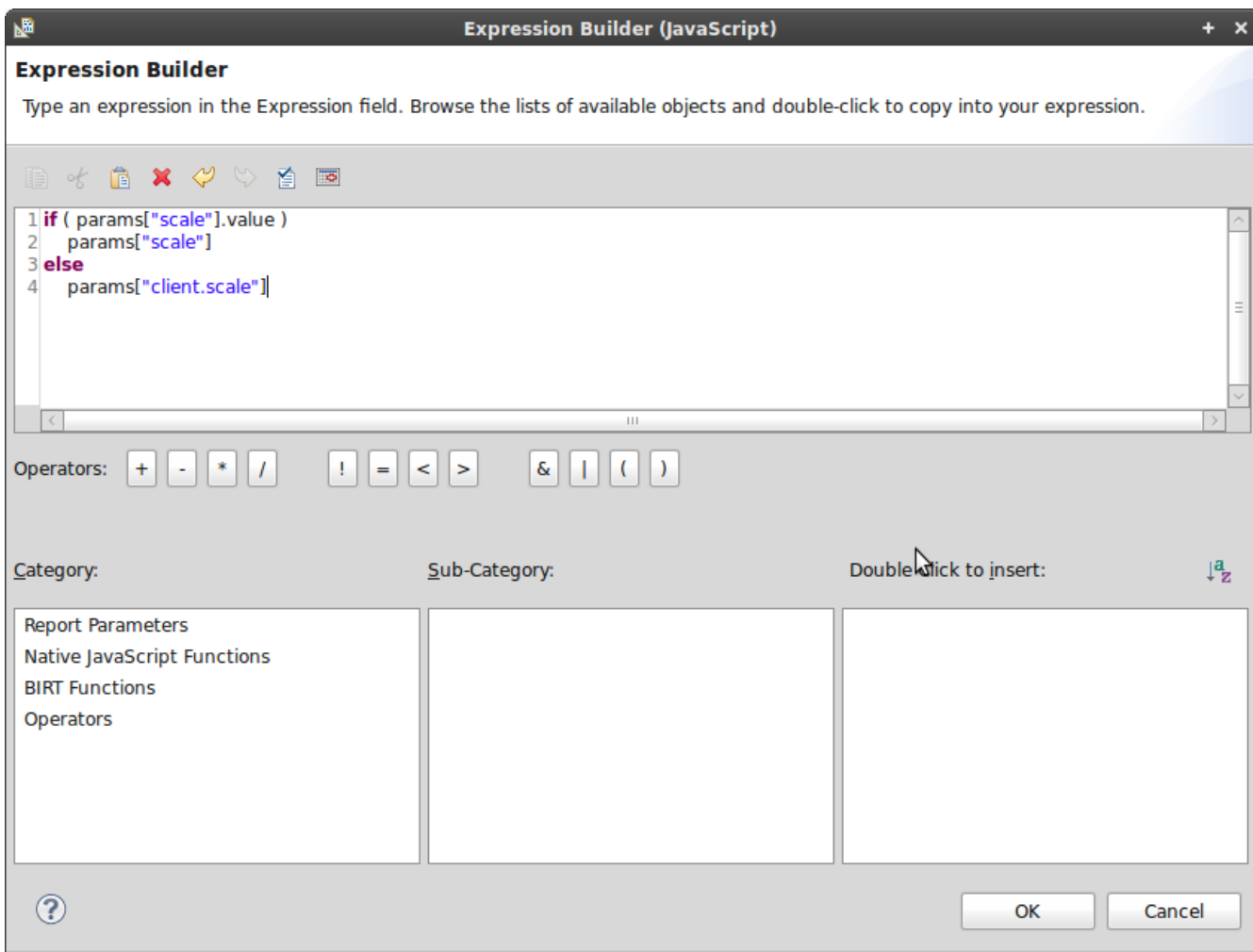
if ( params["scale"].value )
    params["scale"]
else
    params["client.scale"]
    
```

to use the current view scale ("client.scale" is an auto-generated parameter created by Weave to represent the current map scale)

This can be done by using a Specific Scale and hitting the formula button



The pasting the formula into the formula builder that appears



*Displaying generated map scale*

As of version 2.1.0 of the com.cohga.server.report.birt.map bundle (Weave 2.4.3) a new parameter is available, after a map component is generated, that provides the value of the scale that the map was actually generated at. The parameter will be named `scale`, if the map doesn't have a name, or `<name>.scale` if the map does have a name.

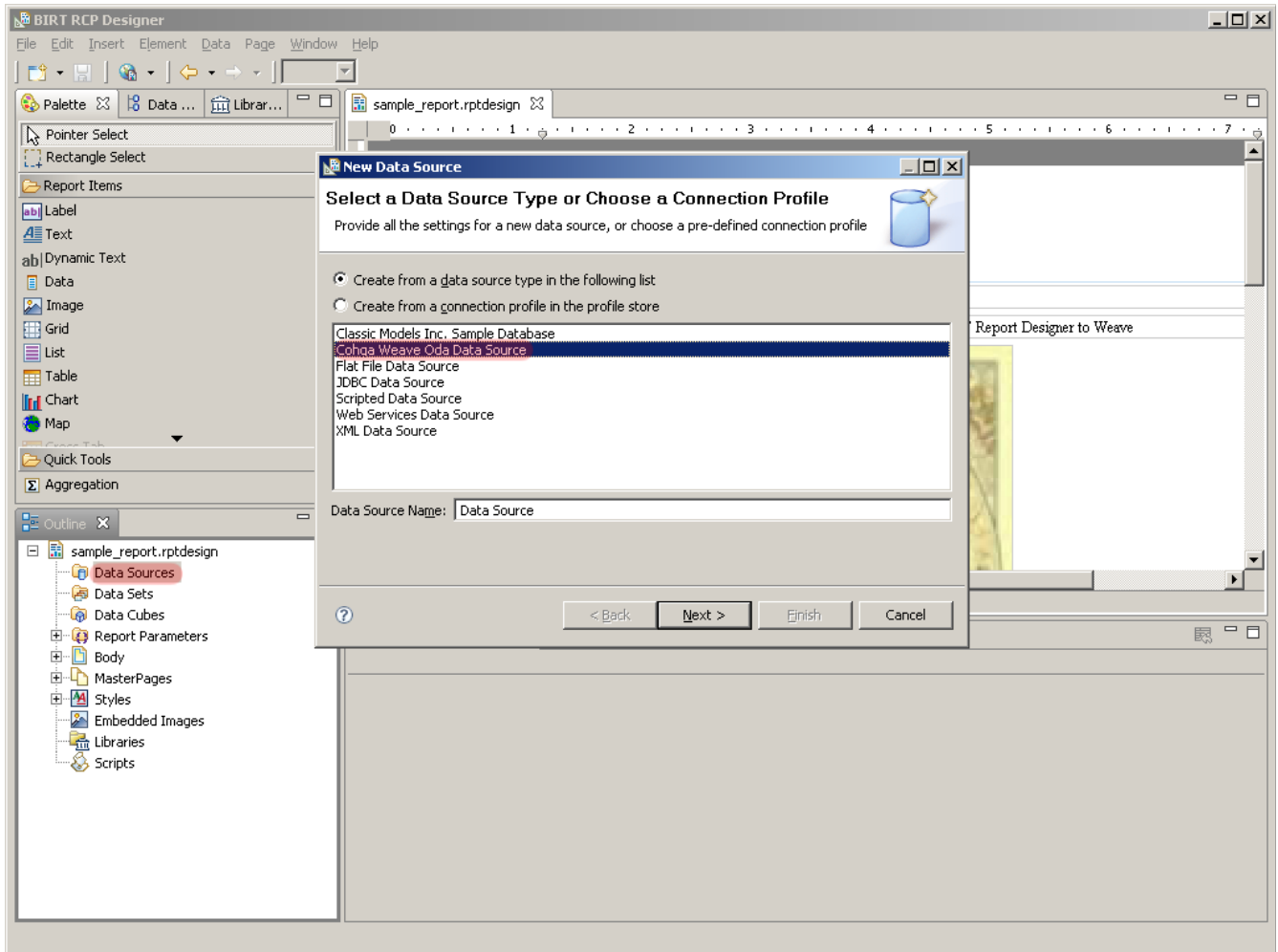
The numeric value is directly available as `params["scale"]` (or `params["<name>.scale"]`) or a formatted string value available using `params["scale"].displayText` (or `params["<name>.scale"].displayText`).

## Adding Data to BIRT Reports

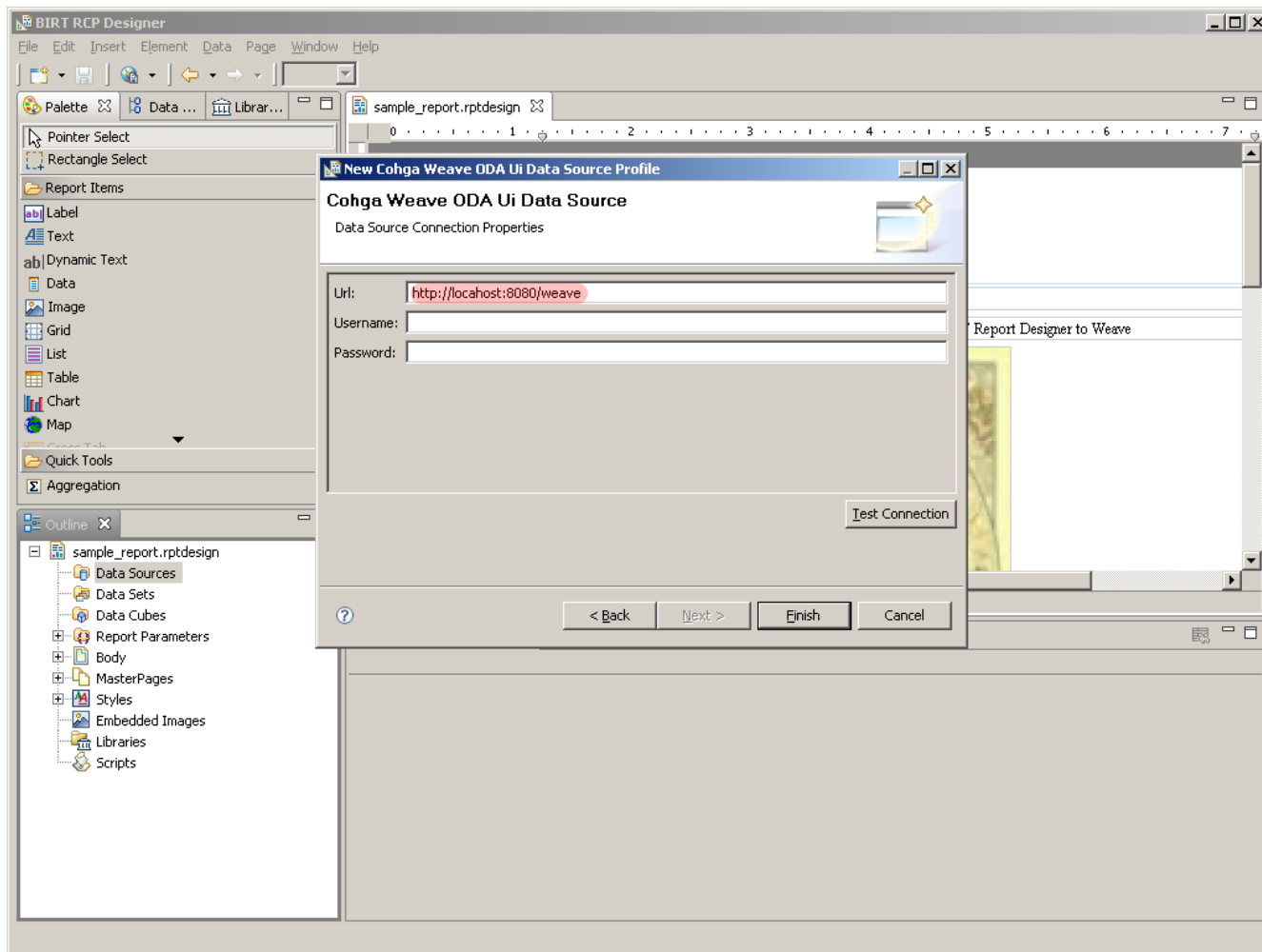
The BIRT designer that is installed with Weave contains a Weave Data Source component that's available for adding data generated by Weave to a report.

Create a Data Source

Adding data to a BIRT report that is generated by Weave is done in the same manner as data from any other data source that BIRT supports, but you need to choose the "Cohga Weave Oda Data Source" driver when creating a new data source, which is done by right clicking on the Data Source item in the Outline panel in the BIRT designer and selecting New Data Source.





From the next screen you need to enter the URL of a Weave server that is currently running and will be serving the data that you wish to include in the report.




The URL entered here is only used during report design, since when the report is actually being generated it's being generated within the Weave server itself and therefore knows how to connect.

This allows us to use localhost, as in the example above which assumes the Weave server is running on the local machine, or the host name of another Weave server and still be able to copy the report to another server running Weave.

 The username and password fields are not currently used

 The Test Connection button will check if a connection can be established to the Weave server pointed to by the URL parameter

 If the Test Connection button does not successfully connect to the server, and you know that the URL parameter is correct, it could be that the connection is being blocked by the security.xml file, particularly if you have enabled NTLM or Kerberos security (i.e. Windows single sign-on).

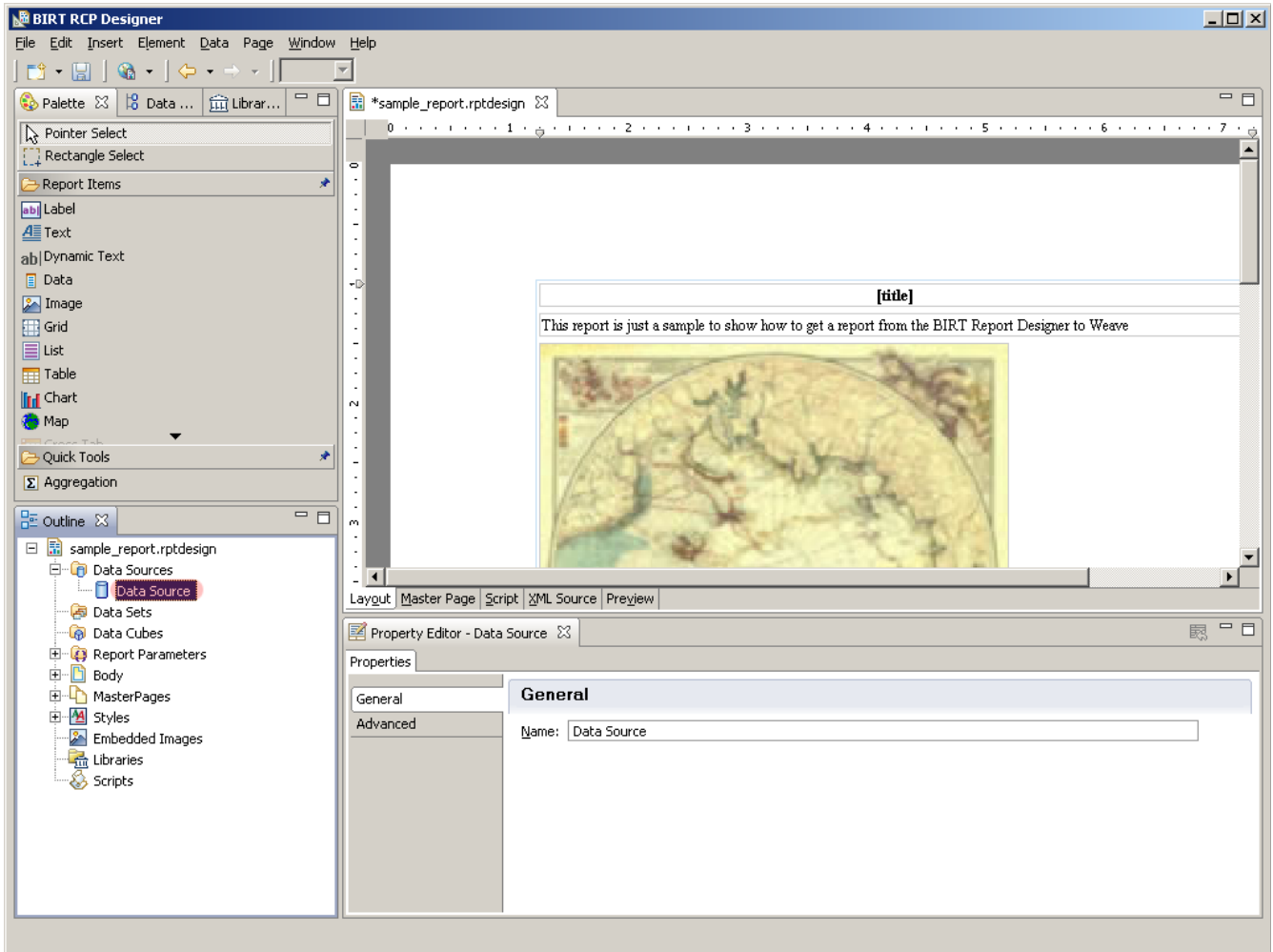
If you do have NTLM or Kerberos security enabled then you can disable it just for the BIRT report designer by ensuring that the URL pattern `/report/remote` does not have any NTLM or Kerberos filters applied, by updating the `filterChainProxy` to include the following (before the last `**` catch-all entry)

```
/report/remote=httpSessionContextIntegrationFilter,authenticationProcessingFilter,
securityContextHolderAwareRequestFilter,rememberMeProcessingFilter,anonymousProcessingFilter,
jsonExceptionTranslationFilter,filterInvocationInterceptor
```

Also, ensure that the `/report/remote` URL pattern is listed in the `objectDefinitionSource` as being accessible anonymously (again, before the last `**` catch-all entry)

```
/report/remote=IS_AUTHENTICATED_ANONYMOUSLY
```

Once you click Finish you will have a new data source connection available for retrieving data from a Weave server

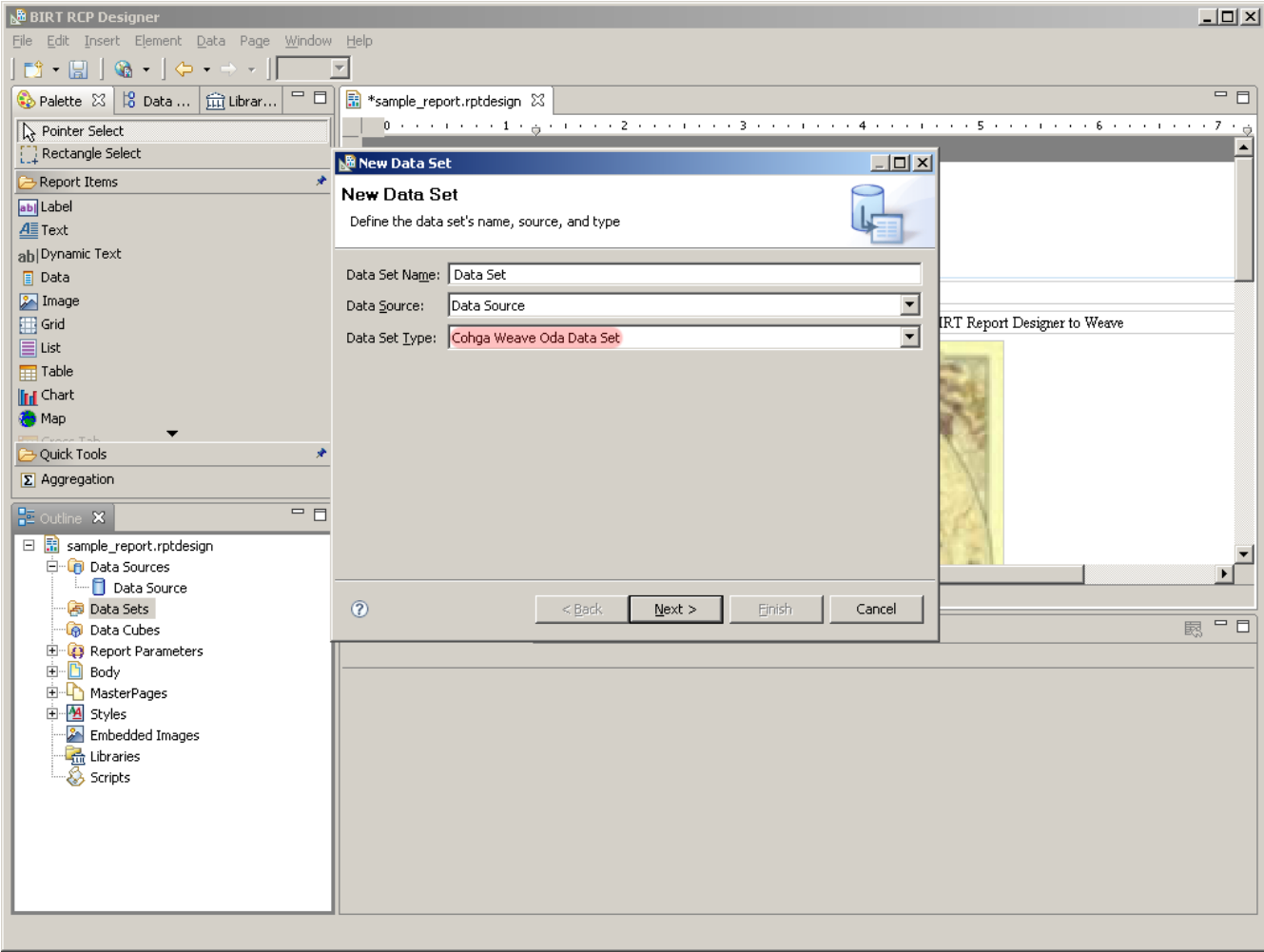


#### Creating a Data Set

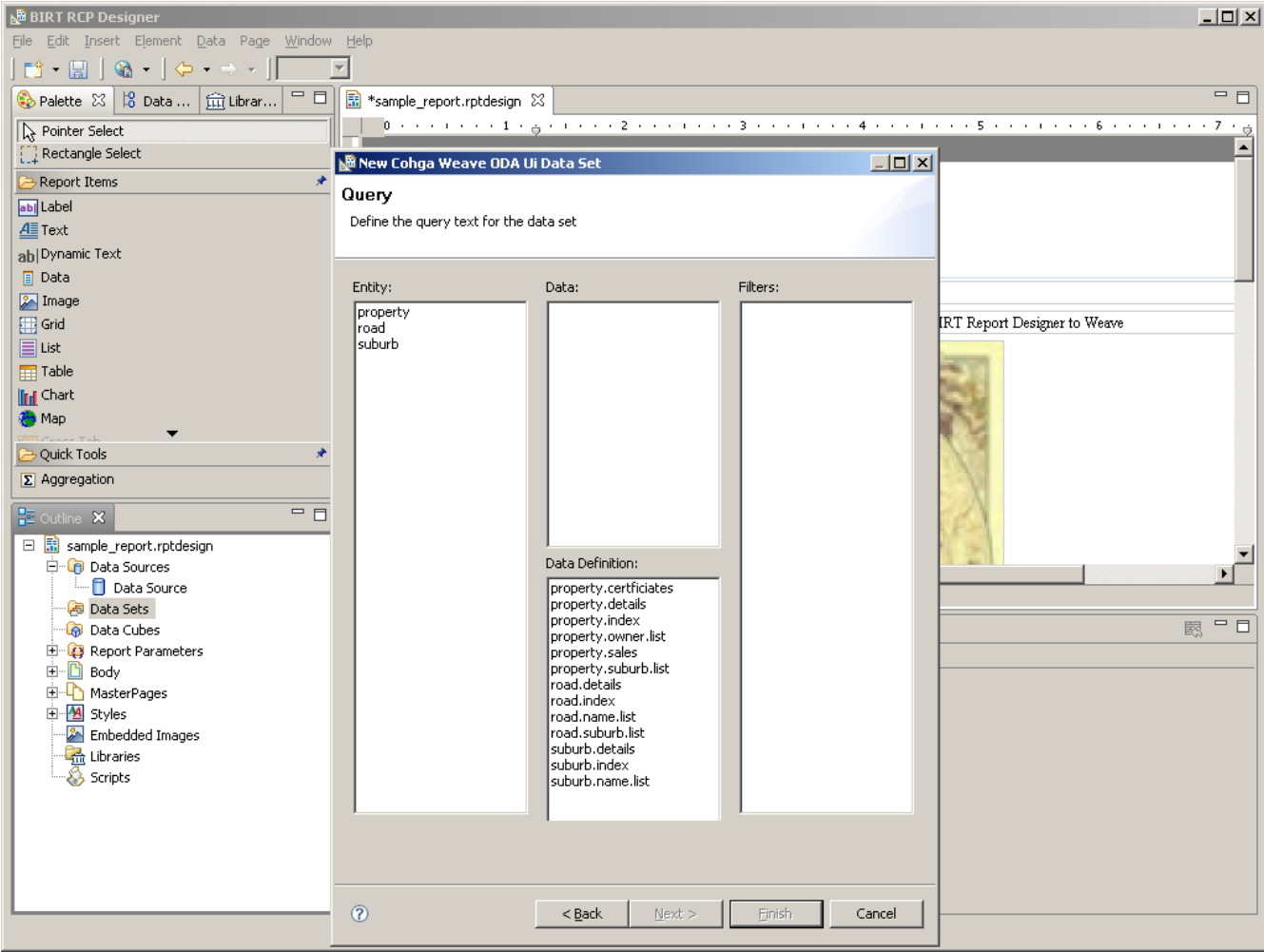
The next step is to create a Data Set, which represents a Weave data definition that can be included in the report design.

Again to create a new Data Set you right click on the Data Sets item in the Outline panel.



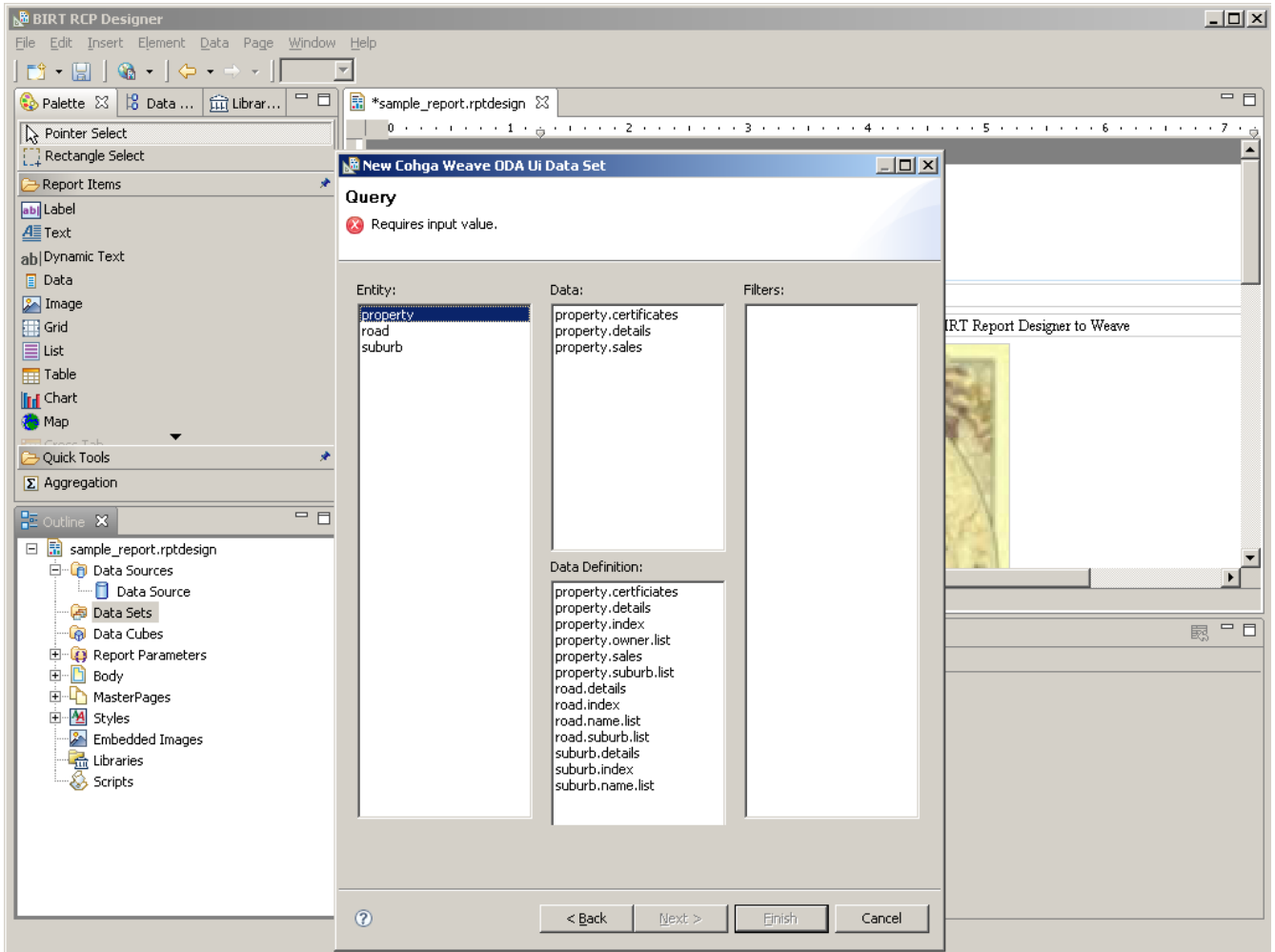



When you hit next you will get a page listing the entities and data definitions available on the server, and here you choose what data you want to retrieve.



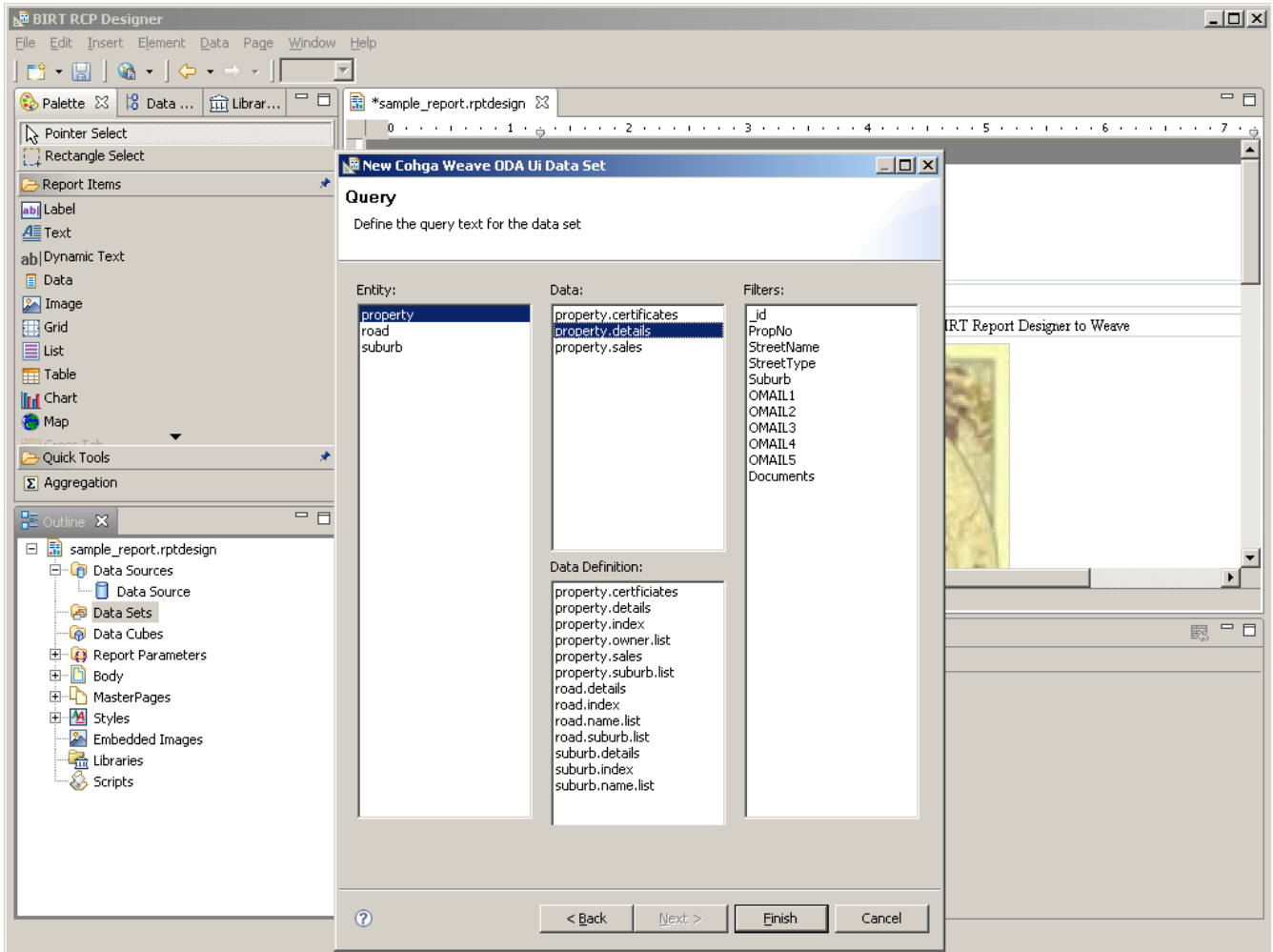
Here you can see the list of available entities in the left column and the list of all available data definition at the bottom of the middle column.

If you select one of the entities in the left column the list at the top of the middle column will populate with the list of data's that are associated with that entity.



 The error at the top of the Data Set form indicates that we must choose a Data or Data Definition now that we've selected an entity

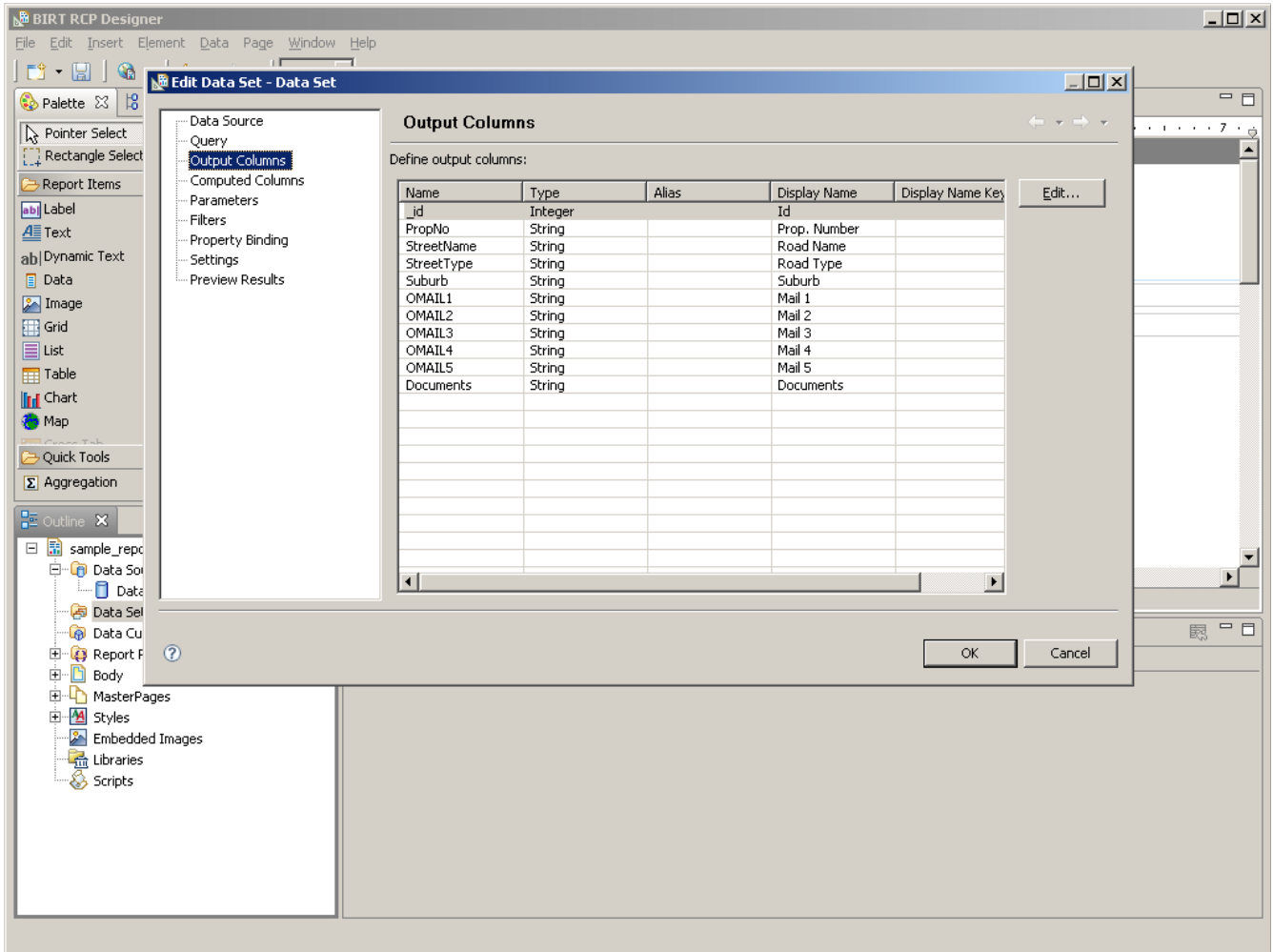
Now we need to select the Data or Data Definition that will be the actual source of the data we wish to display.



Once we've done this the Finish button is enabled again so we can complete the creation of the data definition.

**i** The third column, Filters, is used when creating a one to many data set and will be covered later. But it also gives an indication of the values that will be returned for the data set.

Once you click Finish you will progress to the standard BIRT data set editing wizard



this is where you can edit the properties for the data set and do things like filter the data, create new values based on data retrieved.

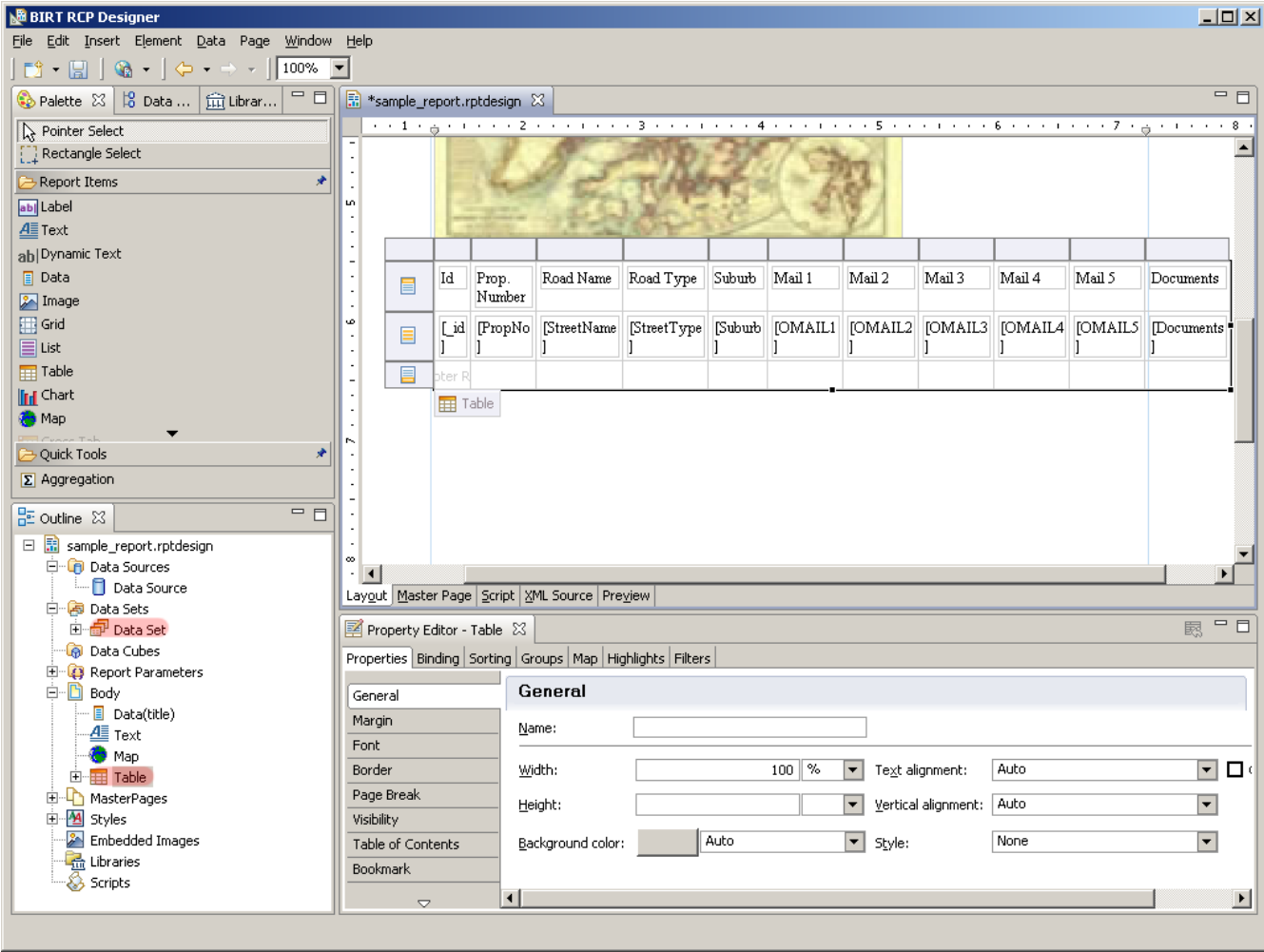
**!** One useful thing you can do in the data set edit page is set the maximum number of rows the report will generate which is done in the Setting tab. This will give you the opportunity to ensure that the user can't generate the report for 1,000,000,000 records or setup the report for a single entity.

**i** The `_id` column represents the key column for the underlying data definition and is used to uniquely identify each entity. This value will be used later when we look at filtering and embedding maps in data sets.

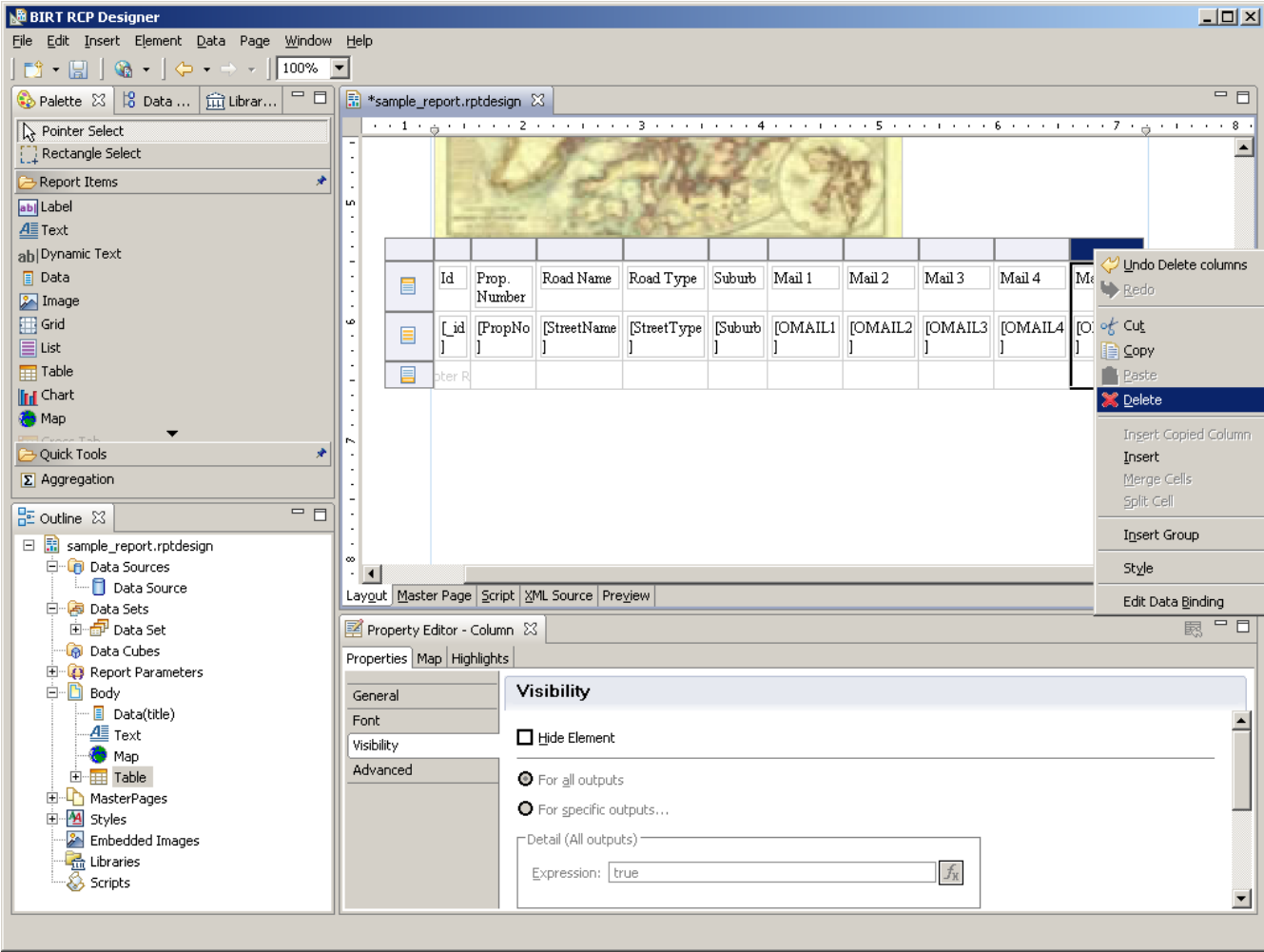
Once you click Ok here it'll complete the Data Set creation

Using a Data Set

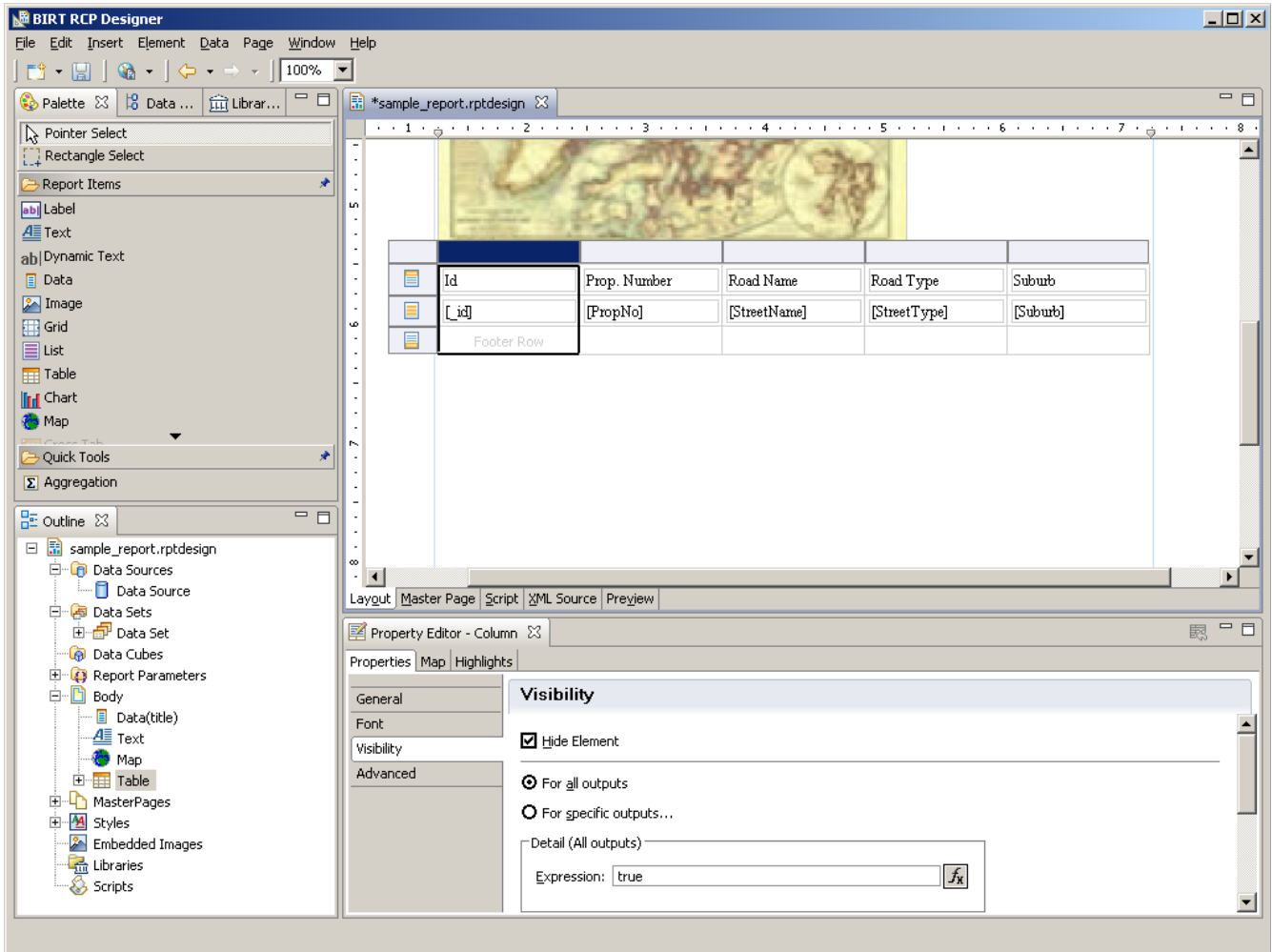
The quickest way to then get the data from a data set into the report design is to simply drag and drop the data set into the report design. This will create a new Table in the report design which represents the data set.



From here you can delete the columns you don't want to appear in the report



**⚠** You probably don't want the `_id` column to appear, but you shouldn't delete it but just hide it. Click on the column header for the `_id` column and change to the visibility tab in the properties, there you can click the Hide checkbox to hide it



Then if we save and generate this updated report we'll see the data for the currently selected entities

**Sample Report**

This report is just a sample to show how to get a report from the BIRT Report Designer to Weave

Prop. Number	Road Name	Road Type	Suburb
94-140	PORTER	STR	TEMPLESTOWE
142-164	PORTER	STR	TEMPLESTOWE
1B	NILAND	RISE	TEMPLESTOWE
10	WAKEFIELD	PL	TEMPLESTOWE
9	WAKEFIELD	PL	TEMPLESTOWE
11	WAKEFIELD	PL	TEMPLESTOWE
12	WAKEFIELD	PL	TEMPLESTOWE
13	WAKEFIELD	PL	TEMPLESTOWE
8	WAKEFIELD	PL	TEMPLESTOWE
7	WAKEFIELD	PL	TEMPLESTOWE
3	WAKEFIELD	PL	TEMPLESTOWE
2	WAKEFIELD	PL	TEMPLESTOWE
6	WAKEFIELD	PL	TEMPLESTOWE
4	WAKEFIELD	PL	TEMPLESTOWE
5	WAKEFIELD	PL	TEMPLESTOWE
19	NILAND	RISE	TEMPLESTOWE
18	NILAND	RISE	TEMPLESTOWE
20	NILAND	RISE	TEMPLESTOWE
21	NILAND	RISE	TEMPLESTOWE

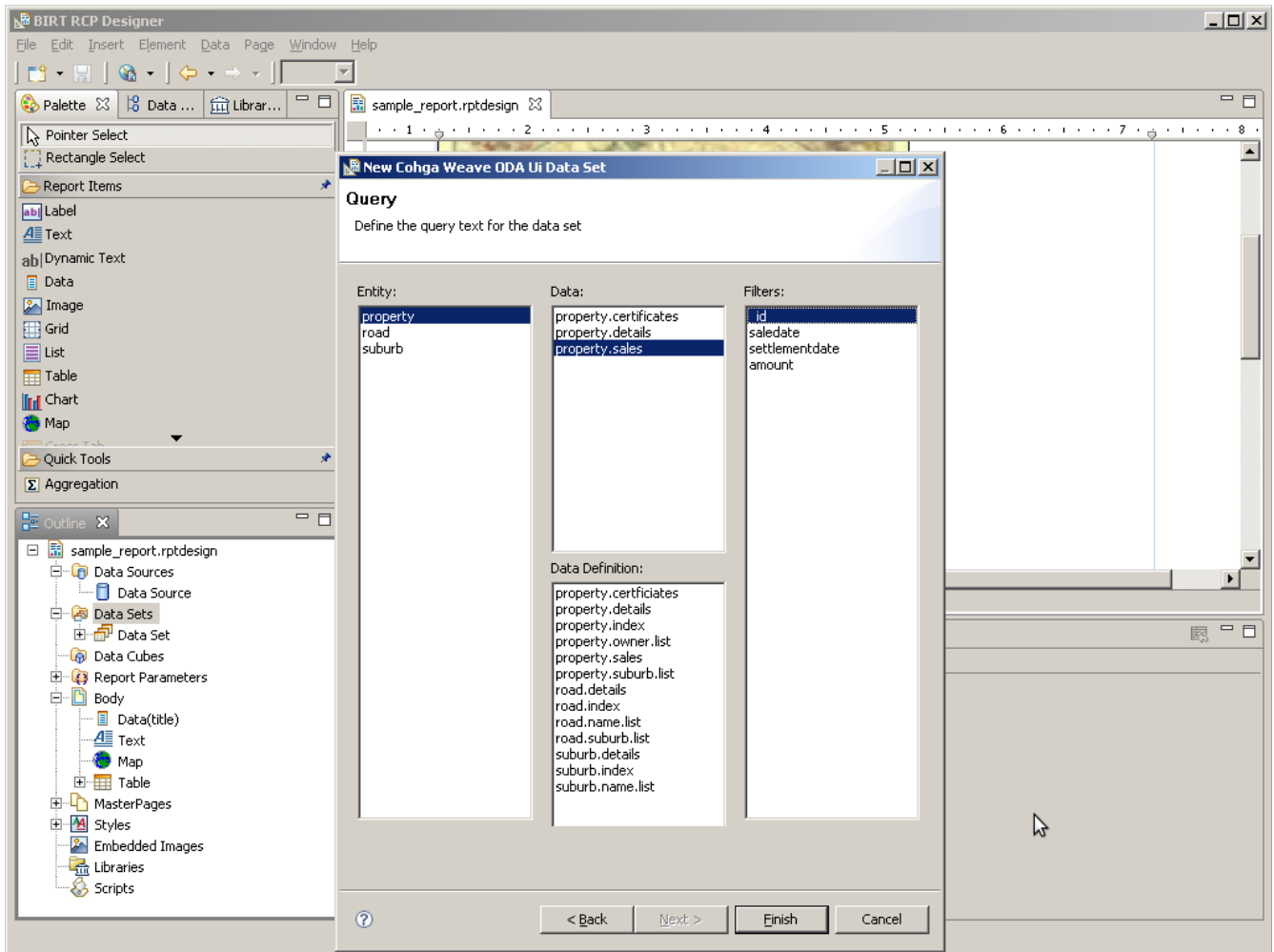
Nov 4, 2010 3:37 PM

The map component in the report design was hidden in the design



## Linking data sets

The filter setting in the data set property pages allows us to display additional details for each row of another data set, for example if we wanted to display registered animals at each property in our sample report we'd create a new data set, which would be linked to an animal registration data definition, but we'd filter it based on the property id and link that filter to the property id of the data set we've already created.

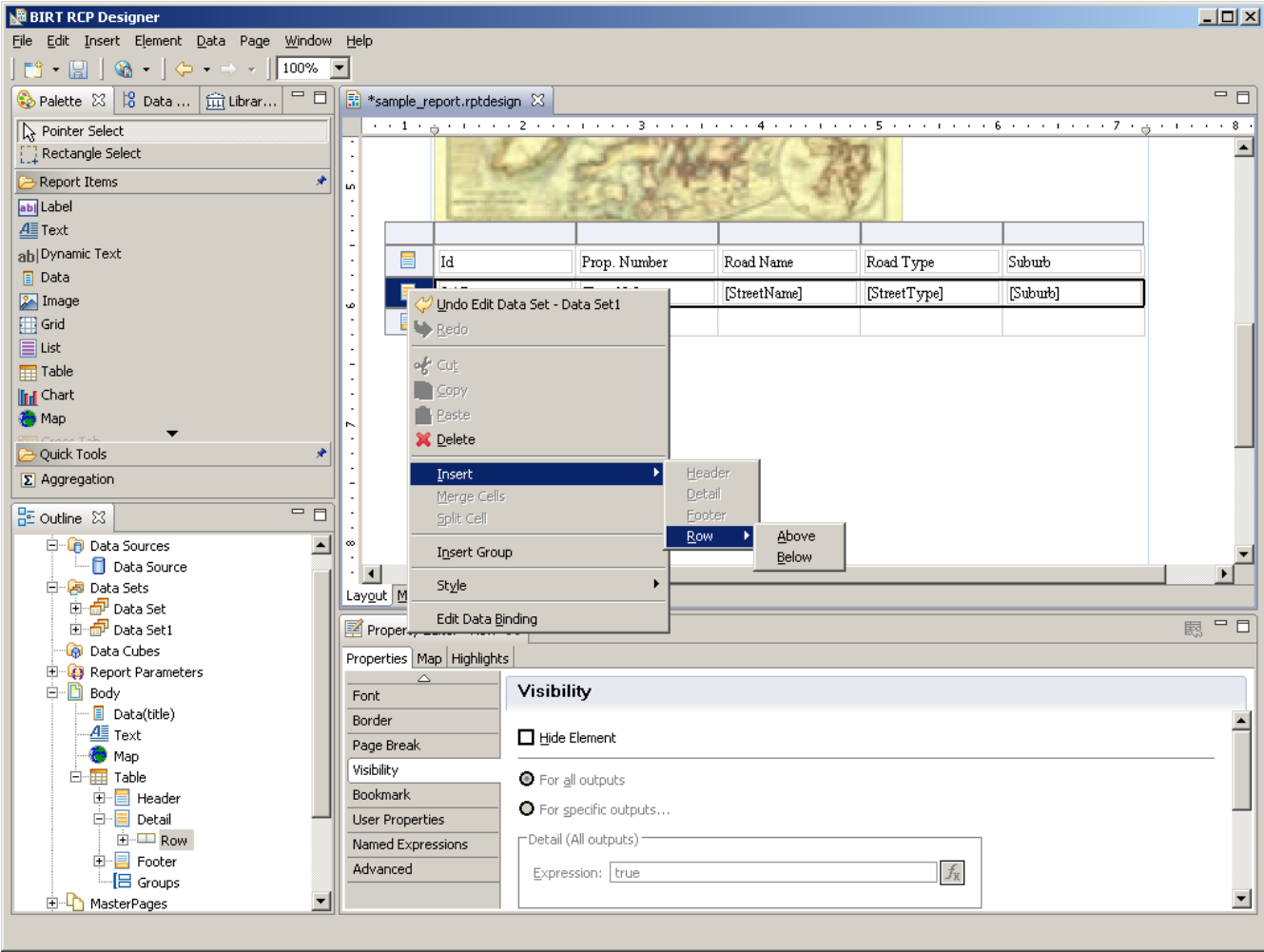


Here we've created a new data set from some sale data and this time setup a filter based on the `_id` column. It doesn't have to be the `_id` column but in this case the `_id` column is directly linked to the `_id` in the property details table we've already created, so we'll be joining them.

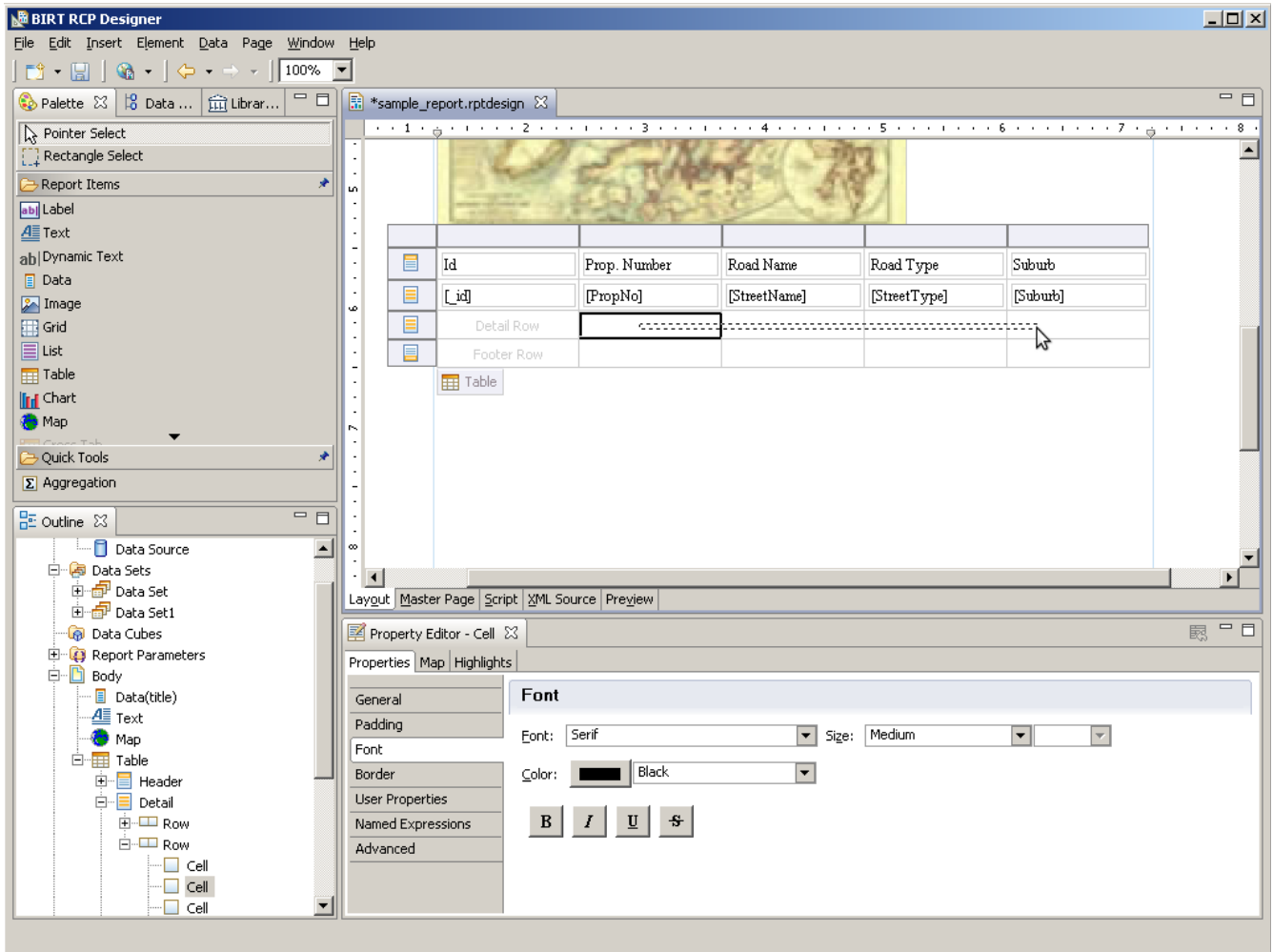
**i** It's important that a filter column be selected when creating the data set.

**x** You may receive a warning when you go to save the new data set, telling you that a value has not been set for the id parameter. This warning can be ignored because we'll be setting the value for the id parameter from its parents data set in the next few steps.

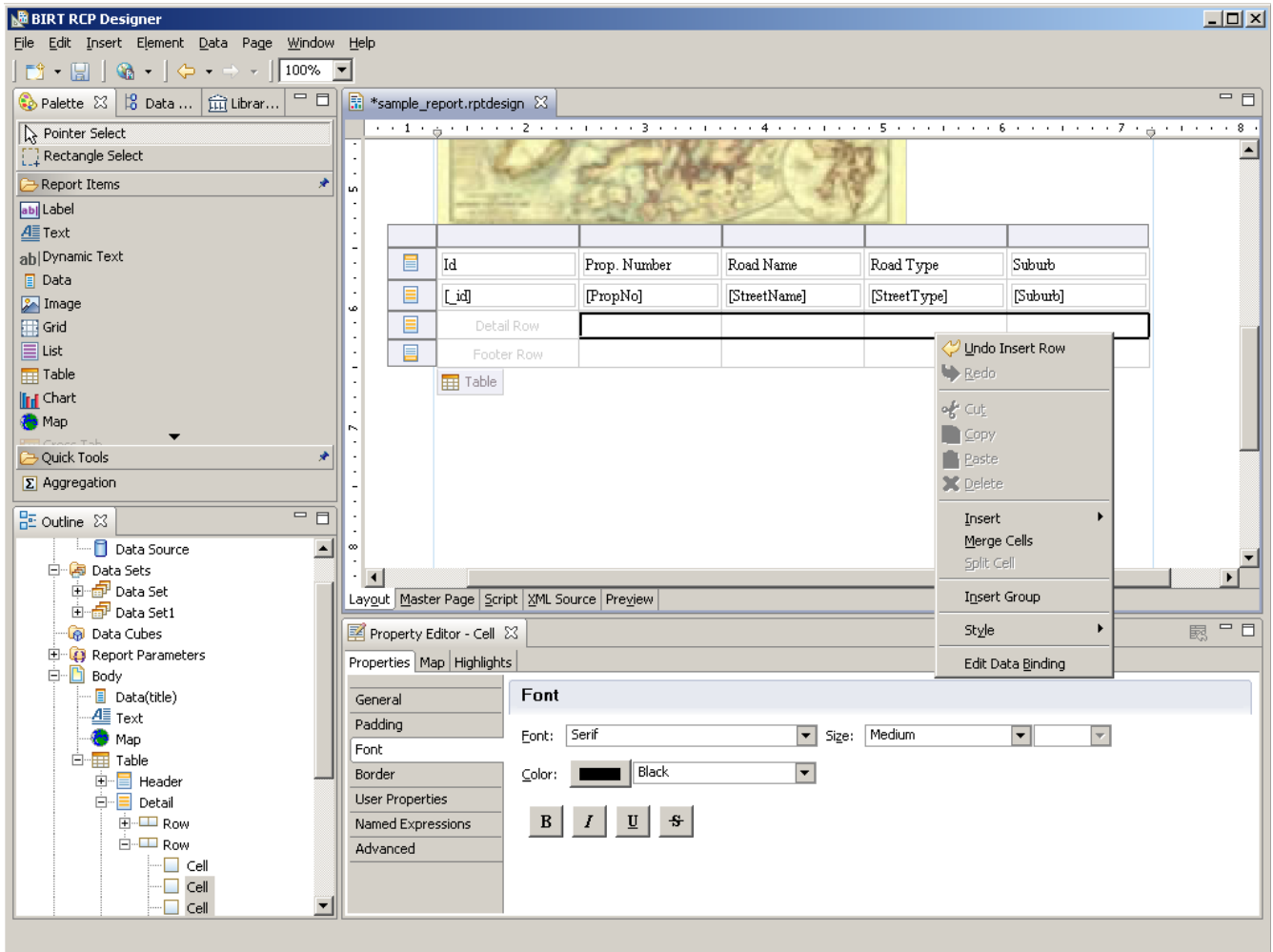
Now that we've created the new sub-data set we need to create some room in the existing table to add the new data. Select the table and right click on the row handle for the detail row and select `Insert|Row|Below` to add a new row below the current detail row, this way the sale details for each property will be listed below the address.



Then we merge the cells together by clicking and dragging across the visible cells in the new row

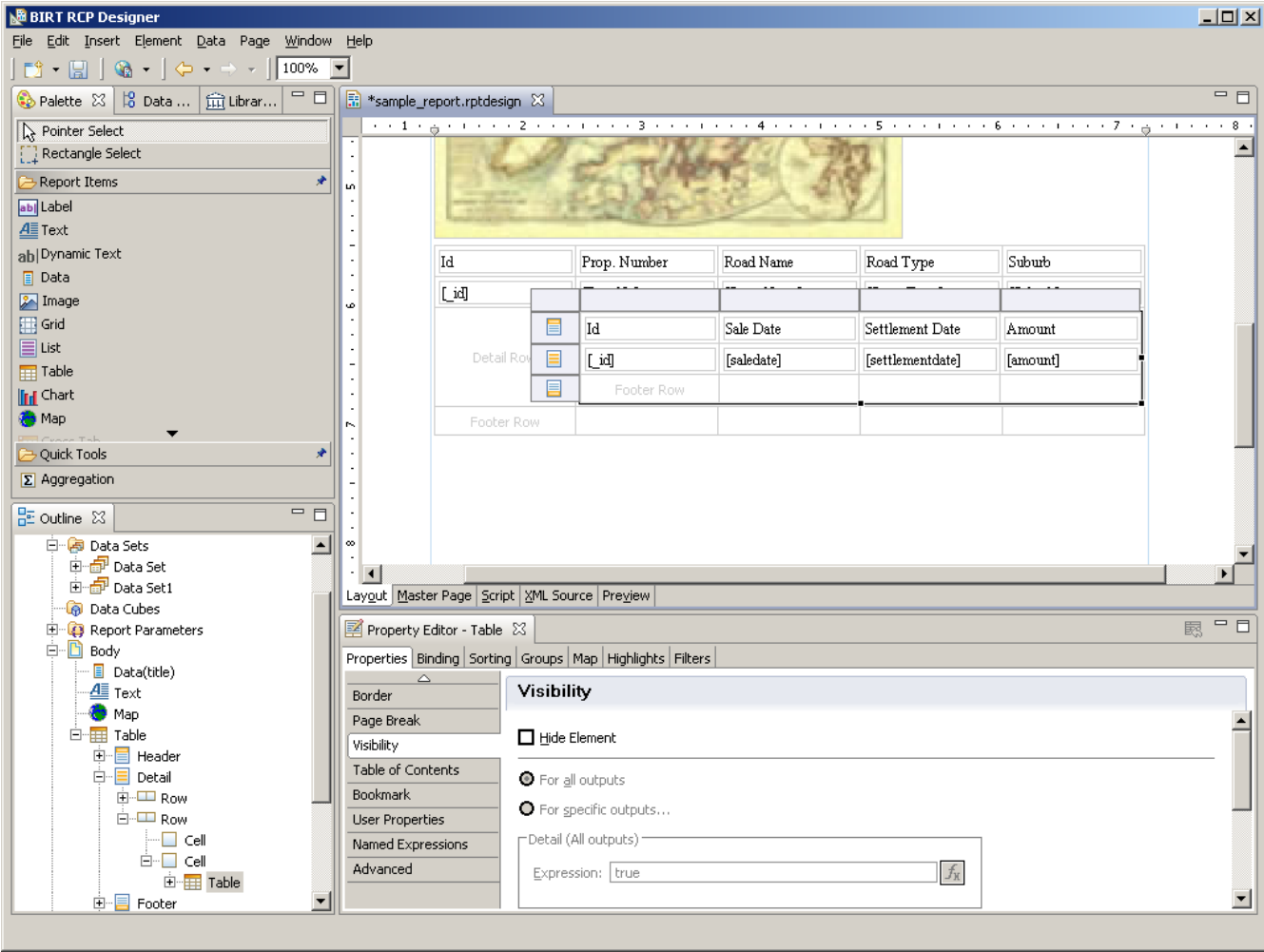


and then right clicking in the selected cells and selecting Merge Cells from the menu



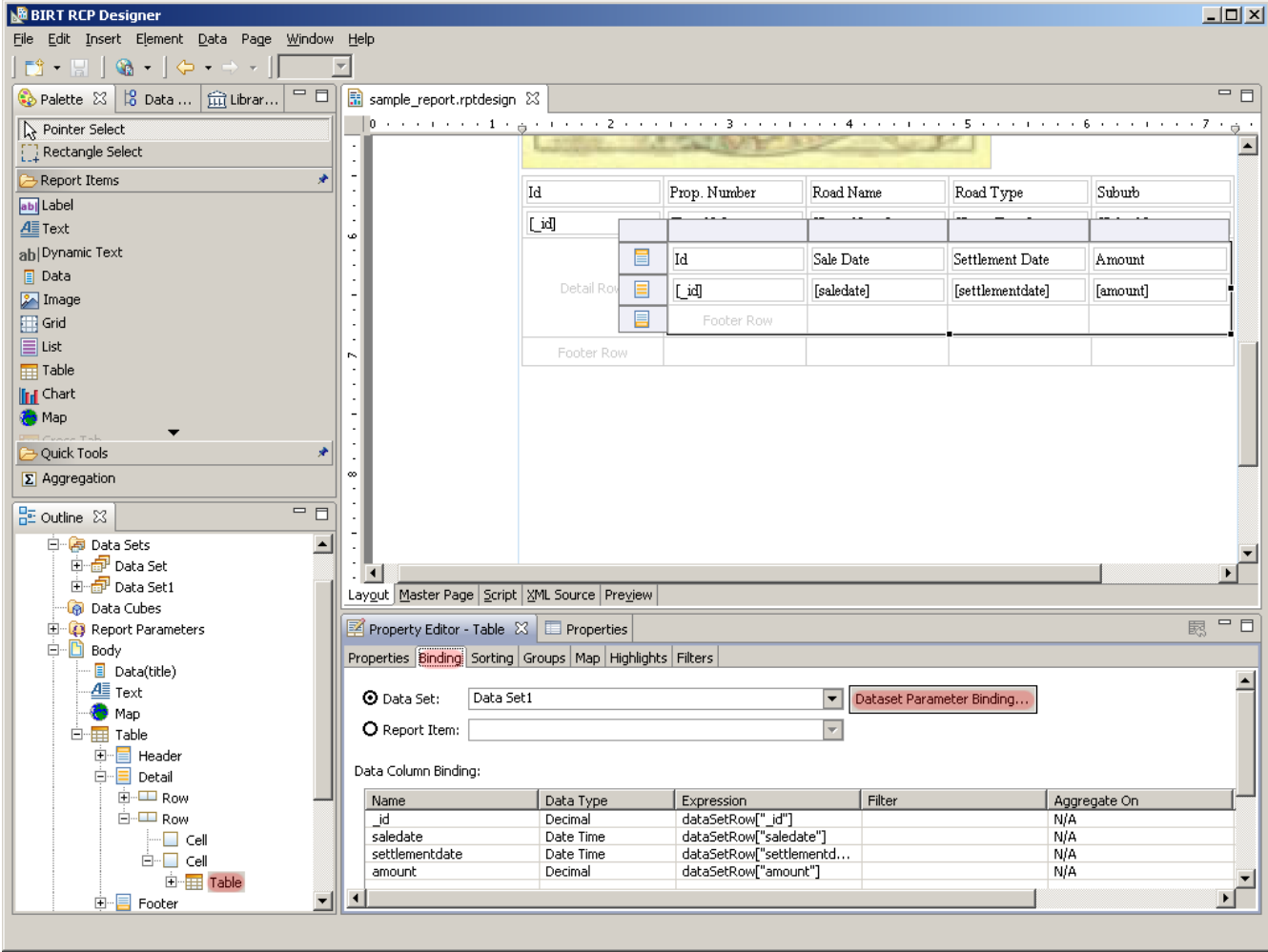
This way when we add the new data set to the existing table it'll fill up the width of the parent table. You don't have to do it this way, and you may find better methods of laying out your report pages, but this is to demonstrate linking data sets, not how to layout pages.

From here we can just drag and drop the new data set into the area we've just created in the existing table

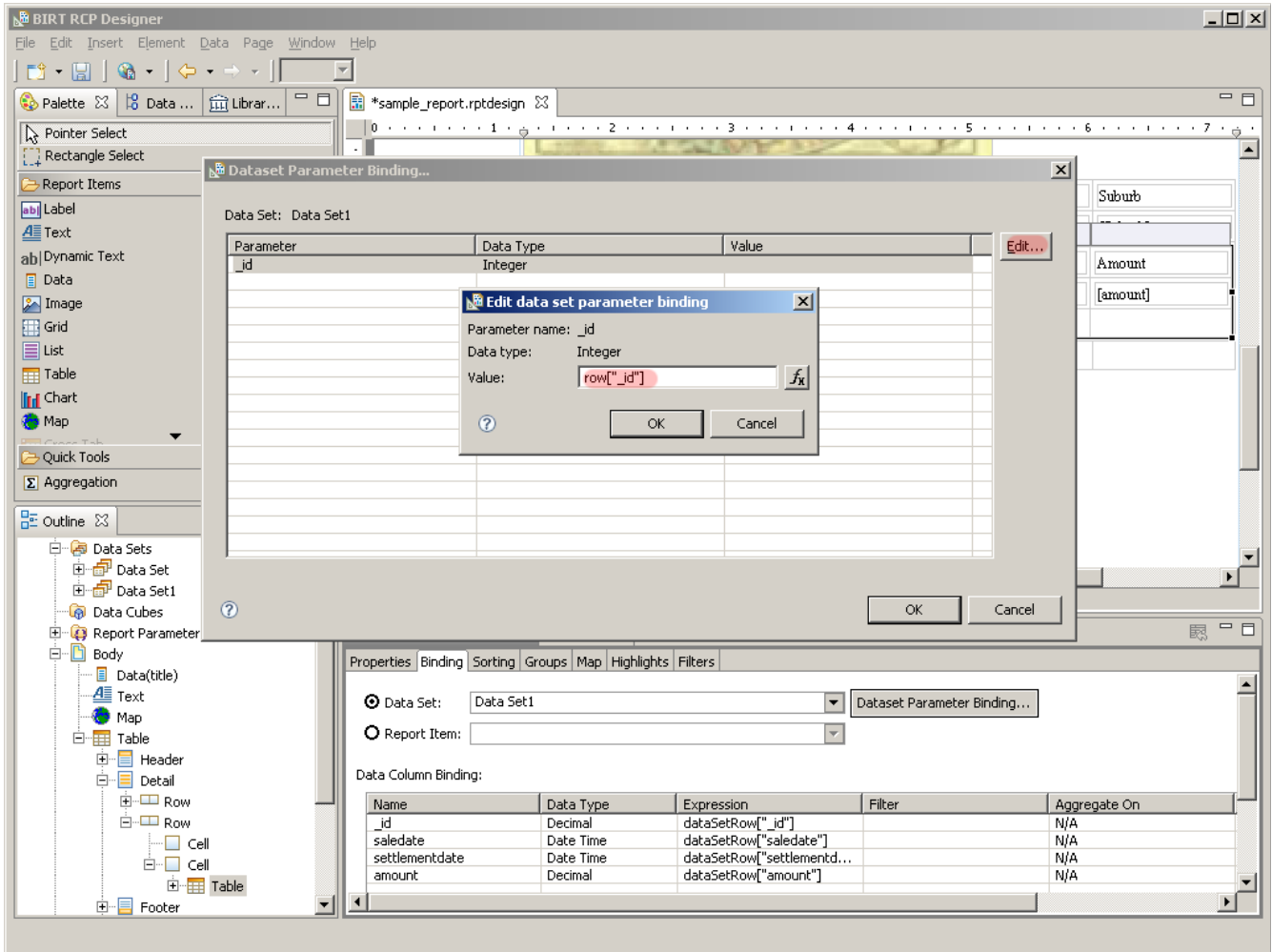


Remember to hide the `_id` column if you don't want it to appear in the final report

The important step here is to link the new table to the parent one, this is done in the Bindings for the tables data set



By editing the bindings for this tables "\_id" parameter and setting it to the value from the "\_id" field in the parent table, which is done by setting the value to row["\_id"], you'll link the sub-table to the parent table.



If it was a different column from the parent table that contained the value to use to filter the child table then you would use the name of that column, for example row["sale\_id"].

Finally, if we generate the report we'll have sales information for each property

Sample Report			
This report is just a sample to show how to get a report from the BIRT Report Designer to Weave			
Prop. Number	Road Name	Road Type	Suburb
87	LATROBE	STR	BULLEEN
<b>Sale Date</b>		<b>Settlement Date</b>	<b>Amount</b>
Nov 30, 2001		Nov 30, 2001	\$0
31	GRANT OLSON	AV	BULLEEN
<b>Sale Date</b>		<b>Settlement Date</b>	<b>Amount</b>
Sep 15, 2001		Jun 27, 1997	
Nov 16, 2002		Nov 20, 2001	\$367,150
		Jan 16, 2003	\$466,200

Nov 5, 2010 12:23 PM

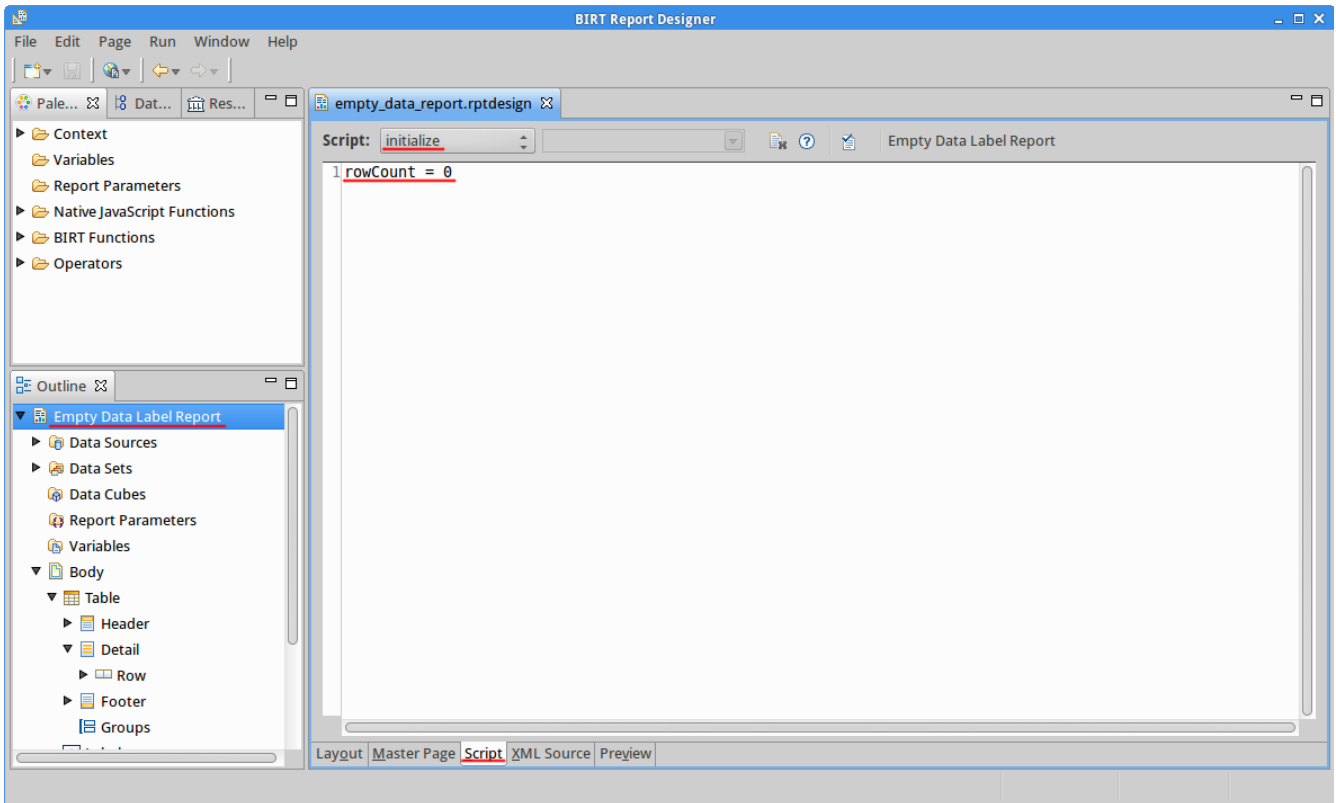
**i** The report design was formatted slightly to improve readability

### Reporting no data generated

To display to the user that there was no data generated for a table in a report follow these steps:

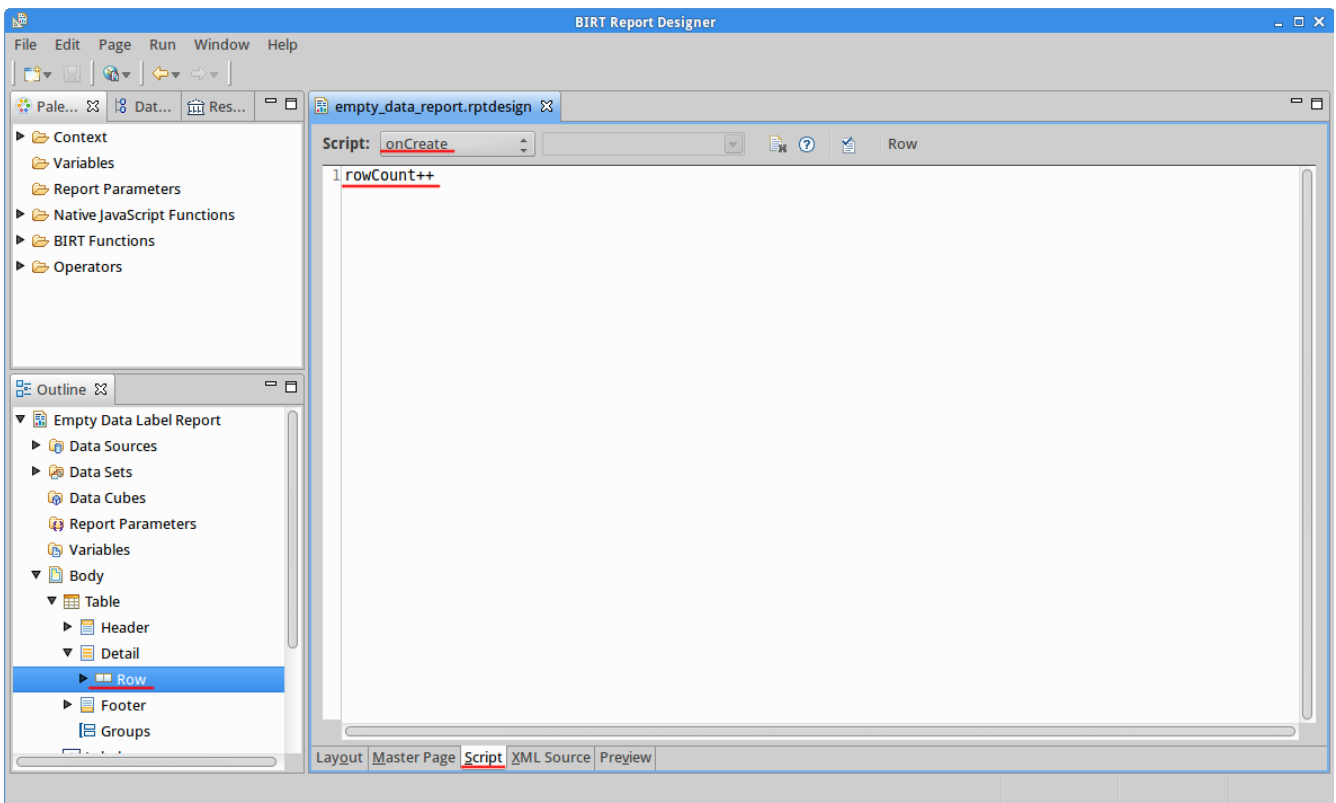
Step 1.

Create an initialize script to set a counter variable to 0.



### Step 2.

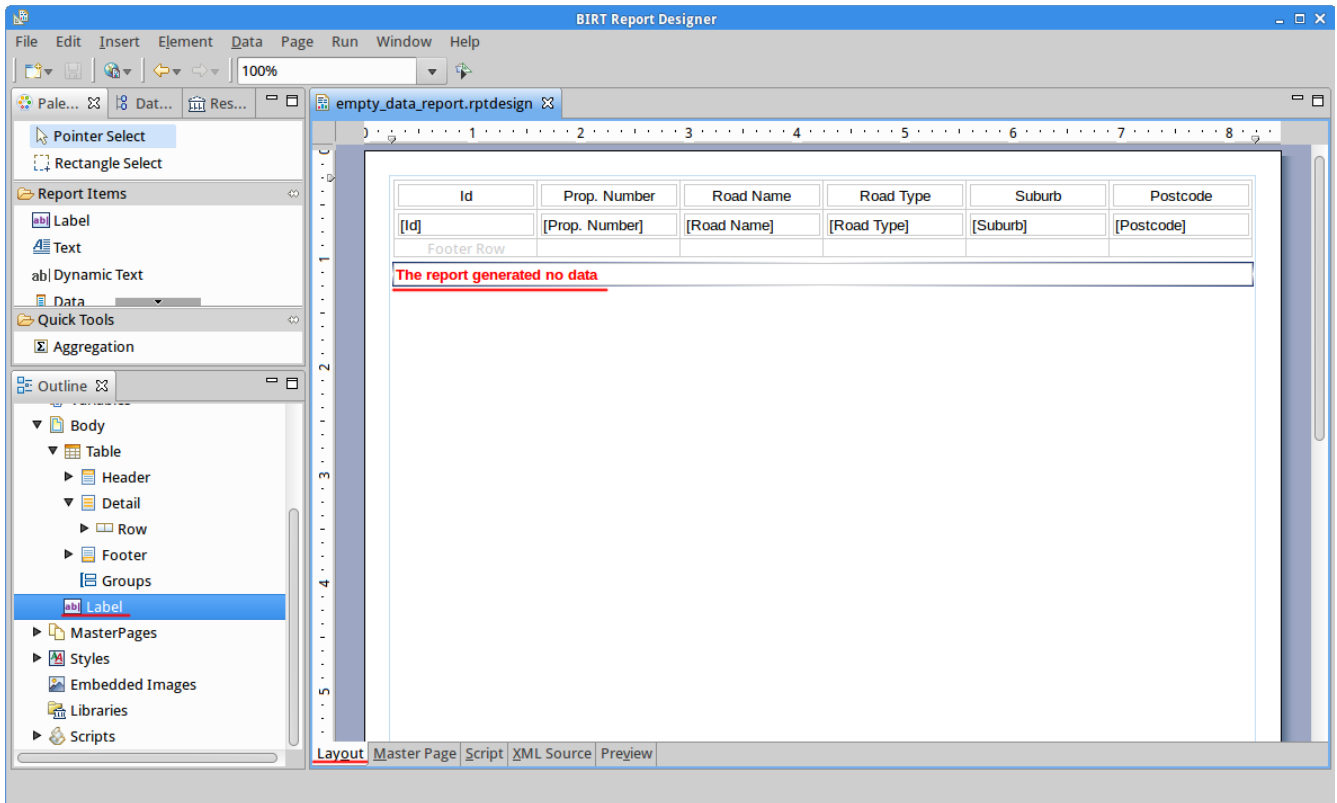
Create an onCreate script to increment the counter for each table row displayed.



### Step3.

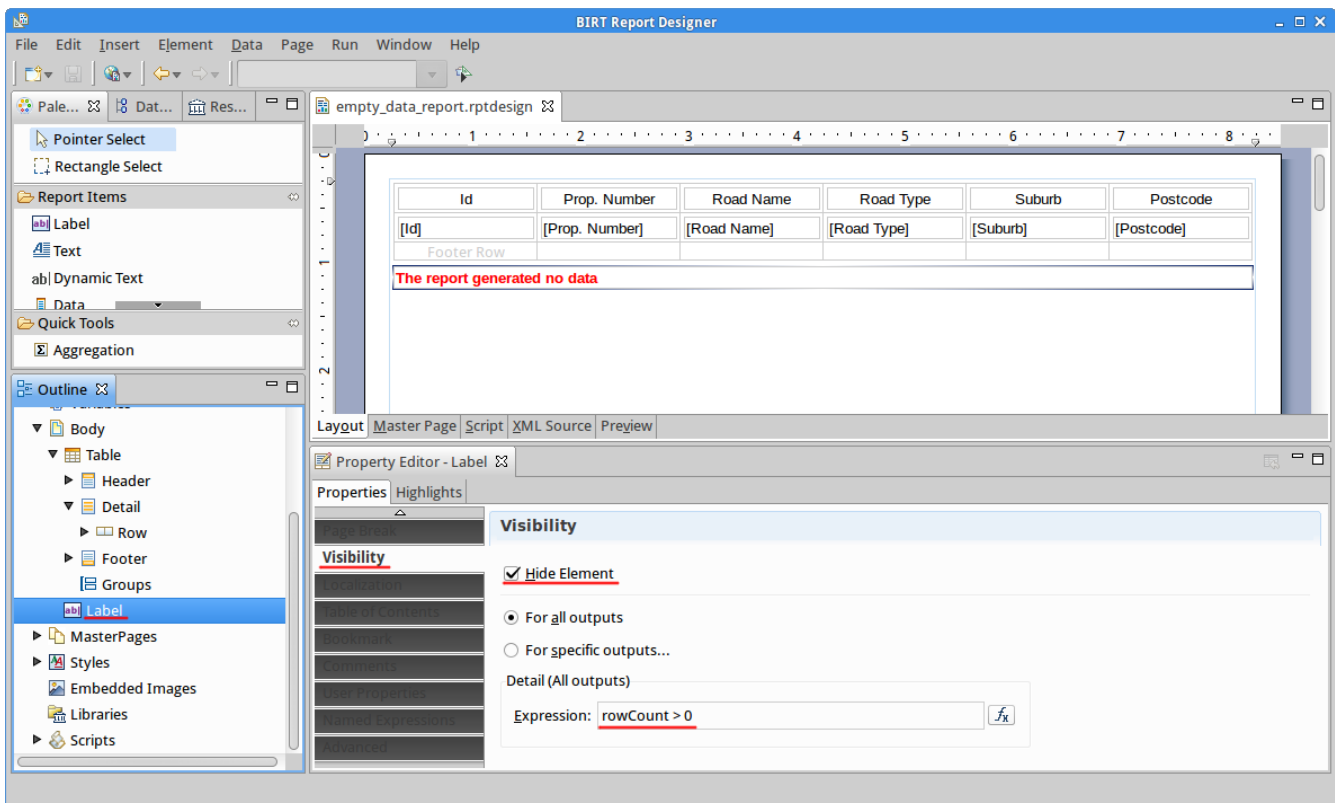
Create a Label with the text you want displayed when no data was generated.





#### Step 4.

Hide the label if the row counter is greater than 0.



## Application Integration

### Filtering Id's

Some of the application integration modules allow the translation of identifiers being sent between Weave and the third party application which can be done by setting up a "filter" and then directing the application integration module to use that filter when sending or receiving a list of identifiers (see the documentation for the application integration module for details on how to link the filter).

To define a new filter you need to add the `com.cohga.selection.filter` namespace to your config file, e.g.

### Adding the filter namespace

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:com.cohga.server.config#1.0" xmlns:filter="urn:com.
cohga.selection.filter#1.0">
  ...
</config>
```

Then you can create a filter that will convert id's from one value to another. e.g.

### Example database filter

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:com.cohga.server.config#1.0" xmlns:filter="urn:com.
cohga.selection.filter#1.0">
  <filter:db id="property.filter">
    <datasource>main</datasource>
    <table>GEMS_LINK</table>
    <keycolumn>PID</keycolumn> <!-- column used in spatial database -->
  >
    <idcolumn>PRUPI</idcolumn> <!-- column required for third party
application -->
  </filter:db>
</config>
```

When configured to use the above filter Weave will translate PID values to PRUPI when sending to the third party application and will do the reverse, translate PRUPI values to PID, when coming from the third party application into Weave.

You can further extend the filter by adding additional `from` and `where` clauses to include additional tables in the translation.

In addition to a `db` (database) filter you can also create a `format` filter or a `chain` filter. A `format` filter can alter the formatting of the id's, for example, if they need to be padded with spaces, and the `chain` filter can combine other `db` and `format` filters, since you can only specify a single filter and you may want to translate and format the id's.

### Format filter

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:com.cohga.server.config#1.0" xmlns:filter="urn:com.
cohga.selection.filter#1.0">
  <filter:format id="property.format">
    <idtrimin>true</idtrimin>
    <keylpadout>8</keylpadout>
  </filter:format>
</config>
```

The `format` filter has 12 values that can be set, to determine if the id/key is left or right padded, or if it's trimmed of extra spaces. (id or key) x (lpad, rpad or trim) x (in or out) = 12.

For left or right padding the property should be set to the number of characters the final value should be padded out to, for trim the property should be set to `true` or `false`.

e.g. `<idlpadout>20</idlpadout>`, `<keyrpadin>5</keyrpadin>`, `<keytrimin>true</keytrimin>`

### Chain filter

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:com.cohga.server.config#1.0" xmlns:filter="urn:com.
cohga.selection.filter#1.0">
  <filter:chain id="property.chain">
    <filter>property.filter</filter>
    <filter>property.format</filter>
  </filter:chain>
</config>
```

## Google Earth

**NOTE:** Google has decided to retire the Google Earth API. Per their deprecation policy, the API will be turned off on December 12, 2015.

<http://googlegeodevelopers.blogspot.com.au/2014/12/announcing-deprecation-of-google-earth.html>

Cohga is not a registered partner of Google nor is Google associated with Cohga or Weave.

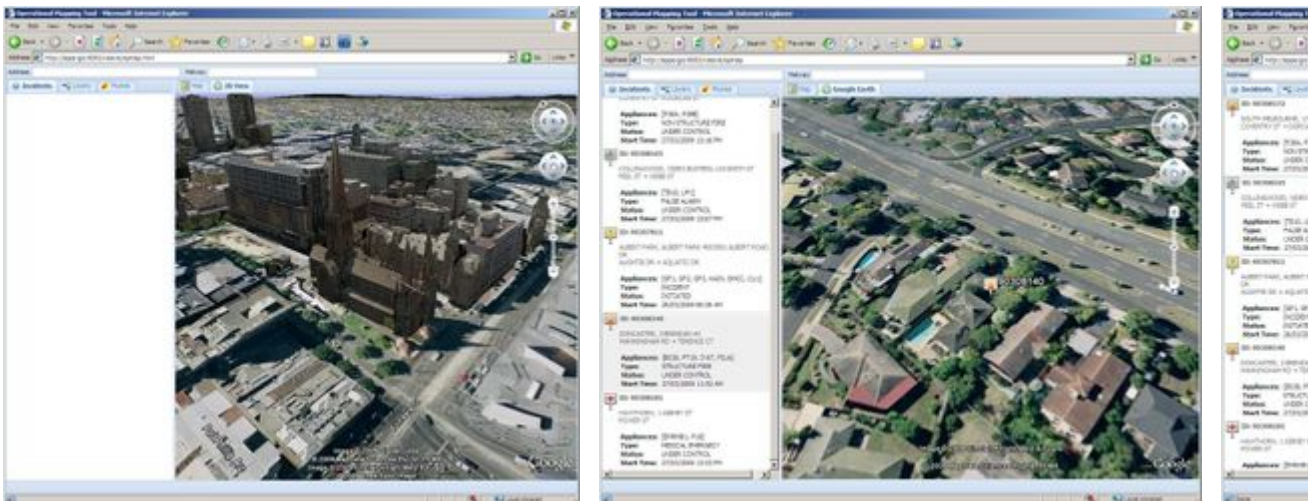
The Cohga developers used the public Google Earth API to create the interface between Weave and Google Earth. Cohga accepts no responsibility regarding up time or stability of the Google Earth plugin or services relating to it.

The integration between Weave and Google Earth exists via a plugin that must be installed on each client machine. This plugin can be installed via the following URL which requires that the user has the appropriate permissions to install applications on the users machine.

<http://www.google.com/intl/en/earth/explore/products/plugin.html>

On existing sites it seems to work best with a true 3D model of the area. If just looking at an image on the ground it does not have the same impact. The down side of 3D models is that they generally consume large amounts of memory and consume considerable bandwidth.

## Weave Google Earth Screenshots

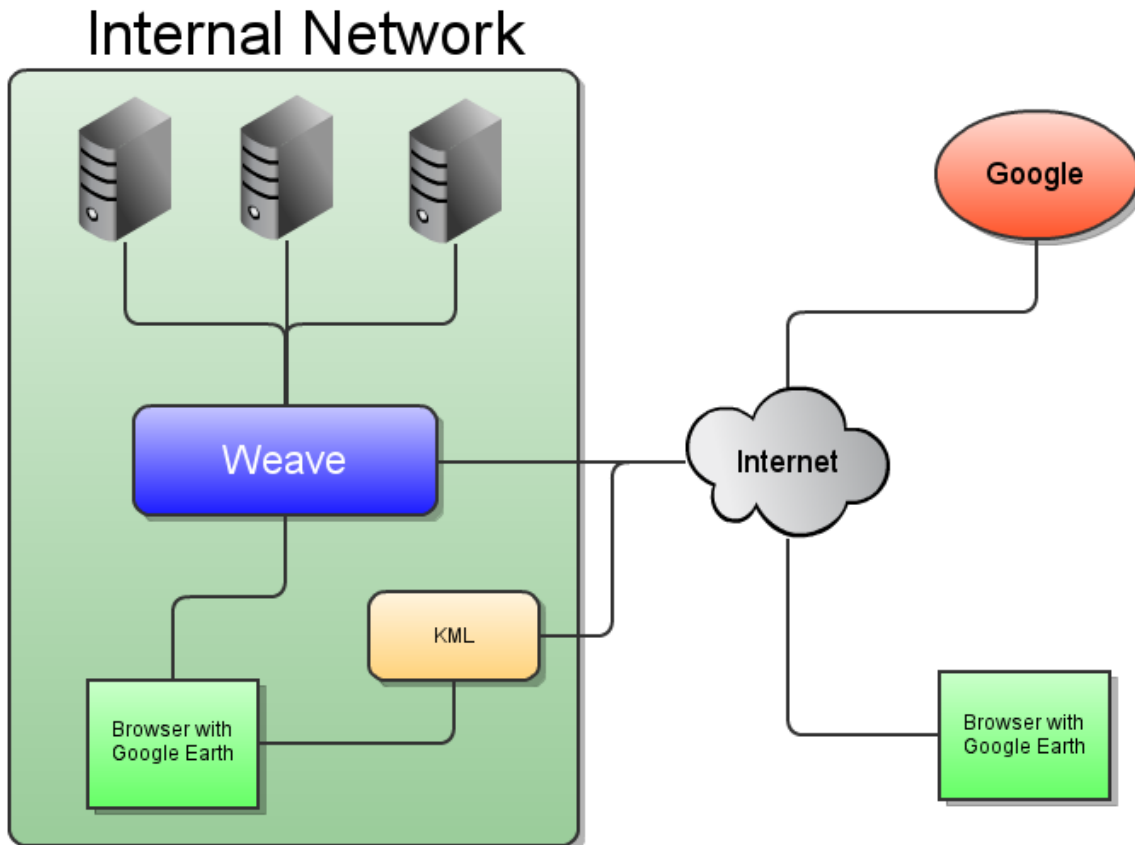


## Architecture

The Google Earth Integration has been implemented using the JavaScript API described in the Google documentation. It is designed to reside either in a tab or inside a floating window within the Weave client interface. The Google Earth plugin communicates with the Google servers for the main datasets and with other servers for sourcing KML and images.

There are some artifact issues in some versions of Internet Explorer (IE) when the plugin resides in a floating window.

The plugin and the data that is exposed to Google Earth needs to be considered. Since the Google Earth plugin renders all content on the client side, it needs to have access to the data sources provided. e.g. any KML dataset needs to be externally accessible for Google Earth to consume it. Depending if the KML source data is intended for both internal and external browsers.



### Advantages

- Licensing is still unknown at this stage. Several attempts by Weave sites to contact Google about licensing have not been successful. You can run the Google Earth plugin on external sites however internal use requires that the organisation purchases an Enterprise License.
- Allows visualising 3D data. However Weave does not support exporting 3D data.
- Adds another dimension to the interface.

### Disadvantage

- Currently Weave only synchronises the map extent with Google Earth. Move the map to a new location and the the View in Google Earth will update.
- The current map scale is translated into a height above the WGS84 datum in Google Earth. This value is only an approximation and the height in Google Earth may not be exactly the same as the scale in the Map View.
- Currently no way of adding the current selection to the Google Earth view.
- Currently no way for the user to add custom KML URLs to the Google Earth view. The administrator defines what the user sees.
- Browser may suffer memory issues as the Google Earth plugin is memory intensive.
- Only Windows and Macintosh systems are supported.
- Large KML files tend to slow down the browser.
- Data under the ground will not be visible.
- The users computer needs to have a recent graphics card and adequate memory. (See the Google Earth system requirements for more information.)

### Google StreetView

```
<?xml version="1.0" encoding="UTF-8"?>

<config          xmlns="urn:com.cohga.server.config#1.0"
```

```

xmlns:client="urn:com.cohga.html.client#1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<client:config id="streetview">
  <!-- Must include the google script with your API key
-->
  <script>
    <![CDATA[http://www.google.com/jsapi?
key=<GoogleApiKeyMustBeCopiedHere>]]>
  </script>
  <debug>true</debug>
  <title>Weave StreetView Client</title>
  <description>A Google StreetView client for Weave<
/description>
  <perspective>
    <text>Main</text>
    <view id="com.cohga.html.client.map.mapView">
      <location>center</location>
      <label>Map</label>

      <toolbar>
        <item action="com.cohga.html.
client.map.previousExtentAction" />
        <item action="com.cohga.html.
client.map.nextExtentAction" />
        <item action="com.cohga.html.
client.map.initialExtentAction" />
        <item action="com.cohga.html.
client.map.zoominAction" isDefault="true" />
        <item action="com.cohga.html.
client.map.zoomoutAction" />
        <item action="com.cohga.html.
client.map.panAction" />
        <item action="com.cohga.html.
client.map.refreshMapAction" />
        <item>-></item>
        <item component="com.cohga.
html.client.components.scaleSelector" />
      </toolbar>

      <statusbar>
        <item component="com.cohga.
html.client.components.loadingComponent" />
      </statusbar>

      <control id='com.cohga.client.mapctrl.
mousePosition' />
      <control id='com.cohga.client.mapctrl.
keyboard' />
      <control id='com.cohga.client.mapctrl.
panZoomBar' />

      <mapEngine id="mapengine.vector">

```

```

        <options>
<transitionEffect>force</transitionEffect>
        <ratio>1.2</ratio>
    </options>
</mapEngine>

<toc ref="toc.vector" />

<crs>EPSG:20255</crs>
<scales>
    <scale>100</scale>
    <scale>250</scale>
    <scale>500</scale>
    <scale>1000</scale>
    <scale>2000</scale>
    <scale>3000</scale>
    <scale>4000</scale>
    <scale>8000</scale>
    <scale>16000</scale>
    <scale>32000</scale>
    <scale>64000</scale>
    <scale>125000</scale>
    <scale>250000</scale>
</scales>
<extents>
    <initial crs="EPSG:20255"
minx="326022" miny="5810769" maxx="352871" maxy="5828285"/>
    <full crs="EPSG:20255" minx="
327098" miny="5811358" maxx="351971" maxy="5827675"/>
    <limit crs="EPSG:20255" minx="
320000" miny="5810000" maxx="360000" maxy="5840000"/>
</extents>
</view>

    <!-- Now add the street view panel -->
    <view id="weave.streetview">
        <location>center</location>
        <updateMap>false</updateMap> <!--
turn off updating of main map when street view is changed -->
    </view>
</perspective>
</client:config>

</config>

```

**Updated functionality available at version 2.10.24 of the `com.cohga.client.weave.google` bundle.**

You can specify additional options to alter the display of the street view panel by adding an options section and adding various options as described at [Google Street View Options](#).

```

<view id="weave.streetview">
    <options>
        <addressControl>false</addressControl>

```

```

        <imageDateControl>false</imageDateControl>
        <zoomControlOptions>
            <position>LEFT_BOTTOM</position>
        </zoomControlOptions>
        <panControlOptions>
            <position>LEFT_BOTTOM</position>
        </panControlOptions>
    </options>
</view>

```

Additionally, you can specify that an overview map is displayed, by setting `showOverview` to `true`, and its size and location. The size is set with `overviewWidth` and `overviewHeight`, the horizontal location with `overviewLeft` or `overviewRight` and the vertical location with `overviewTop` or `overviewBottom`. All size and location values are in pixels and the left, right, top and bottom values specify how far from that side the overview should be placed.

Note that `showOverview`, `overviewWidth` and `overviewHeight` were available prior to version 2.10.24.

```

<view id="weave.streetview">
    <showOverview>true</showOverview>
    <overviewRight>10</overviewRight>
    <overviewTop>10</overviewTop>
    <overviewWidth>300</overviewWidth>
    <overviewHeight>300</overviewHeight>
    <options>
        <addressControl>false</addressControl>
        <imageDateControl>false</imageDateControl>
        <zoomControlOptions>
            <position>LEFT_BOTTOM</position>
        </zoomControlOptions>
        <panControlOptions>
            <position>LEFT_BOTTOM</position>
        </panControlOptions>
    </options>
</view>

```

#### Motion Tracking


Some users have reported issues with motion tracking when using mobile devices, this can be disabled by setting the appropriate options for the street view panel as outlined on the [Google Street View Options](#) link above, e.g.

```

<view id="weave.streetview">
    <options>
        <motionTracking>false</motionTracking>
        <motionTrackingControl>false</motionTrackingControl>
    </options>
</view>

```


#### Nearmap

 As of August 2019 the Nearmap Panel is no longer available, the WMS service is still available and supported by Nearmap.

~~We're working with Nearmap to implement an alternative solution.~~

Nearmap have restored the functionality required for the Nearmap panel within Weave

Integration with Nearmap can be done in two ways, either as a WMS service or as a separate tab in the interface similar to Streetview and Google Earth.

 In order to use either of these two methods, you will need to be a licensed user of the Nearmap service.

#### Server config

To enable the integration with Nearmap and Weave you'll need to provide Weave with the username/password that the Nearmap service is registered with, this is to allow Weave to generate tickets for authentication.

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:nearmap="urn:
com.cohga.nearmap#1.0">

    <nearmap:config>
        <url><![CDATA[https://app.nearmap.com/api/0
/checkaccess?]]></url>
        <username>user@organisation.com</username>
        <password>ENCFASGAHFUASHGASFAS</password>
    </nearmap:config>

</config>
```

#### Separate Tab

The Web interface used in the Nearmap site can be embedded into a separate tab in Weave.

```
<view id="weave.nearmap">
    <location>center</location>
</view>
```

#### Updated client config

Due to changes in the Nearmap API you must include an API key in the panel config, also note the change from `weave.nearmap` to `weave.nearmap2`.

You will need to get an API key from the Nearmap Administration page using your login and place it in the `apiKey` option below.

Note this requires version 2.6.6 of Weave or version 1.3.15 of the `com.cohga.weave.nearmap` bundle.

```
<view id="weave.nearmap2">
    <location>center</location>
    <title>Nearmap</title>
    <tabTip>Provides up-to-date aerial images</tabTip>
    <apiKey>YOUR_API_KEY</apiKey>
</view>
```

#### WMS Service



A simple way that Nearmap can be added is using the WMS service provided by Nearmap. This is a standard WMS service that works the same as any other described in [WMS MapEngine](#).

First, add a reference to the WMS MapEngine

```
<wms:mapengine id="nearmap">
  <url>https://wms.nearmap.com/wms?apikey=[yourapikeyhere]</url>
  <tokens>
    <token name="apikey" value="[yourapikeyhere]" />
  </tokens>
  <layers>
    <layer>NearMap</layer>
    <layer>NearMap\Australia</layer>
    <layer>NearMap\Australia\Taarna</layer>
    <layer>NearMap\Australia\Taarna\2009-11-09</layer>
    <layer>NearMap\Australia\Taarna\2010-01-04</layer>
    <layer>NearMap\Australia\Taarna\2010-07-09</layer>
    <layer>NearMap\Australia\Taarna\2010-10-28</layer>
    <layer>NearMap\Australia\Taarna\2011-04-07</layer>
    <layer>NearMap\Australia\Taarna\2011-12-09</layer>
    <layer>NearMap\Australia\Taarna\2012-01-28</layer>
    <layer>NearMap\Australia\Taarna\2012-02-25</layer>
    <layer>NearMap\Australia\Taarna\2012-10-29</layer>
    <layer>NearMap\Australia\Taarna\2012-11-23</layer>
  </layers>
</wms:mapengine>
```

Add this service to the client that you want the WMS service to be visible in

```
<mapEngine id="mapengine.nearmap">
  <options>
    <transitionEffect>force</transitionEffect>
    <ratio>1.0</ratio>
    <singleTile>true</singleTile>
    <alpha>false</alpha>
  </options>
</mapEngine>
```

Generating the ToC entries for the service is a simple matter of executing the command at the OSGi console.

```
memd toc mapengine.nearmap
```

Running the above command will produce output similar to the following.

### Sample Nearmap ToC

To refine that further to your particular region to can grab a subset of layers. Below is an example of how you would only have the layers for Melbourne.



Some of the layers are visible=false, these are folders that need to be checked in order for the layers underneath them to be visible.

```

<toc:model id='toc.nearmap'>
  <mapengine>mapengine.nearmap</mapengine>
  <entry label="Nearmap">
    <entry layer="NearMap\Australia" label="Australia"
visible="false" checked="true"/>
    <entry layer="NearMap\Australia\Melbourne" label="
Melbourne" visible="false" checked="true"/>
    <entry layer="NearMap\Australia\Melbourne\20091012"
label="Melbourne 20091012"/>
    <entry layer="NearMap\Australia\Melbourne\20091106"
label="Melbourne 20091106"/>
    <entry layer="NearMap\Australia\Melbourne\20091113"
label="Melbourne 20091113"/>
    <entry layer="NearMap\Australia\Melbourne\20100107"
label="Melbourne 20100107"/>
    <entry layer="NearMap\Australia\Melbourne\20100220"
label="Melbourne 20100220"/>
    <entry layer="NearMap\Australia\Melbourne\20100416"
label="Melbourne 20100416"/>
    <entry layer="NearMap\Australia\Melbourne\20100711"
label="Melbourne 20100711"/>
    <entry layer="NearMap\Australia\Melbourne\20101001"
label="Melbourne 20101001"/>
    <entry layer="NearMap\Australia\Melbourne\20101020"
label="Melbourne 20101020"/>
    <entry layer="NearMap\Australia\Melbourne\20101120"
label="Melbourne 20101120"/>
    <entry layer="NearMap\Australia\Melbourne\20110406"
label="Melbourne 20110406"/>
    <entry layer="NearMap\Australia\Melbourne\20110628"
label="Melbourne 20110628"/>
    <entry layer="NearMap\Australia\Melbourne\20110820"
label="Melbourne 20110820"/>
    <entry layer="NearMap\Australia\Melbourne\20111018"
label="Melbourne 20111018"/>
    <entry layer="NearMap\Australia\Melbourne\20111117"
label="Melbourne 20111117"/>
    <entry layer="NearMap\Australia\Melbourne\20111205"
label="Melbourne 20111205"/>
    <entry layer="NearMap\Australia\Melbourne\20120312"
label="Melbourne 20120312"/>
    <entry layer="NearMap\Australia\Melbourne\20120412"
label="Melbourne 20120412"/>
    <entry layer="NearMap\Australia\Melbourne_North"
label="Melbourne North" visible="false" checked="true"/>
    <entry layer="
NearMap\Australia\Melbourne_North\20100621" label="Melbourne North
20100621"/>
    <entry layer="
NearMap\Australia\Melbourne_North\20101224" label="Melbourne North
20101224"/>

```

```

        <entry layer="
NearMap\Australia\Melbourne_North\20110120" label="Melbourne North
20110120"/>
        <entry layer="
NearMap\Australia\Melbourne_North\20110129" label="Melbourne North
20110129"/>
        <entry layer="
NearMap\Australia\Melbourne_North\20120117" label="Melbourne North
20120117"/>
        <entry layer="
NearMap\Australia\Melbourne_North\20120215" label="Melbourne North
20120215"/>
        <entry layer="NearMap\Australia\Melbourne_South"
label="Melbourne South" visible="false" checked="true"/>
        <entry layer="
NearMap\Australia\Melbourne_South\20100622" label="Melbourne South
20100622"/>
        <entry layer="
NearMap\Australia\Melbourne_South\20101003" label="Melbourne South
20101003"/>
        <entry layer="
NearMap\Australia\Melbourne_South\20101224" label="Melbourne South
20101224"/>
        <entry layer="
NearMap\Australia\Melbourne_South\20110120" label="Melbourne South
20110120"/>
        <entry layer="
NearMap\Australia\Melbourne_South\20110429" label="Melbourne South
20110429"/>
        <entry layer="
NearMap\Australia\Melbourne_South\20120116" label="Melbourne South
20120116"/>
        <entry layer="
NearMap\Australia\Melbourne_South\20120201" label="Melbourne South
20120201"/>
        <entry layer="
NearMap\Australia\Melbourne_South\20120320" label="Melbourne South
20120320"/>
    </entry>
</toc:model>

```

## Integrating with AssetMaster

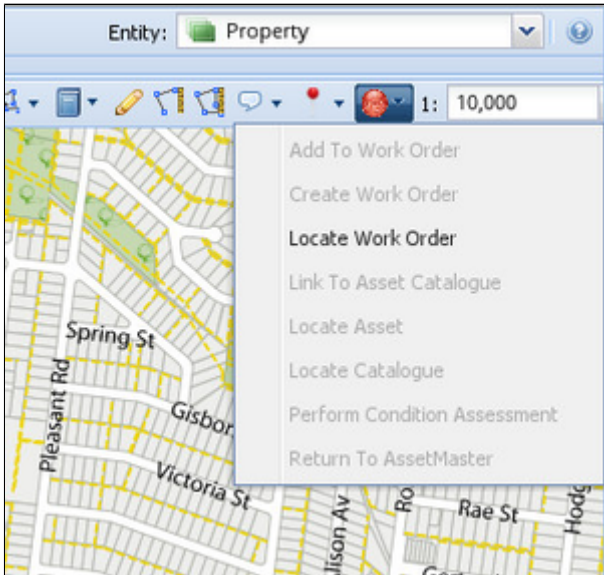
The current Weave/AssetMaster integration supports a two way link between Weave and AssetMaster. That is AssetMaster can send the current set of assets to Weave for display and Weave can send the current set of assets for display in AssetMaster. Currently Internet Explorer is the only browser supported for this type of integration.

The implementation of the Weave - AssetMaster Application Integration Module provides the following functions.

### Link from Weave – AssetMaster

- Locate Asset
- Locate Catalogue
- Link to Asset Catalogue
- Create Work Order
- Add to Work Order
- Locate Work Order
- Perform Condition Assessment

- Return to AssetMaster

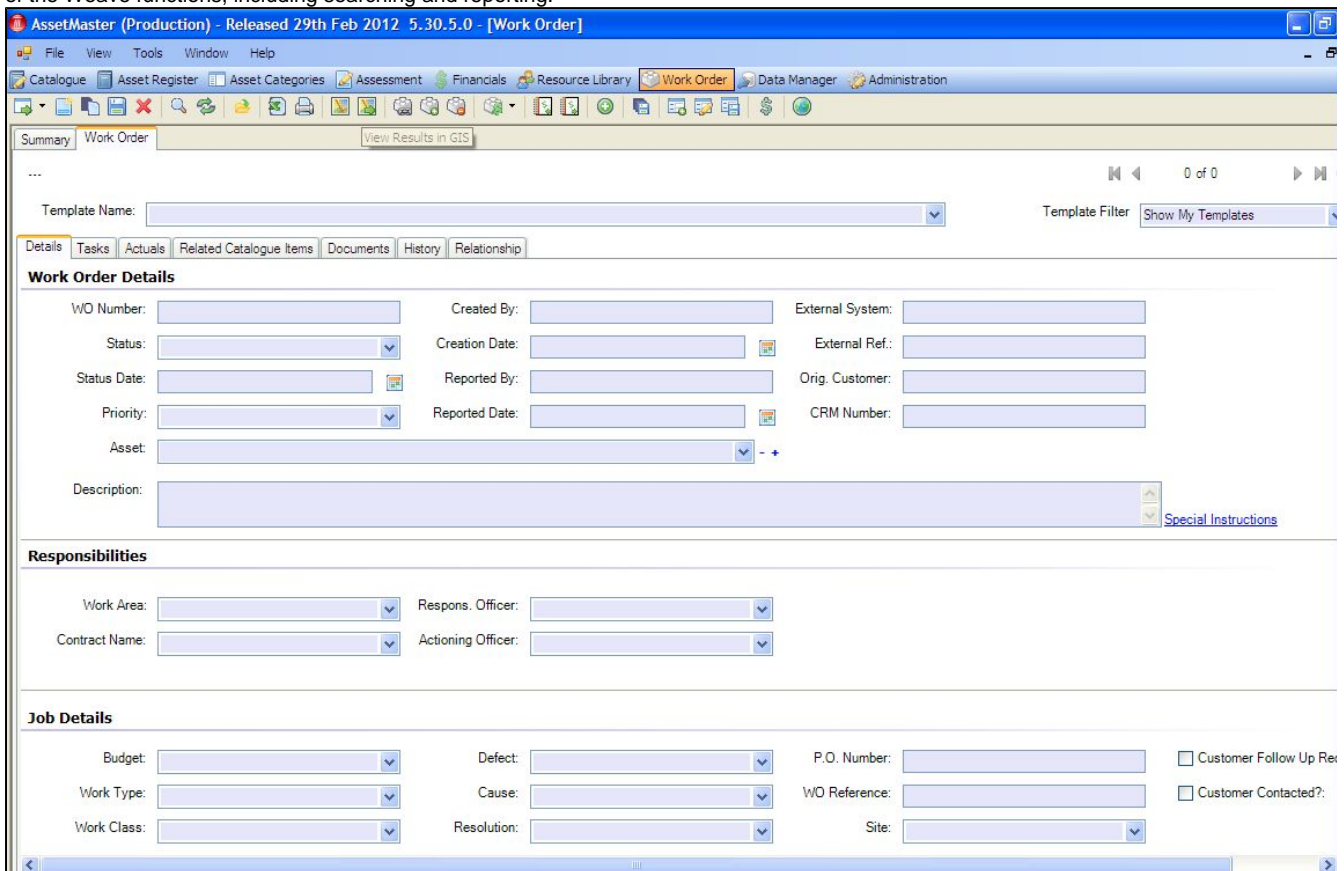


Link from AssetMaster – Weave

- Show GIS
- View Results in GIS

Upon clicking on the AssetMaster button option in Weave, the AssetMaster application will be brought forward with the selected procedure in AssetMaster and will be focused on the records corresponding to the features selected in Weave. From this point, the complete suite of AssetMaster operations is available to the user, including work order creation, reporting, etc. Please refer to AssetMaster's documentation for a full list of features available to the user.

Sending information from AssetMaster to Weave is undertaken using AssetMaster's GIS button. Once a feature(s) is selected in the AssetMaster interface, the user has the ability to send the features from AssetMaster to Weave. From this point, the user has access to all of the Weave functions, including searching and reporting.



## Setup


The setup of the Weave - AssetMaster Application Integration Module comprises two stages:

#### 1. Weave Server Setup

Add the following to the client.xml file to provide a menu with the tools available.

```
<item action="com.cohga.assetmaster.menu">
```

*The AssetMaster Server side code.*

 The namespace used for AssetMaster is

```
xmlns:assetmaster="urn:com.cohga.assetmaster#1.0"
```

```
<assetmaster:config id="default">
  <assetmaster:dir>C:\Program
Files\InfoMaster\AssetMaster_PROD\</assetmaster:dir>
  <assetmaster:exe>applauncher.exe</assetmaster:exe>
  <assetmaster:appname>iAssetMaster</assetmaster:appname>
  <assetmaster:port>9000</assetmaster:port>
  <assetmaster:server>localhost</assetmaster:server>
  <assetmaster:debug>>true</assetmaster:debug>
  <assetmaster:seperator>_</assetmaster:seperator>
  <assetmaster:entity>lyr_assets</assetmaster:entity>
</assetmaster:config>
```

where

name	value
dir	The location of the AssetMaster installation
exe	The executable used to send ids to
port	The port the application is listening on
server	The location of the server
debug	whether to enable debugging on the AssetMaster side
seperator	When sending ids from Weave to AssetMaster the separator used to distinguish the ids
entity	The entity that will be used for asset information.

#### 2. Client PC Setup (setting up the user's computer)

The Weave AssetMaster DLL must be installed on each client PC as well as the Weave Link component

```
regsvr32 Weave.AssetMaster.dll
```

*AssetMaster Setup and Configuration*

The AssetMaster system requires the installation of the AssetMaster Client. Please refer to the Installation Guide documentation supplied with this product for the relevant information.

## Integration with Authority

### Components

Note: These components are for using Authority with [WeaveLink](#), not with [Weave Hub](#), and likely out of date anyway and should instead be installed via the Interop installer at <http://downloads.cohga.com/weave/>

For additional information about using the newer Weave Hub integration please see this page [Weave Hub - Third-party Application Integration](#).

### Authority Specific Components

[WeaveAuthorityLink100.dll](#) - This component must be installed and registered (`regsvr32 /s WeaveAuthorityLink100.dll`) on each client PC. It allows Weave to launch and communicate with Authority.

[com.cohga.weave.authority\\_1.0.0.jar](#) - This bundle must be copied to the `weave\platform\plugins` directory and the server restarted.

[Setup\\_customer.bat](#) - This is the batch file that Authority will call to initiate communications with Weave. This must be installed according to the requirements of Authority (along with `Setup_customer.vbs`)

[Setup\\_customer.vbs](#) - This is the script that the `Setup_customer.bat` file will actually call. It must be edited before being deployed to change the Windows Title and Startup URL.

### Other Components

[com.cohga.client.weave\\_2.3.0.jar](#) - Minimum version of the Weave client provider bundle (if you have something later than 2.3.0 then keep it)

[com.cohga.client.weave.main\\_2.3.0.jar](#) - Minimum version of the Weave client bundle (if you have something later than 2.3.0 then keep it)

[com.cohga.client.interop\\_2.1.0.jar](#) - Minimum version of the Weave interop bundle (if you have something later than 2.1.0 then keep it)

[com.cohga.server.selection.filter.impl\\_1.0.0.jar](#) - Minimum version of the Weave filter implementation bundle (if you have something later than 1.0.0 then keep it)

### Configuration

#### Server

Minimum required server configuration, showing the Authority namespace, setting the data source that's used to communicate between Authority and Weave and specifying which Weave entities correspond to which modules in Authority (at least 'DD' and 'PR' must be set).

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:authority="urn:com.cohga.weave.authority#1.0">

    <authority:config>
        <datasource>datasource.authority</datasource>
        <module id="DD" entity="parcels"/>
        <module id="PR" entity="parcels"/>
    </authority:config>
</config>
```

**i** Note that by default Weave will use the `key` value defined in the spatial mapper for the entity to determine what values to send to Authority. You can change this by setting a `key` directly in the module definition to point to a different column.

Additionally you can also specify a `filter` that will be applied to the id's before being sent to Authority.

Full configuration showing all options (set using their default values)

```

<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:authority="urn:
com.cohga.weave.authority#1.0">

    <authority:config>
        <datasource>datasource.authority</datasource>
        <module id="DD" entity="parcels"/>
        <module id="PR" entity="parcels"/>

        <!-- Other optional settings with their default
values -->

        <!-- The table to use to communicate between Weave
and Authority -->
        <table>aualmapl</table>

        <!-- The name of the batch file to launch when Weave
tries to start Authority, it must contain 'ulaunch' but now you can
specify a path -->
        <ulaunch>ulaunch</ulaunch>

        <!-- The SQL to execute to obtain a list of all of
the available neighbour notification letters -->
        <alllettersssql>select ext_typ, ext_dsc from audmextp
where mdu_ref = 'DD' order by ext_dsc</alllettersssql>

        <!-- The SQL to execute to obtains a list of the
available neighbour notification letters for a specific type -->
        <lettersssql>select ext_typ, ext_dsc from audmextp
where mdu_ref = 'DD' and (for_doc is null or for_doc = ?) order by
ext_dsc</lettersssql>

        <!-- The name of the file to create for a bulk update
-->
        <bulkupdatefilename><!-- userid.txt or useridHHMMSS.
txt depending upon the value of 'simplebulkupdatefilename' --><
/bulkupdatefilename>

        <!-- This value is only used if bulkupdatefilename is
not set -->
        <!-- If true then the auto generated
bulkupdatefilename will be formatted as <userid>.txt -->
        <!-- If false then the auto generated
bulkupdatefilename will be formatted as <userid><currenttime>.txt -->
        <simplebulkupdatefilename>>false<
/simplebulkupdatefilename>

        <!-- The directory to save the bulk update file to -->
        <bulkupdatedirectory><!-- determined by
'bulkupdatesql' if not set --></bulkupdatedirectory>

        <!-- The SQL to execute to obtain the name of the
bulk update output directory -->

```

```

        <bulkupdatesql>SELECT
chr_no1||chr_no2||chr_no3||chr_no4 FROM aualparm WHERE mdu_ref='PR'<
/bulkupdatesql>

        <!-- What version of the Authority integration uis
being used -->
        <!-- Currently only effects the generation of the
client id -->
        <version>2</version>

    </authority:config>
</config>

```

#### Client

Tools that can be added to the client toolbar/statusbar to enable communication between Weave and Authority

```

<item action="com.cohga.authority.Send"/>
<item action="com.cohga.authority.BulkUpdate"/>
<item action="com.cohga.authority.NeighbourNotification"/>

```

Same tools, but presented as a menu

```

<item action="com.cohga.client.actions.menuAction">
    <text></text>
    <iconCls>icon-authority</iconCls>
    <item action="com.cohga.authority.Send">
        <iconCls>icon-authority_send</iconCls>
        <text>Send to Authority</text>
    </item>
    <item action="com.cohga.authority.BulkUpdate">
        <iconCls>icon-authority_bulkupdate</iconCls>
        <text>Bulk Update</text>
    </item>
    <item action="com.cohga.authority.NeighbourNotification">
        <iconCls>icon-authority_notification</iconCls>
        <text>Neighbour Notification</text>
    </item>
</item>

```

#### Authority Documentation

Below are links to the documentation provided by Civica which was used as the basis for implementing the Weave/Authority link.

[Overview](#)

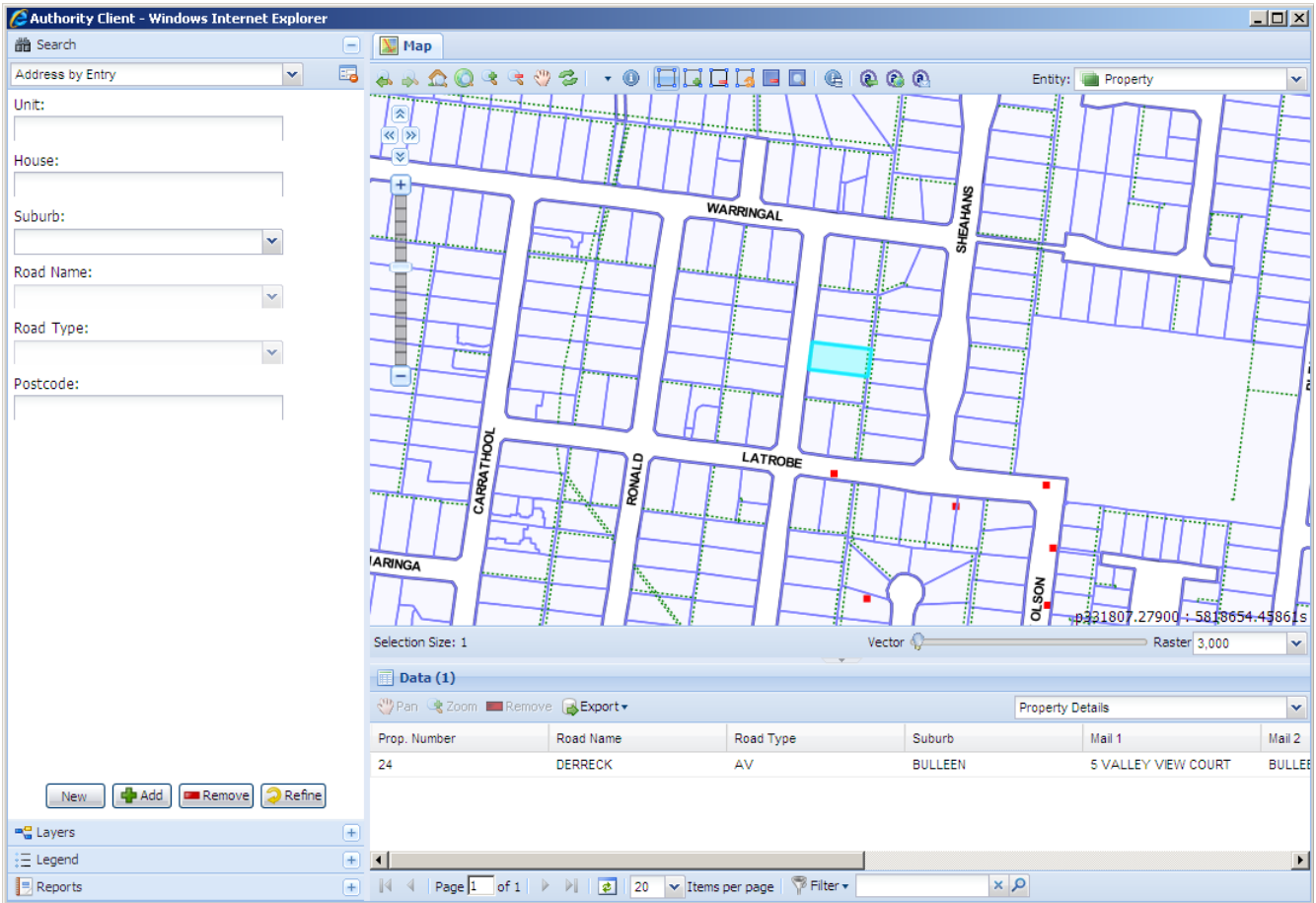
[User Guide](#)

[Developer Guide](#)

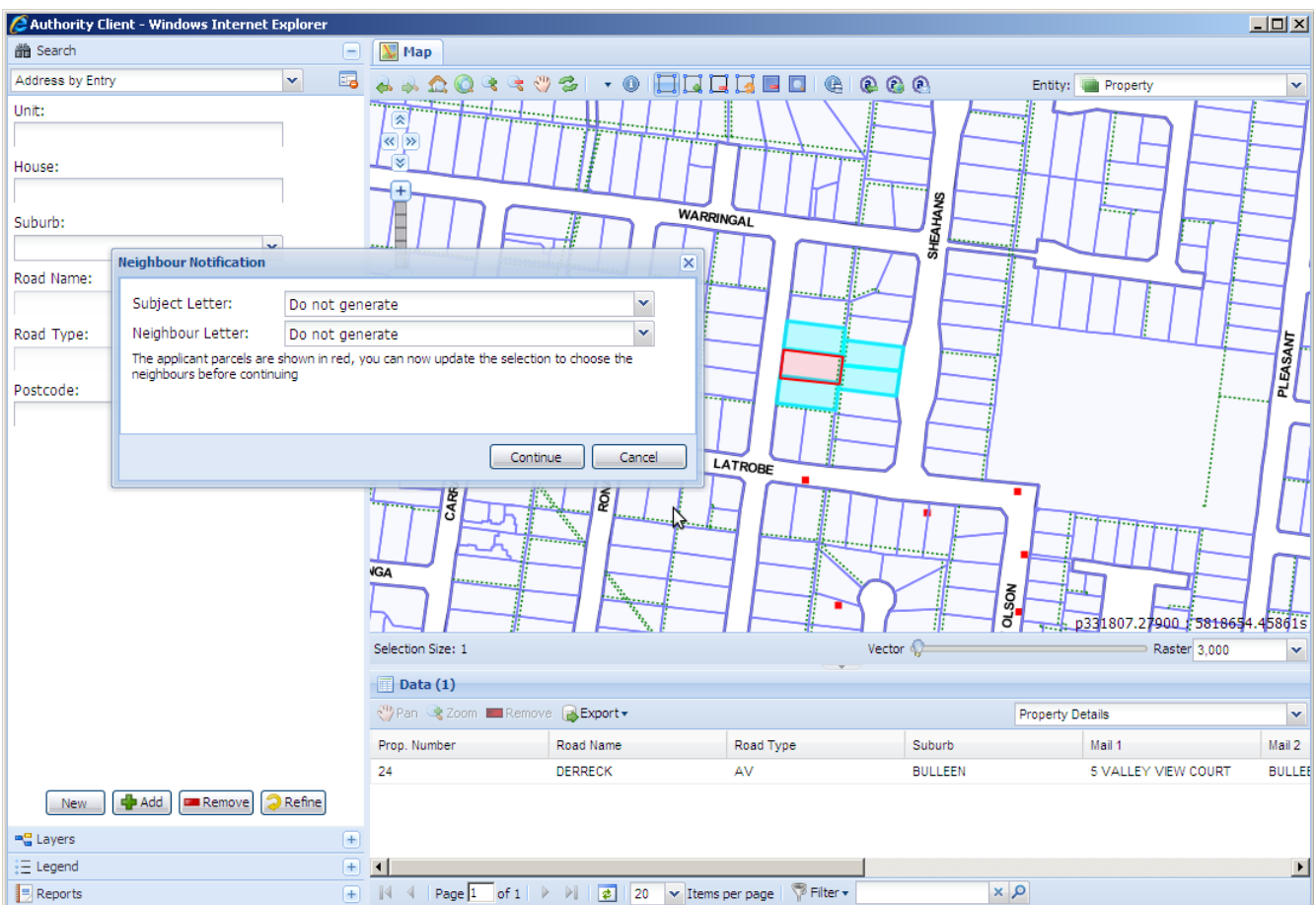
#### Screenshots

**The weave client displaying the three Authority tools after being sent a parcel from Authority**

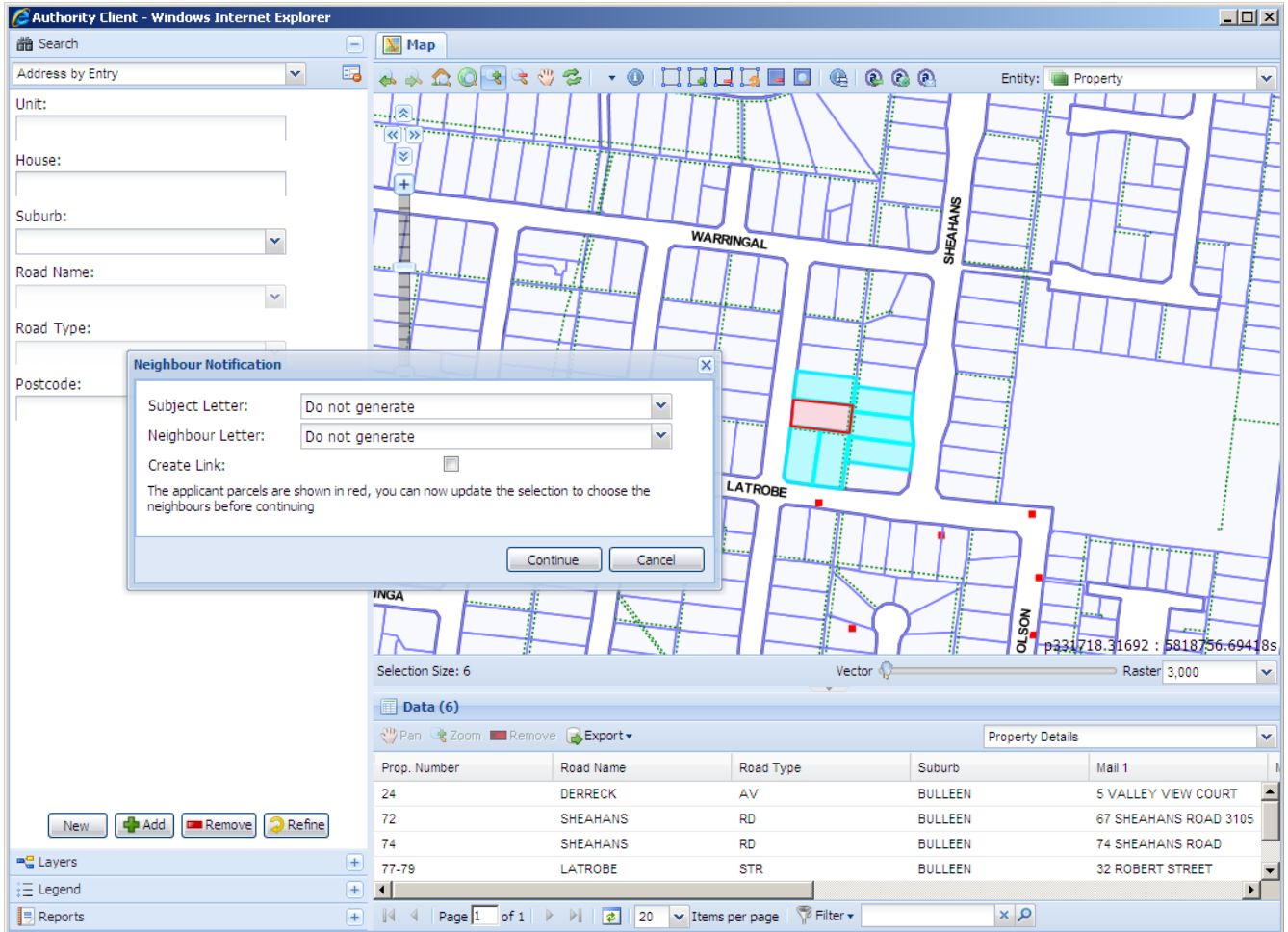




A neighbour notification initiated from the Weave client on the parcel previously send from Authority



## A neighbour notification initiated from Authority



### Integrating with Confirm

The current Weave/Confirm integration supports a two way link between Weave and Confirm. That is Confirm can send the current set of assets to Weave for display and Weave can send the current set of assets for display in Confirm or have them saved as a named selection in Confirm or have Confirm display it's asset selection screen. Currently Internet Explorer is the only browser supported for this type of integration.

### Client Workstation

The link between Weave and Confirm requires installation of components on each client computer that will be requiring communication between Weave and Confirm.

The following two section outline the installation and configuration of these components.

### Installation

There are 3 components that need to be installed, one is Confirm specific and the other two are common to a number of integration bundles so may already be installed if you're integrating with other third party applications, for example Pathway.

The Confirm specific component is [WeaveConfirm.exe](#), and this executable contains both the component that Weave uses to talk to Confirm and is the executable that Confirm calls to talk to Weave.

The preferred location (according to the Confirm documentation) for the WeaveConfirm.exe is

```
C:\Program Files\SouthBank Systems\Confirm\GIS\
```

and once copied to the local PC the WeaveConfirm.exe must be registered by executing it with the -regserver command line parameter

```
"C:\Program Files\SouthBank Systems\Confirm\GIS\WeaveConfirm.exe" -
regserver
```

This will register the COM components installed within the executable so that the browser running the Weave client can access them. Confirm will access the executable directly, but that must be configured via the registry manually which is covered later.

The other components to be installed are [WeaveLink.dll](#) and [WeaveLinkImpl.exe](#). Their preferred location is

```
C:\Program Files\Cohga\Weave\
```

and once copied to the local PC both components must be registered, the WeaveLinkImpl.exe in the same way as WeaveConfirm.exe, but WeaveLink.dll using regsvr32

```
regsvr32 /s "C:\Program Files\Cohga\Weave\WeaveLink.dll"
"C:\Program Files\Cohga\Weave\WeaveLinkImpl.exe" -regserver
```

## Configuration

All registry settings for the Confirm applications reside off the following key

```
HKEY_LOCAL_MACHINE\Software\SouthBank Systems\
```

Each Confirm application installed on a workstation has its own key in the registry which resides under the key specified above and is named the same as the Confirm application.

```
HKEY_LOCAL_MACHINE\Software\SouthBank Systems\\
```

For mapping applications the required keys are stored at the following location (which may not exist and in this case will need to be created):

```
HKEY_LOCAL_MACHINE\Software\SouthBank Systems\\Mapping\
```

Within the Mapping key (at least) two string values must be created, one called 'CurrentEngine' that contains the name of the mapping system that should be used by confirm, in this case the value should be 'Weave'

Additionally an 'Engine' string value should also be created under the Mapping key, and should be set to the name of the mapping system, in this case also 'Weave'. There needs to be a separate Engine entry for each mapping application that Confirm can communicate with, and because of this the string value name should be suffixed with a number, for example if Weave is the only mapping system then the string value name will be 'Engine1', but if there is already another mapping system configured then the name may have to be 'Engine2'.

Name	Value
	Weave

CurrentEngine	
Engine1	Weave

Then the setting for each GIS to interact with Confirm should be created in their own keys under this Mapping key, for Weave the name of the new key will again be 'Weave'

```
HKEY_LOCAL_MACHINE\Software\SouthBank Systems\\Mapping\Weave
```


Under this key the actual setting for Confirm to be able to communicate with Weave will be set

Name	Description
ApplicationName	The full pathname of the WeaveConfirm.exe executable
LinkType	This should be the type of link that used between Confirm and Weave, for Weave it should be 'EXE'
ExportFile	The full pathname to the transfer file
SelectCommand	The command line parameters to pass to WeaveConfirm.exe when Confirm wants Weave to display the current assets. Select "%1" "<Window Title>" <Startup URL> The quotes are important You need to replace <Window Title> with the Internet Explorer window title for the Weave client You need to replace <Startup URL> with the URL required to start Weave if it's not already running

Example values would be:

Name	Value
ApplicationName	C:\Program Files\SouthBank Systems\Confirm\GIS\WeaveConfirm.exe
LinkType	EXE
ExportFile	C:\SBS\Transfer\con2gis.txt
SelectCommand	Select "%1" "Weave Confirm Client" http:\weave:8080\weave\confirm.html

This example assumes that there is a Confirm specific client configuration setup in Weave, with the id 'confirm' and title 'Weave Confirm Client'. This doesn't have to be the case and a more generic client configuration could also be used.

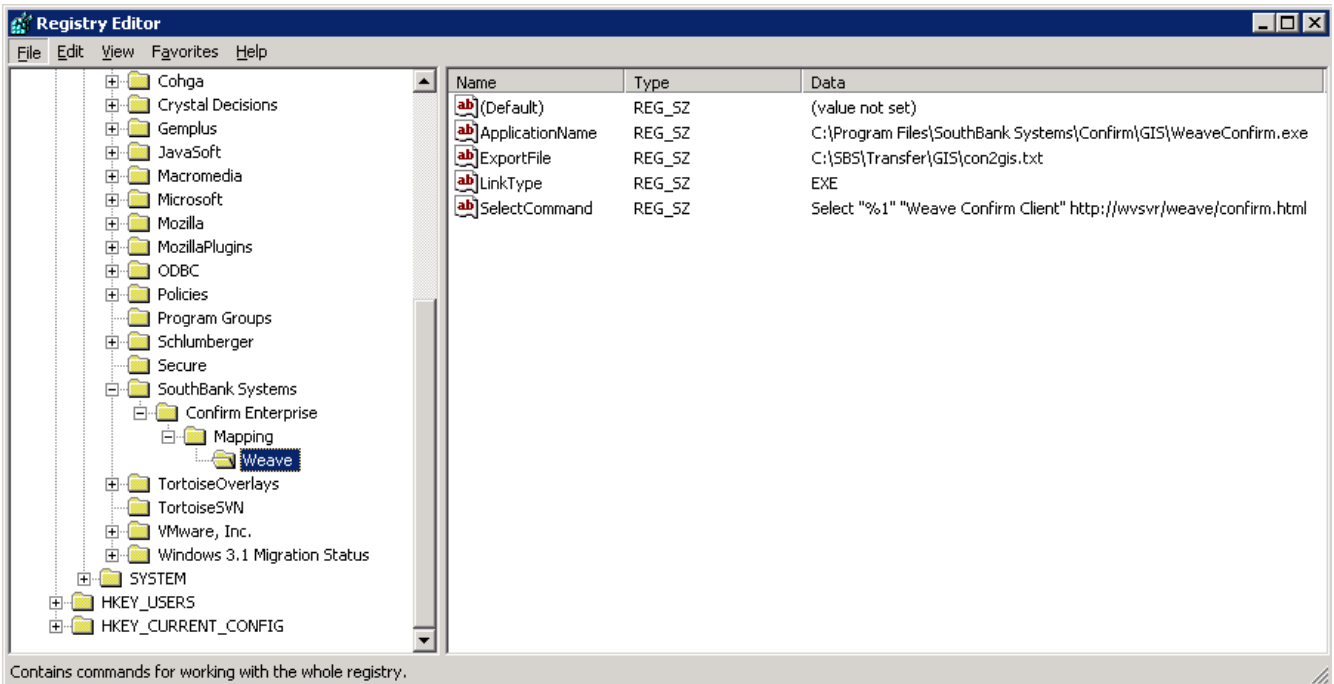
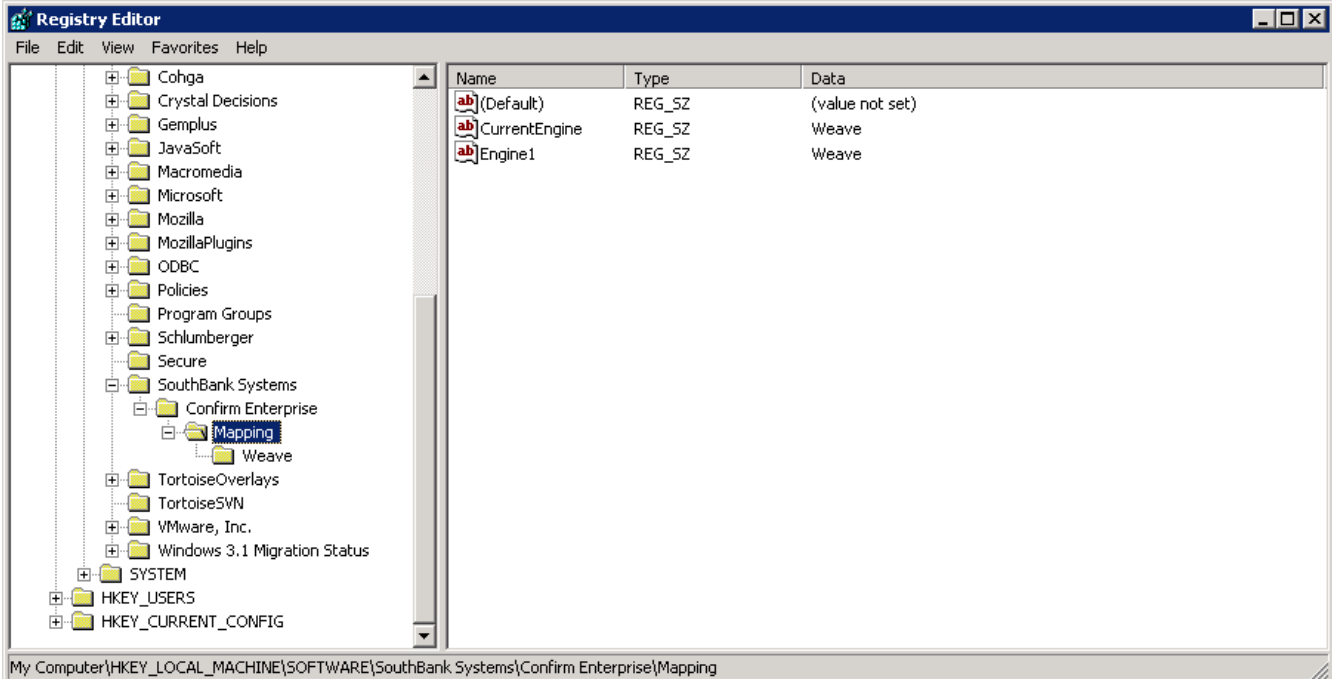
 The directory for the export file may need to be created. It's believed that Confirm does not automatically create this directory if it does not exist and the transfer will fail if the file can not be created.

The following is a Windows [registry export file](#) that can be edited and merged with the registry on each client PC to apply the required settings.

```
Windows Registry Editor Version 5.00
\[HKEY_LOCAL_MACHINE\SOFTWARE\SouthBank Systems\]
\[HKEY_LOCAL_MACHINE\SOFTWARE\SouthBank Systems\Confirm Enterprise\]
\[HKEY_LOCAL_MACHINE\SOFTWARE\SouthBank Systems\Confirm Enterprise\Mapping\]
"CurrentEngine"="Weave"
"Engine1"="Weave"
\[HKEY_LOCAL_MACHINE\SOFTWARE\SouthBank Systems\Confirm Enterprise\Mapping\Weave\]
"ApplicationName"="C:\\Program Files\\SouthBank Systems\\Confirm\\GIS\\WeaveConfirm.exe"
"LinkType"="EXE"
"ExportFile"="C:\\SBS\\Transfer\\GIS\\con2gis.txt"
```

```
"SelectCommand"="Select \"%1\" \"Weave Confirm Client\" http://wvsvr/weave/confirm.html"
```

**Note** that the Confirm application name used in the registry export file is "Confirm Enterprise", and may need to be changed. And also that it's assumed that Weave is the first mapping application that Confirm communicates with, because of the 'Engine1' setting, and this may need to be changed to 'Engine2' or 'Engine3'.



## Server

The Weave server requires the addition of a single bundle to provide the linkage between Weave and Confirm.

The following two sections outline the installation and configuration of this component.

## Installation

The Weave server requires the installation of the [com.cohga.weave.confirm](#) bundle, this bundle should be copied to the ... \weave\platform\plugins directory and an entry added to the ... \weave\platform\configuration\config.ini file to ensure the bundle is started automatically.

Example of addition of Confirm bundle to config.ini

```
...
com.cohga.client.interop@4:start,\
com.cohga.client.details@4:start,\
com.cohga.weave.confirm@4:start,\
...
```

once this is done restarting the server will make the Confirm functionality available on the server.

## Configuration

The configuration of the Confirm bundle in Weave is a two part process, the first step is to configure the Confirm bundle itself, then you add the buttons to the client.

The Confirm bundle uses the namespace

```
com.cohga.weave.confirm
```

So this should be added to your config.xml file, for example

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:confirm="urn:
com.cohga.weave.confirm#1.0">
...
</config>
```

Then you can add the Confirm specific configuration, which involves linking the Weave entity with the feature group in Confirm.


When transferring assets from Confirm to a GIS Confirm provides a list of asset ids and the 'feature group' that each id belongs to, the Weave Confirm configuration provides a link between this feature group and the relevant Weave entity, as well as informing the Weave client that the user is able to send the current selection for any of the listed entities to Confirm.

So to setup the link between Weave and Confirm you would use something like:

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:confirm="urn:
com.cohga.weave.confirm#1.0">
  <confirm:config>
    <entity id="property" featuregroup="PROP"/>
    <entity id="roads" featuregroup="ROAD"/>
  </confirm:config>
</config>
```


In this example the entities in Weave, 'property' and 'roads', will be enabled for transfer from/to Confirm, and the feature group they're assigned to in Confirm is 'PROP' and 'ROAD'

 Note that the two values, 'PROP' and 'ROAD', should be set according to Confirm.

The Weave to Confirm buttons can be either added directly to any toolbar/statusbar in a Weave client configuration or added as a menu.


To add the items individually use the following:

```
<toolbar>
  ...
  <item action="com.cohga.confirm.ShowFeatures" />
  <item action="com.cohga.confirm.SaveAsNamedSelection" />
  <item action="com.cohga.confirm.SelectFeatures" />
  ...
</toolbar>
```

 You don't have to enable all actions, but at least ShowFeatures should be enabled to allow sending of selection from Weave to Confirm

To add them as a menu use the following:

```
<toolbar>
  ...
  <item action="com.cohga.client.actions.menuAction">
    <text>Confirm</text>
    <iconCls>icon-confirm</iconCls>
    <item action="com.cohga.confirm.ShowFeatures" text="Show
Assets" />
    <item action="com.cohga.confirm.SaveAsNamedSelection" text="
Save Assets" />
    <item action="com.cohga.confirm.SelectFeatures" text="Select
Assets" />
  </item>
  ...
</toolbar>
```

 To remove the Confirm label from the menu don't include the <text> item

## Integrating with ECM

Weave supports a two way link between Weave and ECM.

## Client Workstation

The link between Weave and ECM requires the XXX

The following sections outline the installation and configuration of these components.

 **Background**

*ECM* (Enterprise Content Management) is TechnologyOne's electronic content management system.

TechnologyOne's *ECM Classic Client* is their Records Management System that was formerly known as *Dataworks*.

## Integrating with Hansen

There are now 3 versions of the Hansen integration, the older version supports a push/pull type operation but the newer versions have buttons available in Hansen and Weave to call each other directly.

Bundles with version 4.0.0 and later provide the newer integration options, versions earlier than this provide the older style integration.

Neither option requires specific software to be installed on the client PC.

There may need to be configuration performed for Hansen which is not covered by this documentation.

[Interface between Weave and Infor Hansen applications.doc](#)  
[URL Helper to Hansen Deployment Document.pdf](#)

## Server

The Weave server requires the addition of a single bundle to provide the linkage between Weave and Hansen.

The following two sections outline the installation and configuration of this bundle.

## Installation

**i** The newest 5.x version is available via the Weave 2.5 Interop installer and does not need to be manually installed.

The Weave server requires the installation of a single Hansen related bundle

Older: [com.cohga.weave.hansen\\_2.0.2.jar](#)

Newer: [com.cohga.weave.hansen\\_4.1.0.jar](#)

which should be copied to the ...\\weave\\platform\\plugins directory and an entry added to the ...\\weave\\platform\\configuration\\config.ini file to ensure that the bundle is started automatically whenever the server is started.

An example of the addition of the Hansen bundle to config.ini is as follows:

```
...
com.cohga.client.details@4:start, \
com.cohga.weave.confirm@4:start, \
com.cohga.weave.hansen@4:start, \
...
```

Once this is done a restart of the server will make the Hansen functionality available on the server.

## Configuration

The configuration of the Hansen bundle in Weave is a two part process. The first step is to configure the Hansen bundle itself, and then the buttons are added to the client.

### Server

Both Hansen versions uses the namespace:

```
com.cohga.weave.hansen
```

So this should be added to your config.xml file, for example:

**Hansen v2 and v4 Initial Setup**



```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config" xmlns:hansen="urn:com.
cohga.weave.hansen">
...
</config>
```

### Hansen v2 Server Configuration

The older Weave/Hansen integration supports the push/pull method of integration. Using this method, a transfer between the two systems is done in two steps. The first step pushes the transfer to a database from one application, and in the second step the user switches to the other application and pulls that transfer from the database.

The Hansen specific configuration involves defining the data source and table name that will be used to perform the transfer, and specifying which entities should be enabled for transferring between the two systems. Column names in the table can also be changed if they're different from the defaults.

A template [hansen.xml](#) file to configure the server components is shown as follows:

#### Hansen v2 Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:hansen="urn:com.
cohga.weave.hansen#1.0">

  <hansen:config>
    <!-- Must have the following two -->
    <datasource>DATASOURCE_NAME</datasource>
    <table>HANSEN_TABLE_NAME</table>

    <!-- Must have at least one entity -->
    <entity>ENABLED_ENTITY</entity>
    <entity>ANOTHER_ENABLED_ENTITY</entity>
    <entity>AND_YET_ANOTHER_ENABLED_ENTITY</entity>
  </hansen:config>

</config>
```

It is also possible to specify the 'comp type' for each entity that will allow the Hansen bundle to perform an optimisation when transferring selections

#### Hansen v2 Alternate Configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:hansen="urn:com.
cohga.weave.hansen#1.0">

  <hansen:config>
    <!-- Must have the folloing two -->
    <datasource>DATASOURCE_NAME</datasource>
    <table>HANSEN_TABLE_NAME</table>
```

```

    <!-- Must have at least one entity -->
    <entity id="ENABLED_ENTITY" comptype="COMP_TYPE_1" />
    <entity id="ANOTHER_ENABLED_ENTITY" comptype="COMP_TYPE_2" />
    <entity id="AND_YET_ANOTHER_ENABLED_ENTITY" comptype="
COMP_TYPE_3" />
  </hansen:config>

</config>

```

### Hansen v4 Server Configuration

If you're using the newer version of the Hansen integration bundle then the configuration is different from version 1.

The configuration maps the Hansen viewer types to Weave entities, you need to know what the viewer type codes are and which entities they correspond to. Currently more than one entity can map to a single Hansen viewer, but entities can only map to a single viewer type.

The link between the 2 systems is performed by opening a browser window with a URL containing the viewer type and the id's to transfer between them.

#### Hansen v4 Configuration

```

<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:hansen="urn:com.
cohga.weave.hansen#1.0">

  <hansen:config>
    <!-- Must have the following, the URL to open Hansen for
asset display -->
    <url>http://server/InformationViewer.aspx</url>

    <!-- Must have the following, the URL to create an asset
group in Hansen, available at version 4.1.0 -->
    <group>http://server/HansenInterface/CreateAssetGroup.aspx<
/group>

    <!-- And must have at least one viewer defined -->

    <!-- Mapping a Hansen viewer type to a single Weave entity -->
    <viewer id="HANSEN_VIEWER_TYPE">
      <entity id="WEAVE_ENTITY_ID" />
    </viewer>

    <!-- Mapping a Hansen viewer type to a single Weave entity
and filtering keys -->
    <viewer id="HANSEN_VIEWER_TYPE">
      <entity id="WEAVE_ENTITY_ID" filter="WEAVE_FILTER_ID"
/>
    </viewer>

    <!-- Mapping a Hansen viewer type to multiple Weave entities
-->
    <viewer id="HANSEN_VIEWER_TYPE">
      <entity id="WEAVE_ENTITY_ID" />

```

```

        <entity id="ANOTHER_WEAVE_ENTITY_ID"/>
        <entity id="AND_ANOTHER_WEAVE_ENTITY_ID"/>
    </viewer>

    <!-- Mapping a Hansen viewer type to multiple Weave entities
    and filtering keys -->
    <viewer id="HANSEN_VIEWER_TYPE">
        <entity id="WEAVE_ENTITY_ID" filter="WEAVE_FILTER_ID"
    />
        <entity id="ANOTHER_WEAVE_ENTITY_ID" filter="
    ANOTHER_WEAVE_FILTER_ID"/>
        <entity id="AND_ANOTHER_WEAVE_ENTITY_ID" filter="
    AND_ANOTHER_WEAVE_FILTER_ID"/>
    </viewer>

</hansen:config>
</config>

```

The Hansen viewer types will be something like "WO" for Work Orders, "PRCL" for Parcel or "ASSET" for asset types.

If a filter is specified then the id's coming from Hansen will be passed through the filter before being applied to the entity, and when sending the Hansen the process will be reversed.

#### *Filtering identifiers*

If you need to filter the values then you can specify a filter when setting up the entities and add the definition for the filter. **Note the addition of the new `xmlns:filter="urn:com.cohga.selection.filter#1.0"` namespace.**

#### **Example with filter**

```

<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config" xmlns:hansen="urn:com.
cohga.weave.hansen" xmlns:filter="urn:com.cohga.selection.filter">

    <hansen:config id="production">
        <url>http://server/InformationViewer.aspx</url>
        <viewer id="PROP">
            <entity id="property" filter="hansen.property"/>
        </viewer>
    </hansen:config>

    <filter:db id="hansen.property">
        <datasource>main</datasource>
        <table>GEMS_LINK</table>
        <keycolumn>PID</keycolumn> <!-- column used in spatial database --
    >
        <idcolumn>PRUPI</idcolumn> <!-- column required for trim -->
    </filter:db>

</config>

```

The newest version provides more flexibility in the server configuration, **note the updated version number 5.0 in the namespace urn:com.cohga.weave.hansen#5.0**

### Example Hansen 5.x configuration

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:hansen="urn:com.
cohga.weave.hansen#5.0">

  <hansen:config>
    <!-- Base URL for launching Hansen viewer -->
    <url>http://vhansen/hanseninterface/InformationViewer.aspx?
viewertype=ASSET</url>

    <!--viewer is for Send to Hansen, mapping is for Send
to Weave -->
    <viewer id="96">
      <entity>lyr_buildings</entity>
    </viewer>
    <mapping id="96">
      <entity>lyr_buildings</entity>
    </mapping>

    <viewer id="1000001">
      <entity>lyr_playspaces</entity>
    </viewer>
    <mapping id="1000001">
      <entity>lyr_playspaces</entity>
    </mapping>

    <viewer id="30">
      <entity>lyr_sw_pits</entity>
    </viewer>
    <mapping id="30">
      <entity>lyr_sw_pits</entity>
    </mapping>

    <viewer id="31">
      <entity>lyr_sw_pipes</entity>
    </viewer>
    <mapping id="31">
      <entity>lyr_sw_pipes</entity>
    </mapping>

    <viewer id="32">
      <entity>lyr_sw_nodes</entity>
    </viewer>
    <mapping id="32">
      <entity>lyr_sw_nodes</entity>
    </mapping>

    <viewer id="1000003">
```

```

        <entity>lyr_bbqs</entity>
</viewer>
<mapping id="1000003">
    <entity>lyr_bbqs</entity>
</mapping>

<viewer id="68">
    <entity>lyr_roadseg</entity>
</viewer>
<mapping id="68">
    <entity>lyr_roadseg</entity>
</mapping>

<viewer id="97">
    <entity>lyr_roadpath</entity>
</viewer>
<mapping id="97">
    <entity>lyr_roadpath</entity>
</mapping>

<!-- create asset group in Hansen -->
<group url="http://vhansen/hanseninterface
/CreateAssetGroup.aspx">
    <entity>lyr_buildings</entity>
    <entity>lyr_playspaces</entity>
    <entity>lyr_sw_pits</entity>
    <entity>lyr_sw_pipes</entity>
    <entity>lyr_sw_nodes</entity>
    <entity>lyr_bbqs</entity>
    <entity>lyr_roadseg</entity>
    <entity>lyr_roadpath</entity>
</group>

<!-- Setup a project link -->
<project url="http://wrath:8080/InformationViewer.
aspx">
    <entity>lyr_buildings</entity>
    <entity>lyr_playspaces</entity>
    <entity>lyr_bbqs</entity>
</project>

<!-- Setup a work order link -->
<workorder url="http://wrath:8080/InformationViewer.
aspx">
    <entity>lyr_buildings</entity>
    <entity>lyr_playspaces</entity>
    <entity>lyr_bbqs</entity>
</workorder>

<!-- Filter the bbq id's when sending/receiving -->
<!-- The filter is defined the same as v4 -->
<filter entity="lyr_bbqs" id="hansen.filter.bbq"/>

</hansen:config>

```

```
</config>
```

### Testing

It's possible to test the link from Hansen to Weave without having the Hansen side installed, you just need to open a Weave client with a couple of additional parameters, 'viewertype' and 'srcid', e.g.

```
http://server:8080/weave/hansen.html?viewertype=PROP&srcid=11000,
11001,11002
```

This should open the 'hansen' client (which presumably you've configured with the Hansen button) and select and zoom to the entities that have been setup with the viewer type of 'PROP', performing any filtering required to map 11000, 11001 and 11002 to the appropriate values if filtering was configured for the viewer type in Weave.

Similarly testing the Hansen send button in Weave will open a new browser window with similar parameters but if the Hansen integration components aren't installed then the browser will not display the correct results but you can at least check that the URL parameters are what you'd expect to be passed.

Client

#### v2 Client

The two buttons available for integration between Weave and Hansen are:

#### Hansen v1 Client Button Configuration

```
<item action="com.cohga.hansen.Send" />
<item action="com.cohga.hansen.Get" />
```

These buttons should be added to a toolbar at the appropriate location to make them available to the user.

#### v4/5 Client

The one button available for integration between Weave and Hansen is:

#### Hansen v2 Client Display Button Configuration

```
<item action="com.cohga.hansen.Send" />
```

This button should be added to a toolbar at the appropriate location to make it available to the user.

As of version 4.1.0 of the bundle there is another client button for creating groups in Hansen.

#### Hansen v2 Client Group Button Configuration


```
<item action="com.cohga.hansen.Group" />
```

This button can also be added to a toolbar at the appropriate location to make it available to the user. Or both buttons can be added to a menu in the toolbar instead.

#### Integrating with Hansen (Update)

There's a newer Hansen bundle available, com.cohga.weave.hansen version 5, that contains a major update to the way the integration is configured.

## com.cohga.weave.hansen\_5.0.1.jar - Updated Hansen integration bundle

 The new bundle supports the same functionality as the older versions, along with some new functionality, but the configuration must be changed to use the newer version.

Previously the configuration for Hansen used the `viewer` items to configure the link between the viewer types in Hansen and entity types in Weave, but now the `viewer` items only configure the link between Weave and Hansen when using the Send to Hansen button in Weave.


When sending from Hansen to Weave a new `mapping` item is defined, for the generic Hansen display option, along with some specific mapping types, for creating groups, updating work orders and creating project groups (the last two are new functions that have been added in this update).

The Hansen configuration in Weave is now made up of multiple parts:

- `viewer` and `url`
  - Creates a link between Weave entity types and the viewer type to be displayed in Hansen using the provided url.
- `mapping`
  - Created a link between Hansen viewer types and the entity types to be displayed in Weave.
- `filter`
  - Associates a filter to be used with the id's send to/from Weave when passing a list of id's between Weave and Hansen.
- `group`
  - Provides the Hansen URL to open when creating a group plus the entities that can be used to create a group.
- `project`
  - Provides the Hansen URL to open when creating a project group plus the entities that can be used to create a project group.
- `{{workorder}}`
  - Provides the Hansen URL to open when creating/updating a work order plus the entities that can be used to create/update a work order.

The namespace for the updated configuration should be changed to `urn:com.cohga.weave.hansen#5.0`, e.g.

```
<config xmlns="urn:com.cohga.server.config#1.0"
        xmlns:hansen="urn:com.cohga.weave.hansen#5.0">
```

 The following sections show example configurations as separate `hansen:config` items, but when configuring Hansen they must be combined into a single `hansen:config` item.

Send to Hansen (`com.cohga.hansen.Send`)

To configure the Send to Hansen button you must create at least one `viewer` item and set the `url` value, e.g.

```
<hansen:config id="prod">
  <url>http://wrath:8080/InformationViewer.aspx</url>
  <viewer id="PROP">
    <entity>property</entity>
    <entity>parcels</entity>
  </viewer>
  <viewer id="BLDG">
    <entity>council_buildings</entity>
  </viewer>
</hansen:config>
```

The `url` contains the base URL used to open Hansen.

Each `viewer` item describes the viewer type, which is appended to the Hansen URL, and the entity types and 1) are available to be sent to Hansen, and 2) link to the viewer type.


So, using the above example, the only entity types for which the Send to Hansen button will be enabled for are the `property`, `parcels` and `council_buildings` entities (and only when there's at least one entity selected).


And, when the button is pressed the URL opened for Hansen will be:

```
http://wrath:8080/InformationViewer.aspx?viewertype=<vt>&srcid=<ids>
```

where <vt> will be replaced with the `viewertype` that corresponds to the currently active entity, and <ids> will be the list of id's of the selected entities (after any filters have been applied)

By creating more than one `viewer` item you can specify different viewers to be opened in Hansen depending upon the source entity.

 The complete list of entities listed in all of the `viewer` items must be unique, so that there is only even one viewer type that corresponds to a given entity.

 Previously the `viewer` items also configured the link from Hansen to Weave between the viewer type and entity type, but this is now done with the `mapping` item

#### Send to Weave

To configure the link from Hansen to Weave you must create at least one `mapping` item.

The `mapping` item creates a link between the Hansen viewer type and the entity types that should be updated when a list of id's of a particular viewer type is sent from Hansen to Weave.

The `id` of the `mapping` corresponds to the viewer type send from Hansen, if no `id` is set for a mapping then that mapping will be used if there is no matching `mapping` to the viewer type sent from Hansen.

```
<hansen:config id="prod">
  <mapping id="PROP">
    <entity>property</entity>
    <entity>parcels</entity>
  </viewer>
  <mapping id="BLDG">
    <entity>council_buildings</entity>
  </viewer>
</hansen:config>
```

When Hansen opens a link to Weave it will include the viewer type in the URL along with a list of id's, Weave will use the viewer type in the URL to find the corresponding mapping and then update the entities using the list of id's based on the entities listed in the mapping. In the above example there will be an error if Hansen sends a viewer type to Weave that it doesn't know about, you can handle this with a `mapping` without an `id`, e.g.


```
<hansen:config id="prod">
  <mapping id="PROP">
    <entity>property</entity>
    <entity>parcels</entity>
  </mapping>
  <mapping id="BLDG">
    <entity>council_buildings</entity>
  </mapping>
  <mapping>
    <entity>property</entity>
    <entity>parcels</entity>
    <entity>council_buildings</entity>
  </mapping>
</hansen:config>
```



In this case if Hansen calls Weave with the viewer type "ASSET" then the `property`, `parcels` and `council_buildings` entities will be updated based on the list of id's sent from Hansen.

But if the viewer type is "PROP" then only `property` and `parcels` will be updated, likewise `council_buildings` for the "BLDG" viewer type.

The entity does not have to be unique across all mappings, as is the case for the viewer items, since a given viewer type from Hansen may apply to the same entity.

 You could just set a single mapping with no id to catch everything, but be careful if you have the same keys in multiple entities, since that will update the selection for all entities which probably isn't what you want. It's better to specify the mappings for the correct viewertypes, if you know all of the viewer types sent from Hansen (I don't).

#### Filtering

Occasionally the id's that use in Weave to uniquely identify an entity is different from what is used in Hansen, in these cases you can setup a filter then add a `filter` item to the Weave configuration for Hansen.

This filter will then be applied to the id's sent from Hansen to Weave and the reverse used when sending from Weave to Hansen.

```
<config xmlns="urn:com.cohga.server.config" xmlns:trim="urn:com.cohga.weave.hansen#5.0" xmlns:filter="urn:com.cohga.selection.filter#1.0">

    <filter:db id="hansen.property">
        <datasource>main</datasource>
        <table>GEMS_LINK</table>
        <keycolumn>PID</keycolumn> <!-- column used in
spatial database -->
        <idcolumn>PRUPI</idcolumn> <!-- column required for
trim -->
    </filter:db>

    <hansen:config id="prod">
        <filter id="property" filter="hansen.property"/>
    </hansen:config>

</config>
```

The above example defines a filter, `hansen.property`, and then uses that to filter any `property` related id's sent to/from Weave.

#### Create Group (com.cohga.hansen.Group)

The create group button no longer uses the `viewer` items for configuration as it did before, there is now a separate `group` configuration item that sent the URL and entities that should be enabled for creating a group.

If the `url` is not set for the `project` then the global `url` value will be used.

```
<hansen:config id="prod">
    <group url="http://wrath:8080/HansenInterface
/CreateAssetGroup.aspx">
        <entity>property</entity>
        <entity>parcel</entity>
    </group>
</hansen:config>
```

There should be only one `group` item set in the Hansen configuration, as opposed to the `viewer`, `mapping` and `filter` items which can be repeated.

The above example will enable the Create Group button for the `property` and `parcel` entities (when something is selected) and will open the provided URL with the id's of the selected entities (after applying any filters)

#### Create Project Group (com.cohga.hansen.ProjectGroup)

This is a new function provided in this update and is a modification of the above Create Group button.

The configuration is visually identical to the Create Group function, but name is `project` and the URL will be different, and in fact is optional and the existing `<url>` attribute will be used.

If the `url` is not set for the `project` then the global `url` value will be used.

```
<hansen:config id="prod">
  <project url="http://wrath:8080/InformationViewer.aspx">
    <entity>property</entity>
    <entity>parcel</entity>
  </project>
</hansen:config>
```

Again the entity list determines what entities the button will be enabled for.

#### Update Work Order (com.cohga.hansen.WorkOrder)

Another new function provided with this update is the Work Order Update support.

The configuration for this is similar to the `group` and `project`, e.g.

```
<hansen:config id="prod">
  <workorder url="http://wrath:8080/InformationViewer.aspx">
    <entity>sw_pits</entity>
    <entity>sw_pipes</entity>
    <entity>sw_nodes</entity>
  </workorder>
</hansen:config>
```

Unlike the `group` and `project` items the configuration for work orders describes the link from Hansen to Weave and the link from Weave to Hansen.

That is when Hansen initiates a work order update with Weave, Weave will update the entities listed in the `workorder`, not the entities listed in a `mapping`.

Similarly, when Weave then sends the update back to Hansen it will only send one of the entities listed in the `workorder`, not those listed in `mapping` or `viewer`.

#### Complete example

```
<config xmlns="urn:com.cohga.server.config" xmlns:trim="urn:com.cohga.
weave.hansen#5.0" xmlns:filter="urn:com.cohga.selection.filter#1.0">

  <filter:db id="hansen.property">
    <datasource>main</datasource>
    <table>GEMS_LINK</table>
    <keycolumn>PID</keycolumn> <!-- column used in
spatial database -->
    <idcolumn>PRUPI</idcolumn> <!-- column required for
```

```

trim -->
  </filter:db>

  <hansen:config id="prod">
    <url>http://wrath:8080/InformationViewer.aspx</url>

    <filter id="property" filter="hansen.property"/>

    <viewer id="PROP">
      <entity>property</entity>
      <entity>parcels</entity>
    </viewer>
    <viewer id="BLDG">
      <entity>council_buildings</entity>
    </viewer>
    <viewer id="ASSET">
      <entity>sw_pits</entity>
      <entity>sw_pipes</entity>
      <entity>sw_nodes</entity>
    </viewer>

    <mapping id="PROP">
      <entity>property</entity>
      <entity>parcels</entity>
    </mapping>
    <mapping id="BLDG">
      <entity>council_buildings</entity>
    </mapping>
    <mapping id="ASSET">
      <entity>sw_pits</entity>
      <entity>sw_pipes</entity>
      <entity>sw_nodes</entity>
    </mapping>
    <mapping>
      <entity>property</entity>
      <entity>parcels</entity>
      <entity>council_buildings</entity>
    </mapping>

    <group url="http://wrath:8080/HansenInterface
/CreateAssetGroup.aspx">
      <entity>property</entity>
      <entity>parcel</entity>
    </group>

    <project> <!-- url not set since it's the same as the
base url above -->
      <entity>property</entity>
      <entity>parcel</entity>
    </project>

    <workorder> <!-- url not set since it's the same as
the base url above -->
      <entity>sw_pits</entity>

```

```

                <entity>sw_pipes</entity>
                <entity>sw_nodes</entity>
            </workorder>
        </hansen:config>
    </config>

```

## Integration with Pathway

The Weave/Pathway integration provides a two-way link between Weave and Pathway. That is Pathway can call out to Weave to display information and Weave can call out to Pathway to display information.

For additional information about using the newer Weave Hub integration please see this page [Weave Hub - Third-party Application Integration](#).

### Client Workstation

The link between Weave and Pathway requires the installation of components on each client computer that will be requiring communication between Weave and Pathway.

The following two sections outline the installation and configuration of these components.

## Installation

Two Pathway specific .dll files, pathway.dll and weavepathway.dll, must be copied and registered on each client PC, along with the [weave link components](#).

This should be performed using the regsvr32.exe application provided with Windows, e.g.

```

regsvr32 pathway.dll
regsvr32 weavepathway.dll


```

The Pathway integration components must also be installed on the client, this is supplied by Infor and should be located at . . . \msc\Pathway.Integration.Setup.exe at your Pathway installation location.

## Configuration

The Pathway GIS integration must be configured to use Pathway.COMGIS1 as it's mapping component (weavepathway.dll implements Pathway.COMGIS1 ProglId)

Some versions of Pathway require an executable file to be entered as one of the configuration parameters, a dummy executable, WeaveLaunch.exe, can be used if an executable is required.

 The WeaveLaunch.exe can be found here [WeaveLaunch.exe](#) if required.

## Un-installation

The installation can be undone by using regsvr32 with the "/u" parameter, e.g.

```

regsvr32 /u pathway.dll
regsvr32 /u weavepathway.dll

```

And, also removing the PathMap.ocx registration.

### Server

## Configuration

Server

Entities in Weave must be linked to one of the four levels, `prop`, `parc`, `titl` and/or `strt`, in Pathway.

Firstly the pathway namespace must be added to the `config.xml` file header (or a separate `pathway.xml` file can be created and included in `config.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:pathway="urn:
com.cohga.pathway#1.0">
...
</config>
```

then the pathway configuration must be set up to link the pathway levels with the Weave entities

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:pathway="urn:
com.cohga.pathway#1.0">

  <pathway:config>
    <link level="PARC" entity="property"/>
    <link level="STRT" entity="roads"/>

    <!-- Set the name of the Pathway database to launch if it's not
already running -->
    <!-- this is the newer way of setting what Pathway has to be
launched -->
    <database>Pathway</database>

    <!-- Optionally set the name of the Pathway application to launch
if it's not already running -->
    <!-- this is the older way of setting what Pathway has to be
launched, you should probably use database above -->
    <launch>C:\\Infor\\Pathway\\Demo\\msc\\Pathway.exe</launch>
  </pathway:config>

</config>
```

Client

Once the levels have been created the Pathway related actions can be added to the client configuration.

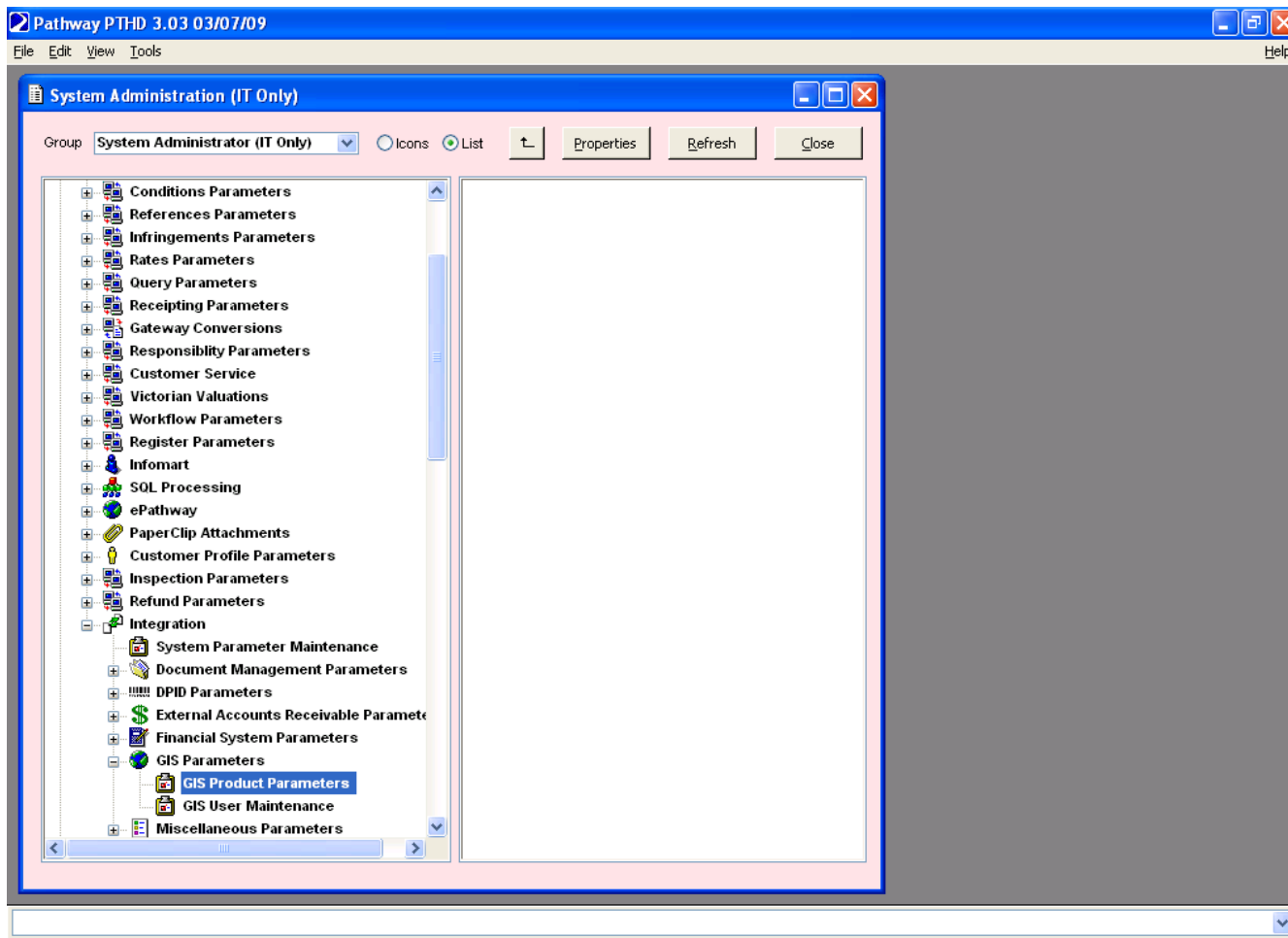
```
<item action="com.cohga.client.actions.menuAction">
  <iconCls>icon-pathway_pathway</iconCls>
  <item action="com.cohga.pathway.Display" text="Display"/>
  <item action="com.cohga.pathway.SendQuery" text="Send Query"/>
  <item action="com.cohga.pathway.DisplayOther" text="Display Other"
/>
  <item action="com.cohga.pathway.SendQueryOther" text="Send Query
Other"/>
  <item action="com.cohga.pathway.CreateLink" text="Create Link"/>
```

```
<item action="com.cohga.pathway.Return" text="Return" />
</item>
```

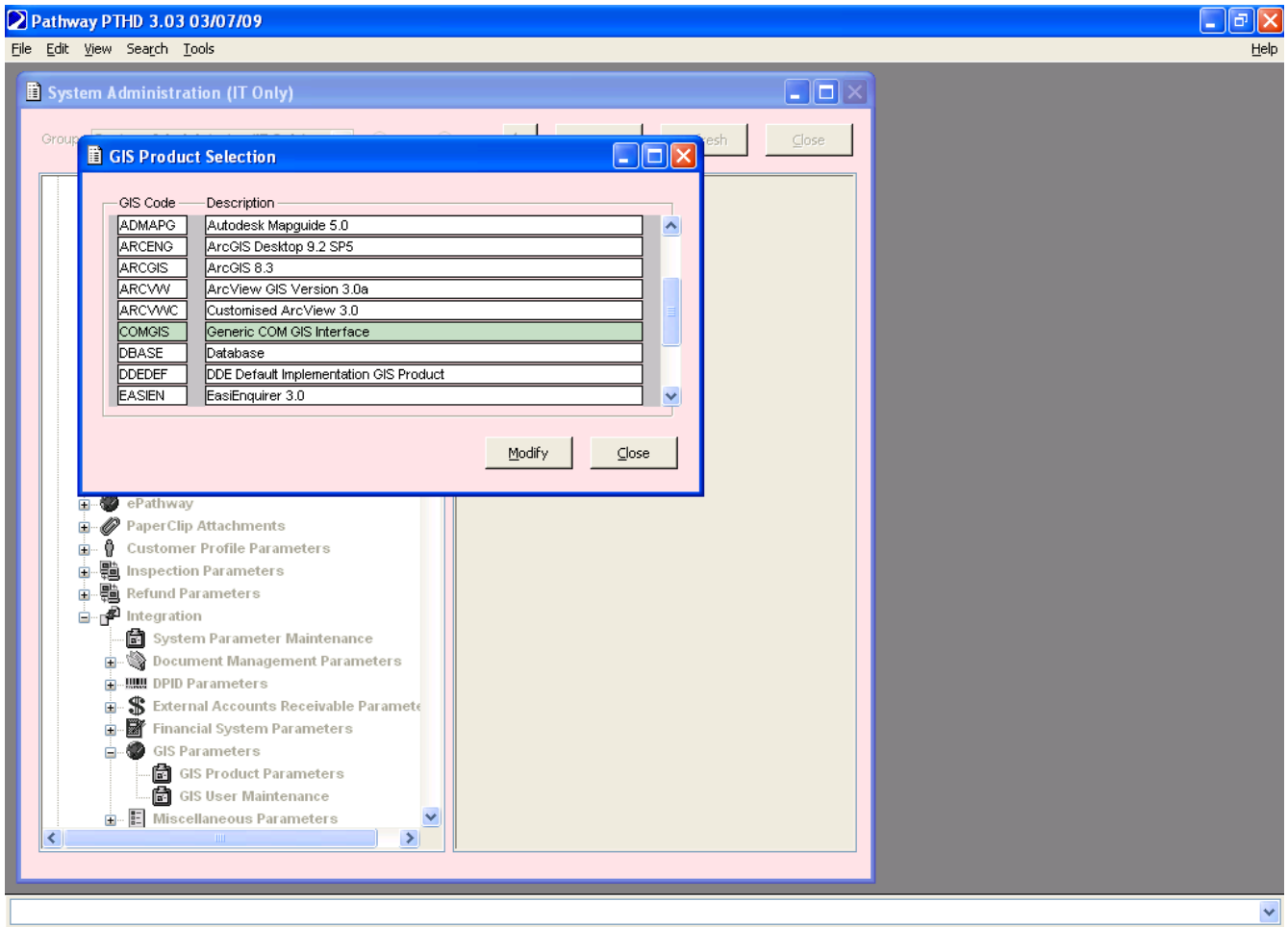
## Pathway Configuration

The configuration of Pathway itself must be done via the Pathway application.

Go to the GIS Product Parameter screen

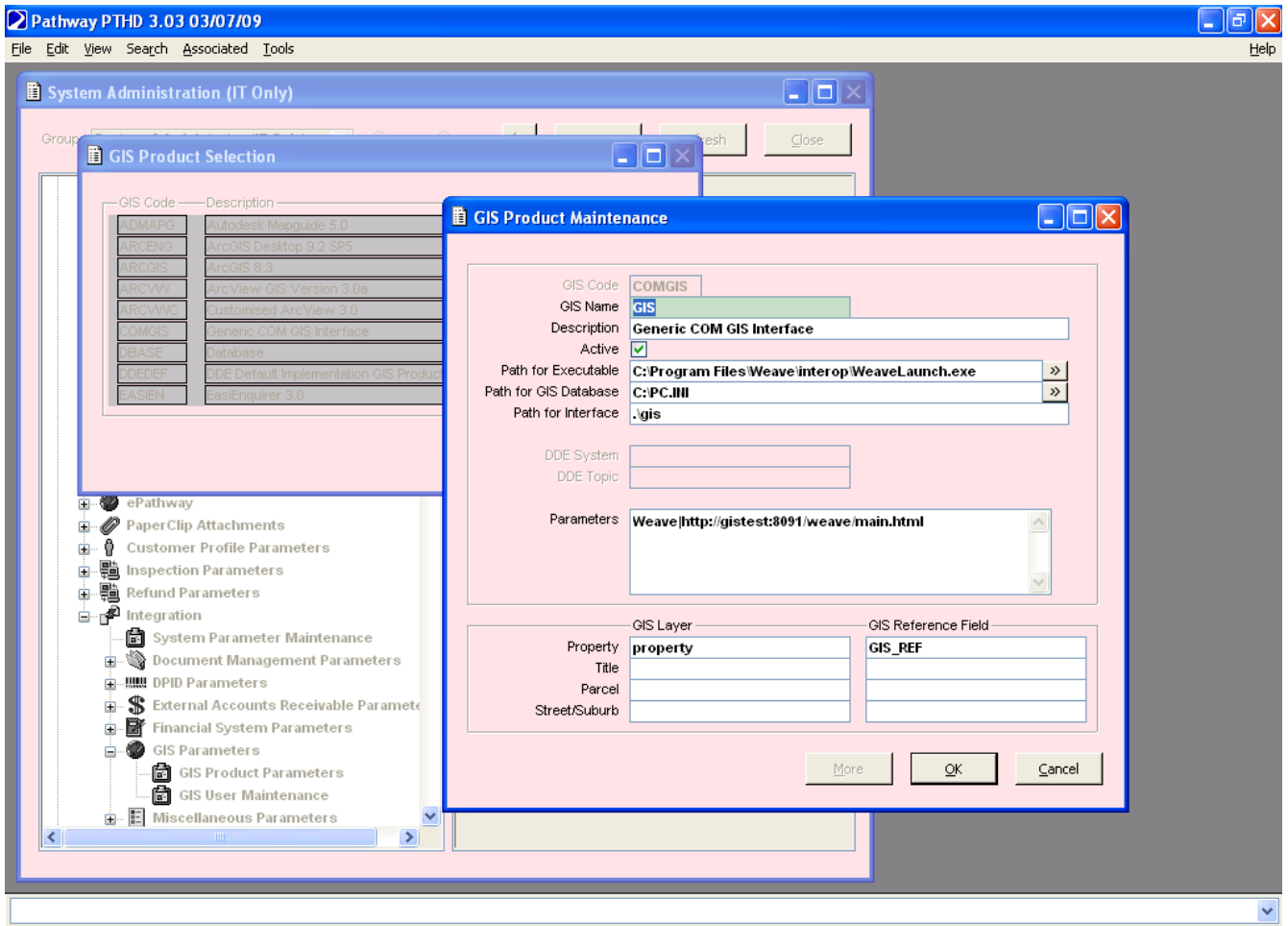


From there edit the Generic COM GIS Interface



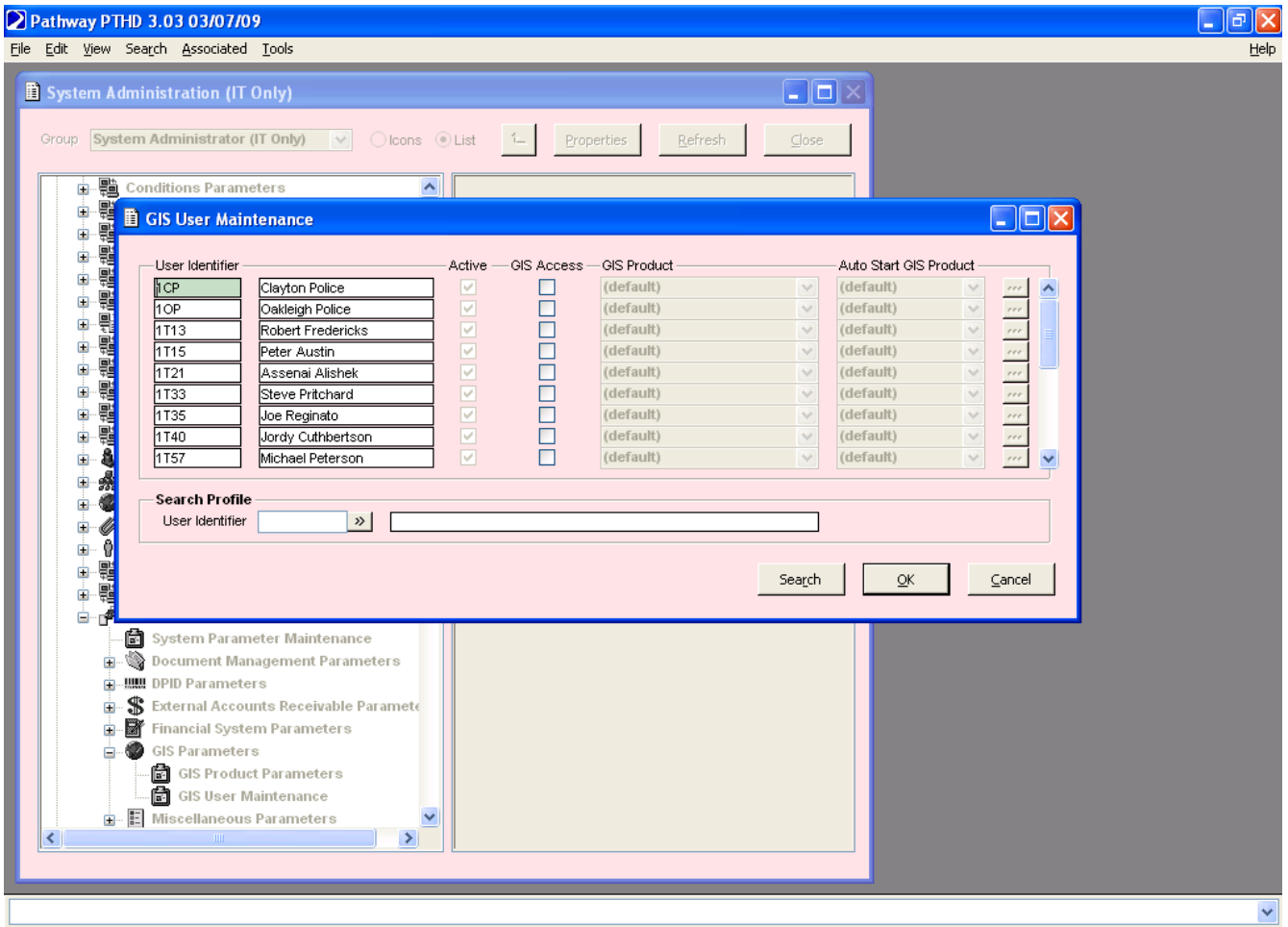
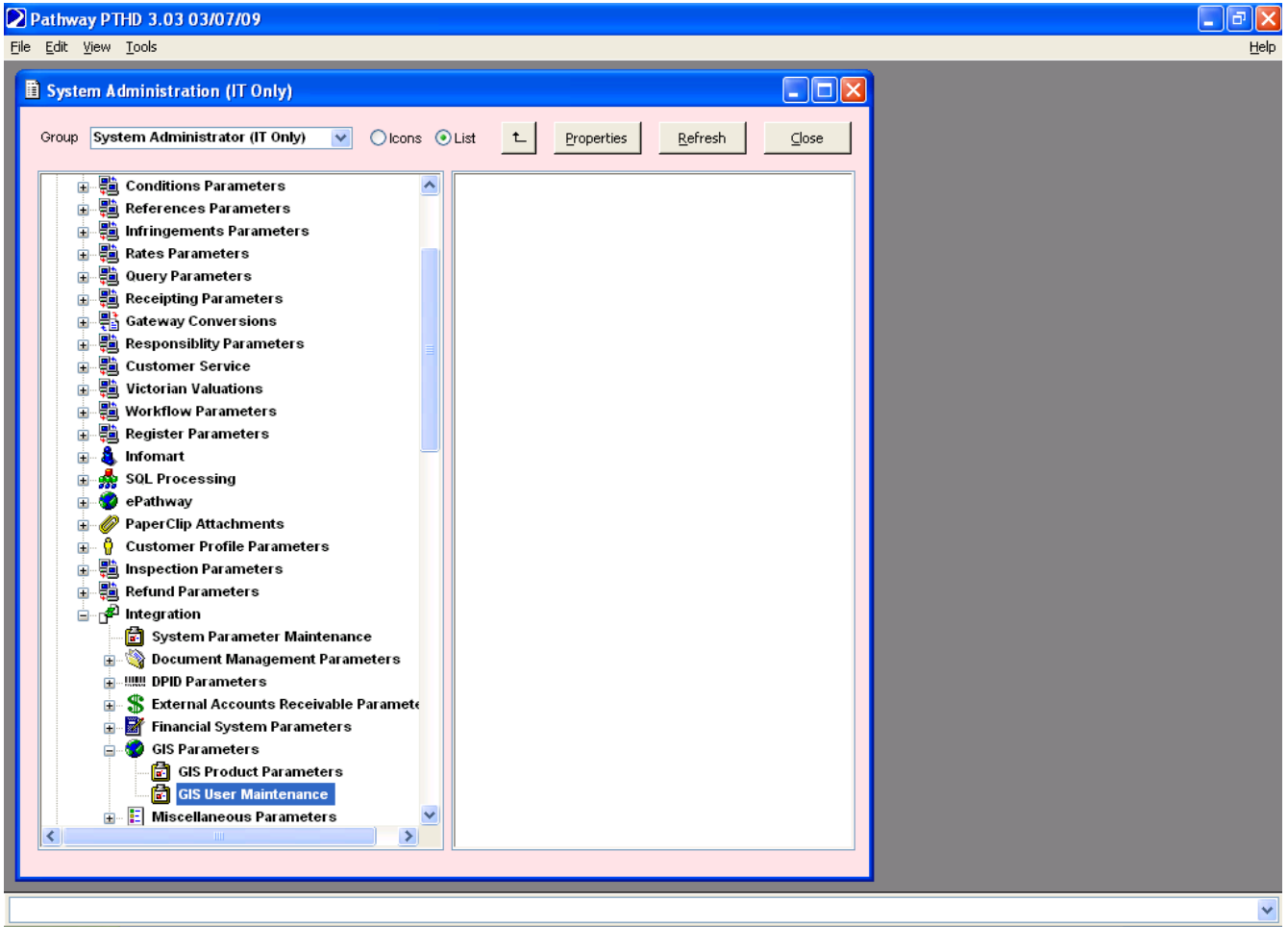
You need to set the GIS Layer and GIS Reference Field values for any items you wish to be able to send to Weave.

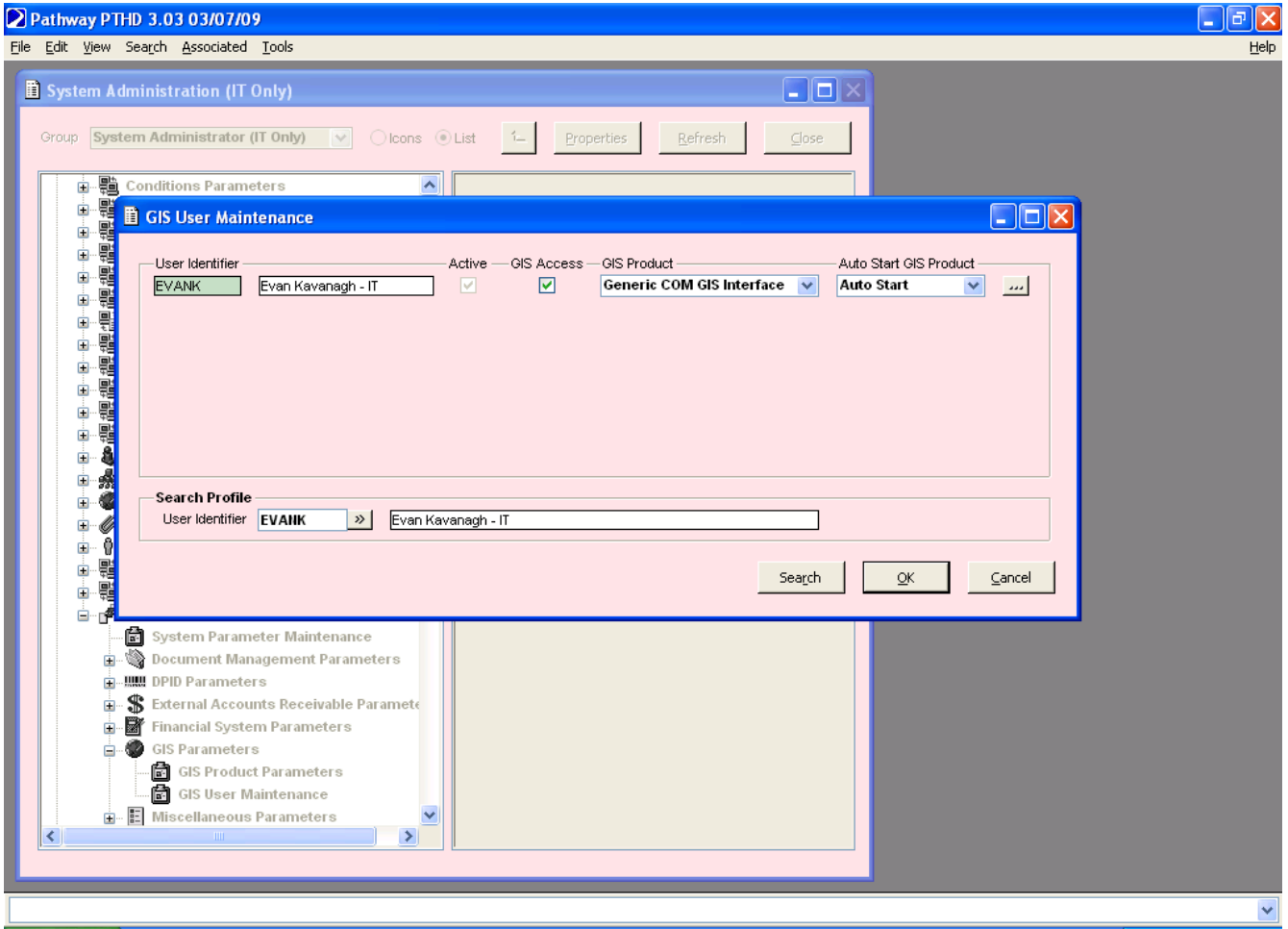
Also you need to set the window title and startup url in the Parameters. These two values are separated by a verticalbar (|), and the first value, the window title, is the text at the start of the Internet Explorer window title which the link uses to locate a running Weave browser window. The second part is the startup URL for Weave for use when there isn't already a Weave browser running, this is the URL that the link will open in a new Internet Explorer browser window to start Weave.

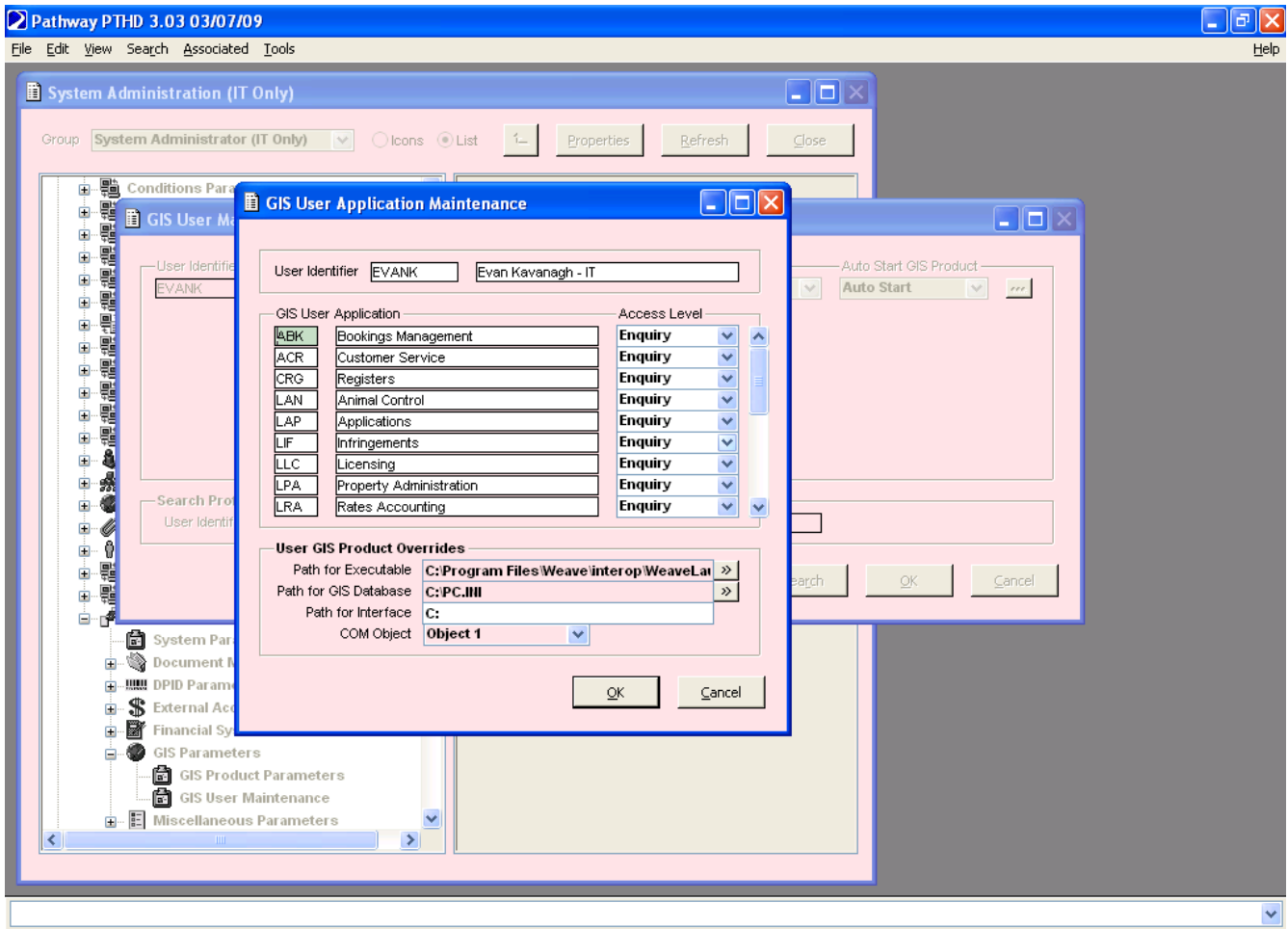


From there you need to configure the users to use Weave as the GIS link (it may be possible to do this globally rather than having to do it individually for each user)









## Integrating with ProClaim

The Weave/ProClaim integration provides a two-way link between Weave and ProClaim. That is ProClaim can call out to Weave to display information and Weave can call out to ProClaim to display information.

For additional information about using the newer Weave Hub integration please see this page [Weave Hub - Third-party Application Integration](#).

## Client Workstation

The link between Weave and ProClaim requires the installation of components on each client computer that will be communicating between Weave and ProClaim.

The following sections outline the installation and configuration of these components.

The latest version of the ProClaim integration requires Weave Hub to be installed on the client PC, see [this page for information on installing Weave Hub](#).

Once Weave Hub is installed on each client PC you need to install the ProClaim add-in for Weave Hub to add support for ProClaim to Weave Hub.

The Proclaim Weave Hub add-in is installed via a .msi file and only provides the option to change the installation directory, which defaults to C:\Program Files\Cohga\WeaveHub\Proclaim\. The installer also creates some registry entries, to register the add-in with Weave Hub.

In addition to the Weave Hub add-in, the installer also installs and registers a COM object that can be used by Proclaim to communicate with Weave. The name of the COM object is `WeaveHubProclaim.MapAutomation` and must be registered within Proclaim as the component to call when communicating with a GIS, which is outlined at the bottom of this page. Like the Pathway and Trim integrations, the COM object just sends messages to Weave Hub which will then forward them on to the browser.

## Server

## Configuration

### Server

For Weave to communicate with ProClaim a number of parameters need to be specified, which can be set separately for different instances of ProClaim.

Firstly the proclaim namespace must be added to the config.xml file header (or a separate proclaim.xml file can be created and included into config.xml)

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:proclaim="urn:com.cohga.proclaim#1.0">
...
</config>
```


then the proclaim configuration must be setup to provide the connection information required for ProClaim and to tell Weave what entities to allow the transfer for.

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:proclaim="urn:com.cohga.proclaim#1.0">

  <proclaim:config id="production">
    <datasourcename>DSN</datasourcename>
    <server>Server</server>
    <database>Database</database>
    <username>Username</username>
    <password>ENCSEMPKCCSFQTJBFMPABFJEBZJAKFCXGME</password>
    <from>Weave</from>
    <to>ProClaim</to>
    <entity id="property" />
    <entity id="street" />
  </proclaim:config>

</config>
```

 In the above example the password has been encrypted using `encrypt osgi` command.

Name	Description
datasourcename	The ODBC datasource (not an Weave data source) used to connect to the ProClaim database, possibly 'ProClaim' or 'Protege'. The value is determined by ProClaim, but you should be able to determine what it might be by looking at the ODBC Administrator on a client PC, where there should be a datasource (either system or user) defined that points to the relevant database.
server	The server that contains the ProClaim database, site specific machine name. This is the host name of the server that is running the ProClaim database. If using SQL Server this may require the instance name, for example SQLSERVER2008\SQLEXPRESS. This is optional if the ODBC data source already specifies the server.
database	The name of the ProClaim database, possibly 'ProClaim' or 'Protege'. This is the name of a database that's running on the server specified by proclaim.server. This is optional if the ODBC data source already specifies the database name.
from	The ProClaim source application id, possibly 'Weave' or 'WEAVE'. This is the application id that the Weave application is registered under in ProClaim.
to	The ProClaim destination application id. This is the application id that the target ProClaim application is registered under in ProClaim.
username	The username to connect to the database. This must be the userid of a valid account on the ProClaim database that has read access to the ProClaim configuration tables.

password	The password to connect to the database. This is the password for the account specified by username.
entity	Each entity tag is a link between Weave and ProClaim

The 'entity' tags setup sources for data to send to ProClaim and targets for data sent from ProClaim.

When setting up a linkage in ProClaim to go from Weave to ProClaim the source entity should be set to the 'id' attribute of the entity tag.

When setting up a linkage in ProClaim to go from ProClaim to Weave the destination entity should be set to the 'id' attribute of the entity tag.

#### Client

Once the server has been configured the ProClaim action can be added to a toolbar in the client configuration. **Note** the change to "proclaim 2" below, it was previously just "proclaim".

```
<item action="com.cohga.proclaim2.Send" version="production"/>
```

You need to include the version to let the Weave to ProClaim link know which connection config to use. The version should match the id of a `proclaim:config` tag.

#### Filtering identifiers

If you need to filter the values before sending them to ProClaim, because ProClaim uses different identifiers for the entity than Weave does, then you can specify a filter when setting up the entities in the config details and add the definition for the filter. **Note the addition of the new `xmlns:filter="urn:com.cohga.selection.filter#1.0" namespace`.**

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:proclaim="urn:com.cohga.proclaim#1.0" xmlns:filter="urn:com.cohga.selection.filter#1.0">

  <proclaim:config id="production">
    <datasourcename>proclaim</datasourcename>
    <server>SQLSERVER2008\SQLEXPRESS</server>
    <database>ProProd</database>
    <username>proclaim</username>
    <password>ENCSEMPKCCSFQTJBFMPABFJEBZJAKFCXGME</password>
    <from>WEAVE</from>
    <to>CES</to>
    <entity id="property" filter="proclaim.production.property"/>
    <entity id="street"/>
  </proclaim:config>

  <filter:db id="proclaim.production.property">
    <datasource>proclaim</datasource>
    <table>GEMS_LINK</table>
    <keycolumn>PID</keycolumn> <!-- column used in spatial database -->
  >
    <idcolumn>PRUPI</idcolumn> <!-- column required for proclaim -->
  </filter:db>

</config>
```

#### ProClaim

When registering the Weave application with ProClaim the Automation Component for Weave is `WeaveHubProclaim` and the Automation Class is `MapAutomation`.

There are two properties for the Weave application that should be set:

- AppWindowName
  - The title of the browser window (matches the start of the actual window title) to locate a running Weave instance.
- URL
  - The URL to start Weave if it is not already running, that is if an Internet Explorer window matching the AppWindowName isn't found

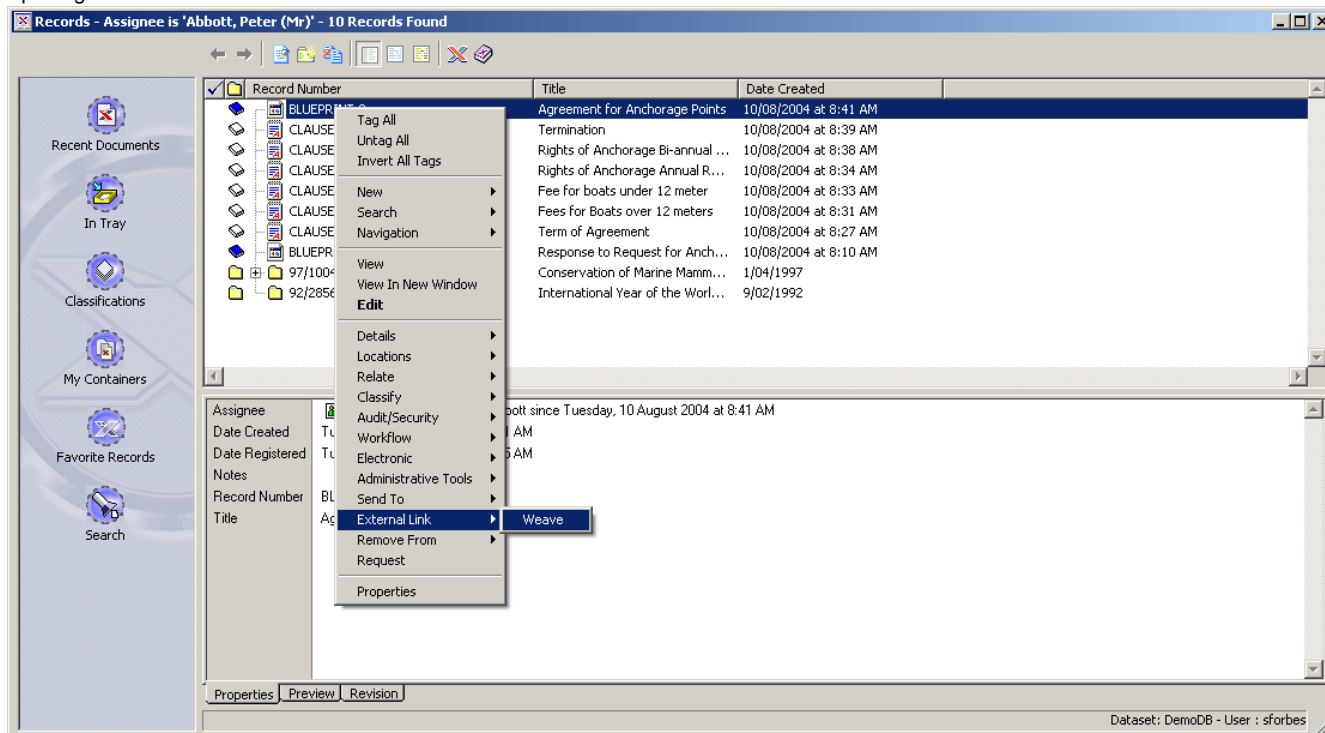
Only Direct Relation criteria type are currently supported.

Integrating with Trim

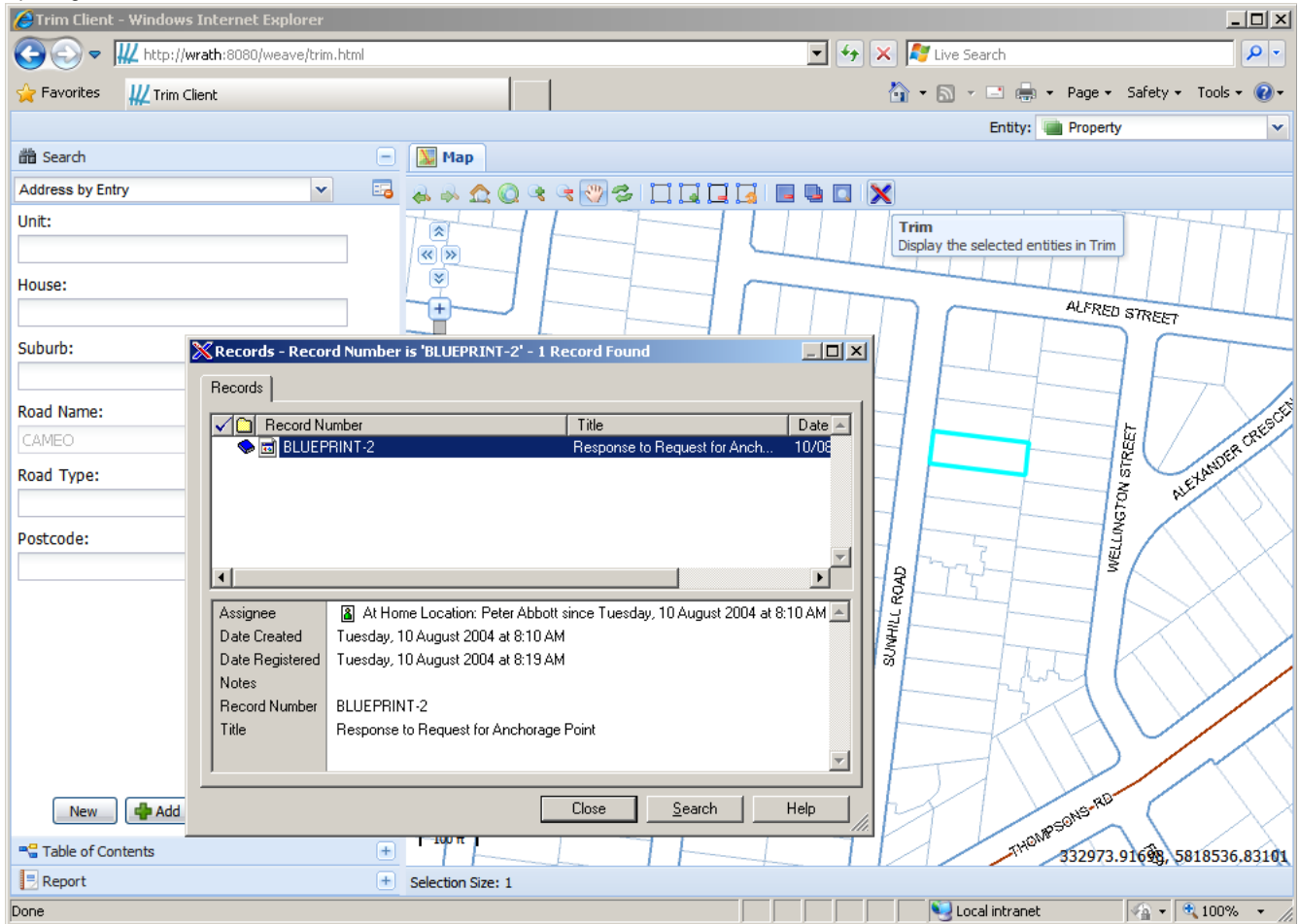
The Weave/Trim integration provides a two-way link between Weave and Trim. That is Trim can call out to Weave to display information about a document and Weave can call out to Trim to display documentation associated with an entity.

For additional information about using the newer Weave Hub integration please see this page [Weave Hub - Third-party Application Integration](#).

Opening Weave from Trim



## Opening Trim from Weave



### Client Workstation

The link between Weave and Trim requires the installation of components on each client computer that will be communicating between Weave and Trim.

The following sections outline the installation and configuration of these components.

#### HP CM 9.x

Installation of the client components is now supported via an msi installation package which can be downloaded as part of the Weave Interop installer, available at <http://downloads.cohga.com/weave/>.

The Interop installer will copy the Weave/Trim client component packages to the `interop\trim\` directory at the installation location specified when running the installer. Note that the Interop installer is intended to be used to install additional components required for a Weave server to interact with Trim, beyond just the client components, and should generally be pointed to a Weave installation directory when it's run, but if you just require the latest versions of the client installation packages you can point the installer to a temporary directory where you can then copy the required installation packages from and then just delete the temporary directory.

There are two installation packages available, `TrimSetup.msi`, for 64-bit Windows, and `TrimSetup32.msi`, for 32-bit windows, which should be copied to each client and installed. The installation requires no additional input from the user so can be automated using standard msi processes.

#### HP RM 8.x

### Downloads

[WeaveTrimLinkNet8.dll](#) - Component to link from Weave to HP RM 8.x

[TrimWeaveLink.exe](#) - Executable to link from Trim to Weave

### Installation

Two Trim specific files, [WeaveTrimLinkNet8.dll](#) and [TrimWeaveLink.exe](#), must be copied to each client PC, along with the weave link components (see [WeaveLink](#) for more information) and additionally [WeaveTrimLinkNet8.dll](#) must be registered.

This should be performed using the RegAsm.exe application provided with Windows, e.g.

```
c:\Windows\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe  
WeaveTrimLinkNet8.dll /codebase /nologo  
c:\Windows\Microsoft.NET\Framework64\v2.0.50727\RegAsm.exe  
WeaveTrimLinkNet8.dll /codebase /nologo
```

## Un-installation

The installation can be undone by using RegAsm.exe

```
c:\Windows\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe  
WeaveTrimLinkNet8.dll /unregister /nologo  
c:\Windows\Microsoft.NET\Framework64\v2.0.50727\RegAsm.exe  
WeaveTrimLinkNet8.dll /unregister /nologo
```

## TRIM

### Downloads

[WeaveTrimLink.dll](#) - Component to link from Weave to Trim  
[TrimWeaveLink.exe](#) - Executable to link from Trim to Weave

### Installation

Two Trim specific files, [WeaveTrimLink.dll](#) and [TrimWeaveLink.exe](#), must be copied to each client PC, along with the weave link components (see [WeaveLink](#) for more information) and additionally WeaveTrimLink.dll must be registered.

This should be performed using the regsvr32.exe application provided with Windows, e.g.

```
regsvr32 "C:\Program Files\Weave\WeaveTrimLink.dll"
```

## Un-installation

The installation can be undone by using regsvr32 with the "/u" parameter, e.g.

```
regsvr32 /u "C:\Program Files\Weave\WeaveTrimLink.dll"
```

## Server

### Installation

The bundles for the server should be installed using the Interop installer for the version of Weave that you are running, which is available from <http://downloads.cohga.com/weave/>.

### Configuration

#### Server

For Weave to communicate with Trim there needs to be a number of parameters specified which link the Weave entities to the database in Trim.



Firstly the trim namespace must be added to the config.xml file header (or a separate trim.xml file can be created and included into config.xml)

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:trim="urn:com.
cohga.trim#1.0">
...
</config>
```

then the trim configuration must be setup to provide the connection information required for Trim and to tell Weave what entities to allow the transfer for.

There can be more than one configuration for trim, for example, one for a production instance and one for a test instance, each identified by their id in the trim:config item (which is production in the following example). Which configuration is used by the client is determined when the Trim action is added to the client configuration (via the "version" attribute of the client configuration action).

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:trim="urn:com.
cohga.trim#1.0">

  <trim:config id="production">
    <entity id="property" field="field" dbid="41"/>
    <entity id="roads" field="field" dbid="41"/>
  </trim:config>

</config>
```

Name	Description
id	The entity that should be enabled
field	The field in the Trim database that contains the id of the selected entity, # for the record number field
dbid	The id of the database on Trim to use
filter	An optional filter that will transform the Weave id's into a value suitable for Trim

#### Client

Once the server have been configured the Trim action can be added to a toolbar in the client configuration.

```
<item action="com.cohga.trim.Send" version="production"/>
```

You need to include the version attribute to let the Weave to Trim link know which Trim config to use, production in this example. The version attribute should match the id of a trim:config tag.

#### Filtering identifiers

If you need to filter the values before sending them to Trim, because Trim uses different identifiers for the entity than Weave does, then you can specify a filter when setting up the entities in the config details and add the definition for the filter. **Note the addition of the new xmlns:filter="urn:com.cohga.selection.filter#1.0" namespace.**

```

<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:trim="urn:com.
cohga.trim#1.0" xmlns:filter="urn:com.cohga.selection.filter#1.0">

  <trim:config id="production">
    <entity id="property" field="field" dbid="41" filter="trim.
property"/>
    <entity id="roads" field="field" dbid="41"/>
  </trim:config>

  <filter:db id="trim.property">
    <datasource>main</datasource>
    <table>GEMS_LINK</table>
    <keycolumn>PID</keycolumn> <!-- column used in spatial database --
>
    <idcolumn>PRUPI</idcolumn> <!-- column required for trim -->
  </filter:db>

</config>

```

#### Trim Configuration

To setup the link from Trim to Weave the Trim application must be configured with an external link to the TrimWeaveLink.exe program. This program will use the WeaveLink API to interact with the Weave client performing any operations necessary to perform the link from Trim to Weave.

This first screen shot shows the external link being set for the record type on trim. The next screen show shows the actual setting required.

**Record Type - Document**

General | Menu | Audit | Defaults | Workflow/Action | Numbering | Electronic  
 User Fields | Form | External Link | Access Controls | Notes | Active

Use an external TRIM Record AddIn Component  
 AddIn PROGID

Use External Links To an Executable  
 External Link (1)

External Link (2)

External Link (3)

OK Cancel Help

The settings required for the link from Trim to Weave are the name of the executable, TrimWeaveLink.exe, plus the parameters to pass to the executable.

Note the executable must have previously been copied to each PC, and the path where the executable was copied to should be used here.

There are four parameters that are specified for the executable:

1. The entity id, in this case 'property', this is the id of the corresponding entity in Weave.
2. The value for the id to send, this is set as a marker using %#...% format, in this case it'll use the PRUPI field attached to the document, this will be different depending upon what value you need to use from the underlying trim document to uniquely identify the corresponding entity in Weave.
3. A filter to use to transform the identifiers sent from Trim to Weave before updating the selection in Weave, if no translation is required then 2 quotes (denoting an empty filter) "" should be used for this parameter, if this value is set then Weave will use a filter that's been previously configured with this id to transform the identifiers to a value suitable for Weave.
4. The window title of the Weave client browser window, this is so that the link can find an existing Weave client before trying to start a new one. This should match the start of the window title of the browser which will generally match the title set in the Weave client configuration.
5. The startup URL, this is the url required to start a new Weave client session if one matching the window title can't be found.

**Define External Link**

Caption  
Weave - Properties

Executable File

Within Special Folder:

Program Files Directory

IRIM Install Directory

Windows Directory

Windows System Directory

None

Executable File Name  
C:\Program Files\Weave\Trim\WeaveLink.exe

Parameters  
property %PRUPI% trim.property "Weave Trim Client" http://se

Working Folder

Finally the field to extract from the document to send to Weave must be specified.

**Insert External Link Parameter**

Record Property Value

Field 3

Record User Defined Field Value

PRUPI

Unique Record Identifier

IRIM DataSet Identifier

Parameter value must be non-blank for link to execute

WeaveLink



#### Internet Explorer 10+

There is a new update to use with Internet Explorer 10 or later. This update should resolve issues with the WeaveLink component when working with newer versions of Internet Explorer.

This link contains an updated version of the WeaveLinkImpl.exe component, along with the current version of the interop bundle for the server.

Note that if you use this version you no longer need to set the X-UA-Compatible header as described in the note below.



#### Downloading WeaveLink


The latest version of the WeaveLink components can be downloaded from [WeaveLink\\_latest.zip](#).

A number of integration modules make use of WeaveLink, which is a set of Windows COM components that provides a means of controlling an instance of the Weave client from a Windows application.


WeaveLink is only supported on a client PC running Windows and also requires the use of Internet Explorer as the client's browser. The Weave client can still be run with other browsers that may be installed on the client PC, but when a third party application tries to communicate with Weave via WeaveLink it will start a new instance of Internet Explorer. Additionally, any buttons/tools within the Weave client that use WeaveLink will not enable themselves when run within a browser other than Internet Explorer.

WeaveLink is supplied as three components, the first `WeaveLink.dll` is the core component that provides the basic interfaces that applications should use to communicate with Weave. This is what third party applications will be programmed against to communicate with Weave.

The other two components are implementations of this interface, `WeaveLinkImpl.exe`, for deployment within a production environment running a Weave server, and `WeaveLinkDebug.exe`, for deployment within a development environment without a running Weave server (primarily for third party developers to develop/test against).


 Use of the WeaveLink component also requires the installation of the client interop bundle on the server, `com.cohga.client.interop`.


Information relating to writing code using the WeaveLink COM API can be found on the page relating to [Third Party Application Integration](#).

 There's a new Weave client launcher helper service available for situations where launching a Weave client via the WeaveLink COM API causes the launching application to freeze.

This service is not required, and should only need to be installed in situations where there are issues with calling applications freezing when calling Weave.

If it's installed on the client PC it will be used to start new instances of the Weave client by the WeaveLink COM API otherwise the existing method will be used.

 You must be using at least version 1.1 of the `WeaveLinkImpl` component to make use of the launcher helper service.

 For IE9 and later you may need to include the following tag in the client config  
`<meta http-equiv="X-UA-Compatible" content="IE=8" />`

## Installation

The WeaveLink components, `WeaveLink.dll` and *one* of `WeaveLinkImpl.exe` or `WeaveLinkDebug.exe`, should be copied and registered on each client PC that will be using an integration module that requires their use (which is generally any implementation that link Weave with a desktop application).

To register `WeaveLink.dll` the `regsvr32` program (supplied with Windows) should be used

```
regsvr32.exe /s "C:\Program Files\Weave\WeaveLink.dll"
```

To register `WeaveLinkImpl.exe` (and similarly `WeaveLinkDebug.exe`) the programs themselves perform the operation

```
"C:\Program Files\Weave\WeaveLinkImpl.exe" /regserver
```

To install the optional launcher helper service

```
"C:\Program Files\Weave\WeaveLinkService.exe" -install
```

## Uninstallation

To remove the components is a similar process

```
"C:\Program Files\Weave\WeaveLinkImpl.exe" /unregserver
```

and

```
regsvr32.exe /u /s "C:\Program Files\Weave\WeaveLink.dll"
```

and

```
"C:\Program Files\Weave\WeaveLinkService.exe" -uninstall
```

## Testing

To test that the WeaveLink components are installed and operating properly the Windows scripting host can be used by creating a small test program.

Copy the following to a new text file called `weavetest.vbs` and change the two parameters, "Weave HTML Client" and "http://hostname:8080/weave/main.htm" to those appropriate for your site.

- The first parameter is the title of the Internet Explorer window that has a running instance of the Weave client. The second parameter is the URL to start Weave, if WeaveLink can't find an Internet Explorer instance with the windows title given in the first parameter.

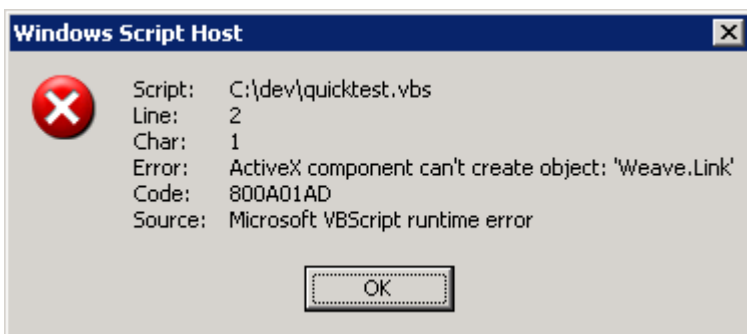
```
Dim link
Set link = CreateObject("Weave.Link")
if not link is Nothing then
    Dim client
    Set client = link.Client("Weave HTML Client", "http://hostname:
8080/weave/main.htm")
    if not client is nothing then
        client.BringToFront
    end if
end if
```

Then by double clicking on the `weavetest.vbs` file in Windows Explorer or running

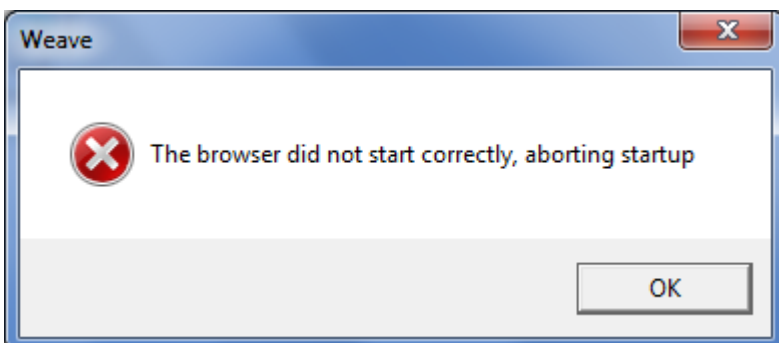
```
wscript weavetest.vbs
```


at a command prompt you should see the Weave client started and brought to the front (or highlighted in some manner if that's not possible). If you've registered `WeaveLinkDebug.exe`, rather than `WeaveLinkImpl.exe`, you'll see a small test form displayed rather than the Weave client.

If you see an error message similar to the following then it means that `WeaveLink.dll` and/or `WeaveLinkImpl.exe` (or `WeaveLinkDebug.exe`) has not been registered correctly.



If you see the following error it probably indicates that the window title parameter is incorrect.



 The newest version of the WeaveLinkImpl.exe doesn't not stop the startup if the window title is incorrect.

## Debugging

Once the components are correctly registered and they do not function correctly then the logged output from the WeaveLink components (and most of the integration modules) can be viewed using the DebugView tool from Microsoft. The tool can be downloaded at [Windows Sysinternals](#).

### Calling WeaveLink from a URL

Previously starting Weave from a URL a new browser window would be created, or at best the current window would be reused but the session would be lost, but as of version 2.4 of the Weave interop bundle it's possible to open Weave using a URL and maintain the current user session.

To accomplish this a new launch page has been created which utilises the WeaveLink component, so that your page doesn't have to.

The "launch" page make use of the WeaveLink component to perform the actual work, so WeaveLink.dll and WeaveLinkImpl.exe still need to be installed on the client PC, but now rather than having to write javascript to talk to the WeaveLink COM API instead you can just open a new URL.

The launch page contains the javascript required to interoperate with the WeaveLink component and uses parameters passed in the URL to the launch page to determine what to do with the component.


The base URL of the launch page is `/weave/interop/launch.html` and it requires at least three parameters `title`, `url` and `event`.

The `title` and `url` parameters are the same values that would be used when getting a client reference with the WeaveLink COM API directly. That is, `title` should be set to the browser window title of the running Weave client instance, so that the WeaveLink component can locate an all ready running Weave client instance. `url` should be set to the URL required to start a new client instance of a running one can not be found using the `title` parameter. And, `event` should be set to the id of an interop action, for example `com.cohga.ZoomSearch`, `com.cohga.SetLocation` or `com.cohga.SetMarker`. The rest of the parameters in the launch URL will be passed to the event as parameters to the event, so depending on the event being triggered different parameters will be required.

```
http://wrath.local:8080/weave/interop/launch.html?title=Weave%
20Client&url=http://wrath.local:8080/weave/main.html&event=com.cohga.
SetLocation&x=145.124&y=-32.412&scale=2000&marker=true
```

It should be noted that there are already way to launch a Weave client using URL parameters that accomplish the same thing, for example launching a client and performing a search to select an entity of interest, and for one off uses of Weave this can still work.

Where this method has an advantage is when there will be multiple interactions between your application and Weave. Using the above URL as an example if your application shows a list of items to the user and you wish to have a link for each item that the user can use to show a map of the item then you could do it using a standard Weave client link, but each client in your application will open a new browser window, by using the launch URL the current Weave browser window will be used and the existing map will just pan across to the new location and the users current ToC settings, active entity, selections, etc remain.

 Obviously opening the launch page will cause a new browser page or tab to be opened, but this window will close as soon as it's either located the existing Weave browser instance or created a new one. For an existing Weave instance this generally happens before the user even has a chance to register what's happening. But, if there isn't an instance of the Weave client already running then this page may stay open for a few seconds, to account for this the page display some text describing to the user what's happening.

## Launching Weave

It's possible to open the Weave client with additional URL parameters to customise its startup.

## Search

You can load the Weave client and ask it to perform a search immediately using the `search` URL parameter.

The value of the parameter should be the id of the search to execute, and the rest of the URL parameters are the values that should be passed to the search.

For example, given the following search definition

### Example search configuration

```
<search:attribute id="property.pid">
  <entity>property</entity>
  <label>By PID</label>
  <datasource>datasource</datasource>
  <table>PROPERTY</table>
  <key>PID</key>
  <parameter id="property_id">
    <promptText>Property ID</promptText>
    <column>PID</column>
  </parameter>
</search:attribute>
```

The following URL will open the Weave client (using the client configured with the id 'main') and execute the `property.pid` search to locate the property with a `PID` of 1234

#### Example URL

```
http://server:8080/weave/main.html?search=property.
pid&property_id=1234
```

This would operate exactly as if the user had started the client, switched to the `By PID` query, typed in 1234 for the `Property ID` and clicked the `New` button.

Note that if the search had more parameters defined then the URL could contain more parameters. The search would be executed by Weave in exactly the same way as if they user entered (or didn't enter) a value for each corresponding parameter when clicking `New`.

Additionally, you can include a `minScale` parameter, to ensure that the client does not zoom in past the minimum scale you specify as the parameter.

## Mapping

You can set the initial map extent, either by specifying an `x` and `y` location and a `scale`, or by specifying `minx`, `miny`, `maxx` and `maxy` values.

Additionally, by specifying a `crs` parameter these values can be in a different coordinate system from the default projection of the client. Also, it's possible with the `x` and `y` version to set a parameter, `marker=true`, to specify that a marker should be placed at the `x`, `y` location. Finally, with the `minx`, `miny`, `maxx` and `maxy` version you can also specify a `minScale` to ensure that the map extent is not zoomed in too far.

#### Example x, y URL

```
http://server:8080/weave/main.html?
x=339690&y=5818946&scale=5000&marker=true
```

#### Example extent URL


```
http://server:8080/weave/main.html?
minx=339680&maxx=339700&miny=5818936&maxy=5818956&minScale=5000
```

Note that the coordinates are assumed to be in the same projection as the map display, if they are different a `crs` parameter can also be included that specifies the EPSG projection code that the coordinates are provided in.




```
http://server:8080/weave/main.html?x=137.2&y=-34.5
&marker=true&crs=4326
```

## Tile Caching

 These instructions describe creating a TMS tile cache, they're not relate to using/creating tile caches in ArcGIS Server.

Weave supports the use of tile caches to improve map display performance on the client.

 Since tile caches are a client side mapping technology unless you have a corresponding server side map engine in Weave that replicates the maps that you can get from the tile cache you will not be able to include the map layers from the tile cache in reports.

Since tile caches are pre-generated in a certain projection, using certain scales and covering a certain area, if you want to enable the use of a tile cache in a client configuration then you will need to use the projection, scales and extent of the tile cache to configure the client.

### Client Configuration

#### Map Engine

To change a map engine to be a tile cache you need to change the map engine type in a client configuration.

The map engine configuration in the client needs to have its 'type' set to 'tilecache' to tell the Weave client that the map it is generating comes from a tile cache rather than the Weave server (the default map engine type is 'weave' if it's not set explicitly).

A tile cache also need to know where to load the tiles from, which is done by setting one or more 'url' parameters that point to the base directory on a web server that's serving the tiles.

You also need to set the 'layername' of the directory under the 'url' that has the actual tiles, since a single web server can be serving multiple tile caches.

There are also some options that may be set for a tile cache, one is the 'format' option, which tells the Weave client what format the tile images are stored in, and should be set to 'image/jpeg', 'image/png' or 'image/gif' depending upon the source tiles. The other option that's required is to set 'singleTile' to false, which tells the Weave client that it should generate the map from multiple smaller tile requests.

This gives us a simple map engine configuration of

```
<mapEngine id="mapengine.aerial.tile">
  <type>tilecache</type>
  <url>http://tileserver.example.com/tilecache</url>
  <layername>aerial</layername>
  <options>
    <format>image/jpeg</format>
    <singleTile>>false</singleTile>
  </options>
</mapEngine>
```

This is assuming that the tile images are available at

```
http://tileserver.example.com/aerial
```

and are stored in the [TileCache](#) format and are JPEG images.

#### Map Scales

A tile cache is generated at fixed scales and as such requires that the client configuration for the map view is told what the scales are (which also means that the client can only view the map at those fixed scales).

To do this you need to ensure that you have a 'scales' setting set for the map view

```

<view id="com.cohga.html.client.map.mapView">
  ...
  <scales>
    <scale>1000</scale>
    <scale>2000</scale>
    <scale>5000</scale>
    <scale>10000</scale>
    <scale>20000</scale>
    <scale>50000</scale>
    <scale>100000</scale>
  </scales>
  ....
  <mapEngine id="mapengine.aerial.tile">
    <type>tilecache</type>
    <url>http://tileserver.example.com/tilecache</url>
    <layername>aerial</layername>
    <options>
      <format>image/jpeg</format>
    </options>
  </mapEngine>
</view>

```

This is assuming that the tile cache has been generated with 7 levels at 1:1000, 1:2000, 1:5000, 1:10000, 1:20000, 1:50000 and 1:100000.

**i** It's also possible to use resolutions/resolution to set the scales (instead of including the scales), since tile caches are generally created based on map units per pixel rather than map scale, so if you're using a pre-generated tile cache and you know the resolution that the tile cache was generated at then you could use them instead. But if you're going to have Weave generate the tile cache then you should use scales.

## Extent

Tile caches are generated to cover a certain area, and the Weave client configuration needs to be told what this area is so that the tiles will line up correctly.

To do this you need to ensure that the 'full' extent for the map matches the tile cache.

### Building a Tile Cache

Weave provides the tools to allow you to build a tile cache based on an existing server side map engine.

To build the tile cache you need to configure a client configuration to use a tile cache, as outlined in the section on [Tile Caching](#), then use the 'tilecache' command from the osgi console to get Weave to build the appropriate tiles.

When building a tile cache there is the extra configuration requirement that there must be a server side map engine available to generate the tiles from, this can be any of the map engines supported by Weave.

To setup the link between the tile cache and the server side map engine you need to tell the client map engine what server map engine to use and what layer within that map engine to turn on, this is done by adding a 'server' option

```

<mapengine id="mapengine.aerial.tile">
  <type>tilecache</type>
  <url>http://tileserver.example.com/tilecache/</url>
  <layername>aerial</layername>
  <options>
    <format>image/jpeg</format>
    <singleTile>>false</singleTile>
    <server>

```

```

    <mapengine>mapengine.raster</mapengine>
    <layers>aerial2005</layers>
  </server>
</options>

```

Where the mapengine is the id of the server side map engine and layers is a comma separated list of the layer ids in that map engine that you want turned on.

## The tilecache command

### Configuring the tilecache command

Executing the tilecache command by itself at the osgi console will display the default setting that will be used to generate the cache

```

osgi> tilecache
--- TileCache ---
base                ... \platform\workspace\.tilecache
extent              full
force               false
format              jpeg
metatile            true
metasize            3
metabuffer          10

```

To change any of the setting you use the tilecache command again, this time with the 'set' command

```
osgi> tilecache set base z:\www\tilecache
```

**i** Any changes made to these settings are kept in the osgi configuration area, so they should persist unless you clean out the configuration directory or start Weave with the -Dosgi.clean parameter

Value	Description
base	The base directory where the files will be written to
extent	Which extent, full, initial or limit, to use as the bounds for the tile cache
force	Should existing files be overwritten when generating the cache
format	What format, jpeg, png or gif, should the tiles be generated in
metatile	Should the tilecache command fetch a larger map from the map engine and split it up to make the tiles, which helps with performance when generating the tile cache
metasize	How much bigger should the map image request be, the tilecache command will request a map images that are metasize x metasize tiles from the map engine
metabuffer	How many pixels extra around the meta tile should the tilecache command request the map image, which helps to ensure that there are no gaps between tiles

**w** You will want to at least update the base setting to ensure that the tile are generated in a more appropriate location. Which would generally be directly to a directory that's already being served out by a web server, otherwise the file will need to be copied to a web server once generated

### Running the tilecache command

Once the tilecache command has been configured and the client configuration is setup correctly you use the tilecache 'seed' command from the osgi console to actually build the tiles.

At the simplest, if you only have a single map engine configured in the client configuration then you can just start the seed command with the id of the client configuration as the only parameter

```
osgi> tilecache seed main
```

But if you have multiple map engines in the client configuration then you also need to provide the name of the map engine

```
osgi> tilecache seed main mapengine.aerial.tile
```

Additionally you can override some of the default tilecache setting in the command

```
osgi> tilecache seed main mapengine.aerial.tile force=true
```

There are two additional parameters that the seed command understands, that aren't configurable by the tilecache set command, and they are 'minlevel' and 'maxlevel'.

These two parameter allow you to set the levels of the tile cache that you want generated, this way you do not have to re-generate the entire tile cache if only data at one scale has changed, so to only generate the second level

```
osgi> tilecache seed main mapengine.aerial.tile minlevel=1 maxlevel=2
```

The minlevel defaults to 0 and the maxlevel defaults to the number of scales plus 1, since minlevel is inclusive and maxlevel is exclusive.

## Spatial Editing Introduction

### ✘ Editing ESRI GeoDatabases

Editing of ESRI GeoDatabases was not previously supported by the spatial editing extension.

If you are editing a database with Weave that has been enabled as a GeoDatabase then it is likely that the GeoDatabase metadata managed by ArcGis is now out of sync and this can result in the inability to create new records with ArcGis.

There is a hotfix available for the 2.5.27, 2.5.28 and 2.5.29 releases of Weave that resolves the issue (currently only for SQL Server) and should be applied as soon as possible, <http://downloads.cohga.com/weave/>.

The hotfix will stop the issue for occurring in the future but will not repair any feature datasets that are already out of sync, and additionally the hotfix will now make it more likely that Weave will no longer be able to create new records (since Weave now follows the same process as ArcGis when creating records).

To repair any feature datasets that have been edited with Weave they will need to be copied to a new feature dataset, the old feature dataset deleted, and the new feature dataset renamed to replace the old broken feature dataset (as this will reset the metadata for the feature dataset).

#### Update:

Postgis/Postgres and Oracle are not supported for editing an ESRI GeoDatabase at all, and in fact GeoDatabase enabling the database may make them unavailable to Weave as a spatial engine completely.

The spatial editing extension for Weave provides a means to edit geometry and attributes for an entity.

#### Features:

- **Simple to setup for basic operation**  
Configure the id, label and entity and Weave will do the rest.
- **Customisable for advanced operation**  
Refine geometry setting and input parameters, and provide pre-defined lists of values for input.
- **Supports editing on an entity with multiple spatial layers**  
A single entity that is composed of separate point, line and/or polygon tables is transparently handled by Weave.
- **Creation and updating on an entity in the client can be initiated from URL parameters**  
The Weave client can be started with parameters in the URL to immediately begin the editing process for a specific entity for the user.
- **Auditing of editing operations**  
Custom auditing can be setup to write a new log records to a database table for each edit operation performed by a user.
- **Snapping of geometry**  
Configurable snapping is available to help with drawing geometry.
- **I18n support**  
All of the text/labels for the editing components can be customized for multiple languages or even just having the default text changed.
- **"Identity" columns**  
It's possible to specify that an attribute for a new record is based on incrementing the previous highest value of the column, or that the column will be automatically generated by the underlying database system.

- **Read-only columns**  
It's possible to specify that an attribute can be displayed but not changed (`readonly`), can only be changed when the entity is created (`readonlyonupdate`) or can only be changed when an entity is updated (`readonlyoncreate`).
- **Hidden columns**  
It's possible to specify attributes that aren't visible to the user but are still written when an edit is performed, using either a fixed value or one of a number of supported functions.
- **Formula columns**  
It's possible to specify a number of in-built formulas as the value for a column, including things like `userid()`, `datetime()`, `entity()`, `area()` and `length()`. These can be used for both the spatial and audit tables.
- **Restrictions on the number and types of geometry**  
Geometry input can be constrained to indicate a minimum and maximum number of geometries that can be entered when creating an entity and the types of geometries can be specified.
- **Customization of client view**  
The display of the input panel can be customized via the configuration. Some things that can be customized are if text and/or icons are display in buttons, which buttons are displayed (hiding the polygon button for example) and the locations of some of the buttons.

#### Limitations:

- **Only attributes directly attached to the spatial table can be edited**  
Weave will only write to the spatial table when editing an entity, currently having some attributes written to the spatial table and other attributes written to separate database tables is not supported. Audit tables may provide enough functionality to support the required workflow though.
- **Spatial mapper for entity must have `<dynamic>` set to true and `<cache>` set to false**  
But this is generally the case for any entity that can have its underlying data altered on the fly.
- **A database table will require a primary key**  
If you're editing data in a database and are unable to edit the table, ensure that it has a primary key.
- **Does not support ESRI GeoDatabase enabled databases**  
Once you've edited a table with Weave the objectid generation might get out of sync with ArcGIS and stop you from being able to edit the table within ArcGIS. Non-geodatabase enabled databases are fine. Update: 2.5.27.14, 2.5.28.10 and 2.5.29.4 now support GeoDatabases editing on SQL Server (other databases to be supported in the future). **Note:** Versioned, Archived and Edit Tracking require special attention to setup in Weave, ensure you're familiar with these before attempting to editing them in Weave, and Branch Versioning is not supported at all.

#### Installation

Currently, the Weave editing sub-system is provided as a single bundle, `com.cohga.spatial.edit`. This bundle must be copied to the `weave\platform\plugins` directory and the server restarted for it to be available.

#### Requirements

### Configuring Entities

Before you can edit an entity there are some criteria that need to be met.

The entity id must be unique within a spatial table. If an entity is composed of multiple spatial tables then it may be the case that the id may not be unique across all of the tables. If for example the entity is composed of more than one geometry type and the underlying spatial engine (e.g. ArcSDE, shapefiles) does not support different geometries in a single table, then the geometry would need to be spread over more than one table. In this situation, the entity id would have to be the same in all of the spatial tables, but it must at least be unique within each table. This is the only way that Weave can determine which row in the table that it needs to update as Weave determines which spatial table to update based on the geometry type stored in the spatial table.

The editing of multiple spatial tables is only fully supported when a different geometry type is stored in each table. That is, if an entity is linked to more than one spatial table then each spatial table should contain a different geometry type (point, line or polygon). This is so that Weave knows which table to write the appropriate geometries to. For example, if you were to have an entity linked to two spatial tables, both containing polygon geometry, then when an edit is submitted that contains a polygon, Weave would not know which table the geometry was supposed to be added to or updated in.

In the case of an entity represented by multiple spatial tables, the schemas for the tables should all be the same. This is a requirement because the same attribute values will be written to each table when an edit is submitted that contains different geometry types.

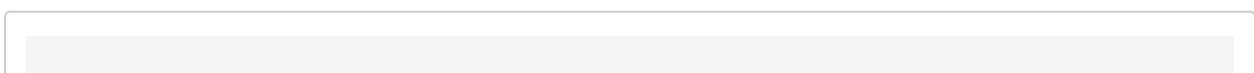
Spatial engines that do not differentiate between geometry types, (i.e. the spatial table stores "geometry" and not specifically points, lines or polygons) are not currently supported in multi-table configurations. This is because in these cases, Weave cannot determine which table to write the geometry to (they are supported in single table configurations). This may change in the future.

Additionally, the spatial mapping associated with any entities that are to be edited should be configured to be dynamic, see [Spatial Mapper](#).

#### Configuration

### Client Edit Plugin

Part of the client side editing sub-system is implemented as a 'plugin' for the map view, so the first thing that needs to be done when enabling the editing sub-system is to register the edit plugin with the map view in the client configuration.



### Adding edit plugin to client

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:client="urn:com.
cohga.html.client#1.0">

  <client:config id="edit">
    <!-- more config items here -->

    <view id="com.cohga.html.client.map.mapView">
      <label>Map</label>
      <location>center</location>

      <!-- Register the edit plugin with the map view -->
      <plugin id="weave.edit"/>

      <!-- map config items here -->
    </view>

    <!-- more config items here -->
  </client:config>

</config>
```

This provides the edit sub-system with a hook into the map view so that it can provide access to the editing layer.

### Client Edit View

When performing an edit a view panel is required to enter/change the attributes associated with the entity being edited. This view is provided by the `com.cohga.client.panel.edit` view and so also needs to be added to the client configuration.

### Adding edit view to client

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:client="urn:com.
cohga.html.client#1.0">

  <client:config id="edit">
    <!-- more config items here -->

    <!-- Add the Edit panel to the client -->
    <view id="com.cohga.client.panel.edit">
      <label>Edit</label>
      <location>west</location>
    </view>

    <view id="com.cohga.html.client.map.mapView">
      <label>Map</label>
      <location>center</location>

      <!-- Register the edit plugin with the map view -->
```

```

<plugin id="weave.edit"/>

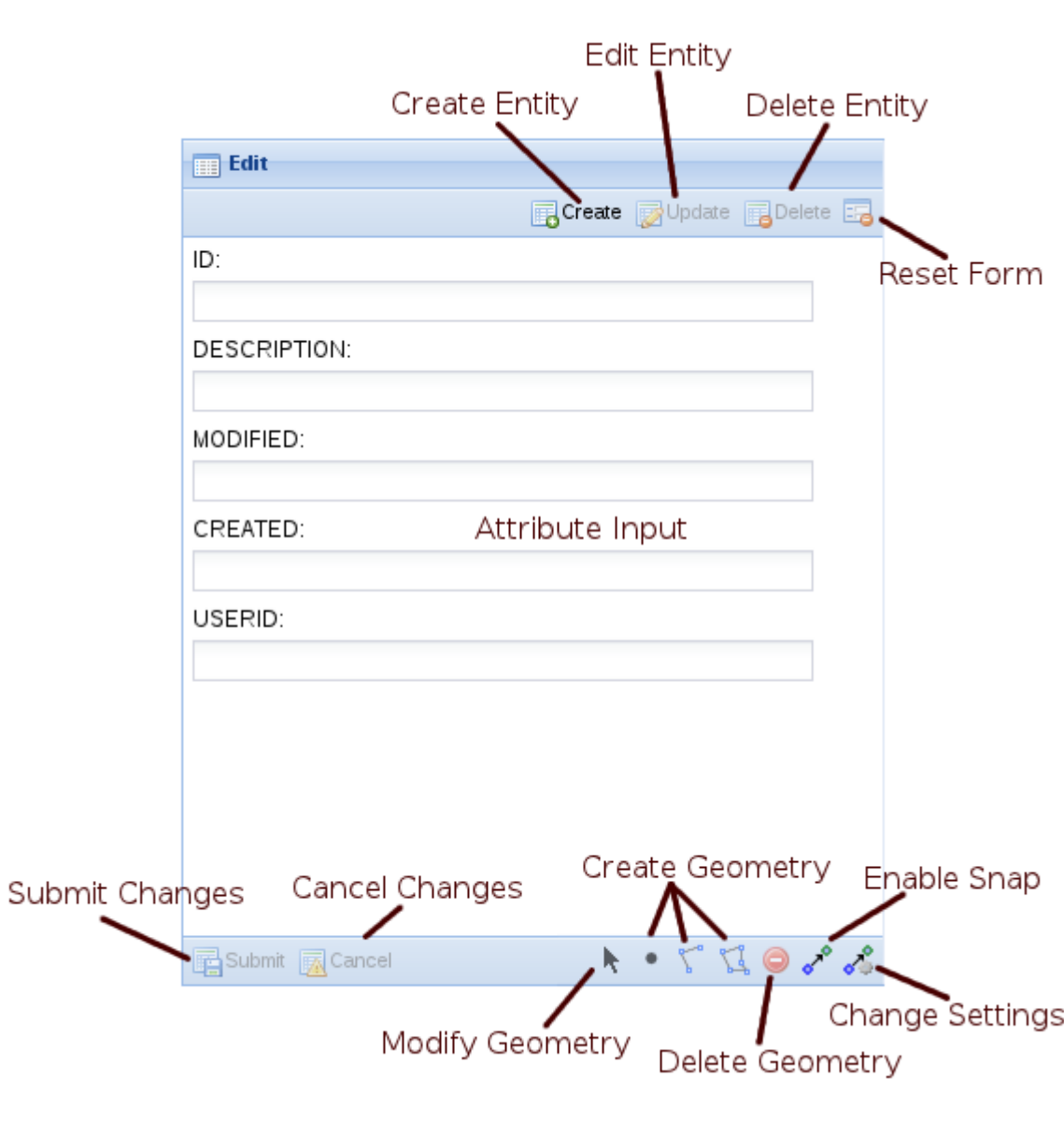
<!-- more config items here -->
</view>

<!-- more config items here -->
</client:config>

</config>

```

This view also provides the actions required to initiate editing and the tools required to perform the spatial edit operations.



Client panel buttons

name	description
Create	Insert a new entity. Will be disabled if editing is in progress.
Update	Change the currently selected entity. Will be disabled if no entity is selected or if there is more than one entity selected or editing is in progress.
Delete	Remove the currently selected entity. Will be disabled if no entity is selected or if there is more than one entity selected or editing is in progress.

Reset	Reset the form fields back to their original values. Will be disabled if editing is not in progress.
Submit	Complete the edit process and save the current changes. Will be disabled when editing isn't in process or if the current state isn't value, for example if a polygon geometry is required but hasn't been added yet.
Cancel	Abort the edit process and abandon the current changes. Will be disabled when editing isn't in process.
Modify	Modify an existing geometry. Will be disabled if there is no geometry attached to the entity.
Point	Create a new point. Will be disabled if the entity doesn't support point geometry or if it only supports a single point and there already is a point attached to the entity.
Line	Create a new line. Will be disabled if the entity doesn't support line geometry or if it only supports a single point and there already is a line attached to the entity.
Polygon	Create a new polygon. Will be disabled if the entity doesn't support polygon geometry or if it only supports a single polygon and there already is a polygon attached to the entity.
Remove	Remove the currently selected geometry. Will be disabled if there is a) no geometry enabled or, b) if the entity requires some geometry and this is the last piece of geometry related to the entity.
Snap	Toggle the snapping function on and off.
Settings	Changes the snapping settings.
Import	The import button allows the user to import geometry from another entity.

#### Customising the client edit view

There are a number of customisation options available to alter the display of the edit view

name	type	default	description
enableCreate	boolean	true	Should the 'Create' button be displayed
enableUpdate	boolean	true	Should the 'Update' button be displayed
enableDelete	boolean	true	Should the 'Delete' button be displayed
enableModify	boolean	true	Should the 'Modify' button be displayed
enablePoint	boolean	true	Should the 'Point' button be displayed
enableLine	boolean	true	Should the 'Line' button be displayed
enablePolygon	boolean	true	Should the 'Polygon' button be displayed
enableRemove	boolean	true	Should the 'Remove' button be displayed
enableSnap	boolean	true	Should the 'Snapping' button be displayed
enableSettings	boolean	true	Should the 'Settings' button be displayed
enableImport	boolean	true	Should the 'Import' button be displayed
showText	boolean	false	Should the text label appear in the create, update, delete and reset buttons?  This only applies to the default toolbar, if you replace the default toolbar with your own you'd need to set showText to the appropriate value for each button, also note that this value defaults to true in that situation.  The default was changed from release 2.6.4, the default was previously true.



			This is removed from release 2.6.5, text will never be shown.
showIcon	boolean	true	Should the icon appear in the create, update, delete and reset buttons?  Note that this only applies to the default toolbar, if you replace the default toolbar with your own you'd need to set showIcon to the appropriate value for each button.  Setting this to false will set showText to true if it is not explicitly set.  This is removed from release 2.6.5, icons will always be shown.
embedButtons	boolean	false	Should the 'Submit' and 'Cancel' buttons be embedded in the form (or remain in the toolbar)  This is removed from release 2.6.4, the buttons will always be embedded in the input panel.

Button display based on showIcon and showText values

	showText unset	showText true	showText false
showIcon unset	icons/no text	icons/text	icons/no text
showIcon true	icons/no text	icons/text	icons/no text
showIcon false	no icons/text	no icons/text	icons/no text

Edit View with showText set to false and showIcon set to true

## Server Edit Configuration

Before an entity can be edited it must have at least one edit configuration associated with it. The edit configuration enables editing for the particular entity and also allows for overriding the details about the way the user can edit the entity. Examples are what attributes the user can edit and or how the attributes are presented to the user.

**i** You can have multiple edits associated with each entity. The user will be able to choose which edit to perform by selecting from a list.

### Basic Server Edit Configuration

The most basic edit configuration interrogates the underlying spatial tables for the information it requires. (e.g. to determine what attributes are associated with the entity or what spatial geometries, point, line or polygon, the user is allowed to create)

---

### Simple edit configuration example

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:edit="urn:com.
cohga.spatial.edit#1.0">

  <edit:config id="simple.edit">
    <entity>graffiti</entity>
    <label>Graffiti</label>
    <description>Report graffiti for removal</description>
  </edit:config>

</config>
```

This provides a simple way to setup editing for an entity and allow the user to directly edit the attributes attached to the entity in the underlying spatial table.

- ✓ The edit configuration has a `publish` setting, which when set to `false` will stop the edit from appearing in the list of edits available for an entity. The edit will still be available but can only be initiated via code or a URL parameter, this provides for the creation of custom editing clients that do not have their edits listed in standard editing clients.

#### Default editing attributes

The basic edit configuration enables all attributes available for editing. Also, the basic configuration does not provide an option to customise how the user will be able to edit those values, which will be determined by the underlying column type in the spatial table and be limited to simple field types (e.g. text fields for string and numbers, calendar field for dates).

#### Default geometry requirements

The types of geometry the user will be able to create or edit will also be determined by the underlying spatial table. If the spatial table contains polygons then the user will only be able to create polygons and they will not be able to create points or lines. If however the entity is linked to two spatial tables, one containing polygons and the other containing points then the user will be able to edit both points and polygons.

There is differentiation between spatial layers that contain single instance geometry and those that contain multi-instance geometry. That is, if the spatial table contains multipolygons, as opposed to just polygons, then the user will be allowed to create more than one polygon shape when editing the entity. When the spatial table contains polygons the user will be limited to a single polygon object. The same differentiation applies to points or multipoints and lines or multilines.

The requirement for the user to create geometry is also determined by the underlying spatial table. If the spatial table allows nil geometry then the user will not be forced to create a geometry when creating a new entity however if the spatial table doesn't allow nil geometries then the user will be required to create a geometry before they will be able to complete the edit.

#### Customising Server Edit Configuration

If you need more control over the attributes and geometry for an edit then you need to override the values that the simple form is generating.

The things that you can override are the parameters that the user can edit, the maximum number of geometry items the user can create, the required number of geometry items the user must create, and if the user can create, update and/or delete an entity.

### Basic custom edit configuration example

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns="urn:com.cohga.server.config#1.0" xmlns:edit="urn:com.
cohga.spatial.edit#1.0">

  <edit:config id="custom.edit">
    <entity>graffiti</entity>
    <label>Grafitti</label>
```

```

<description>Report graffiti for removal</description>
<geometry>
  <point minimum="1" maximum="1"/>
</geometry>
<parameter id="description">
  <label>Description</label>
  <controlType>textarea</controlType>
  <column>DESCRIPTION</column>
</parameter>
</edit:config>

</config>

```

Please note that the <column> value is case sensitive. "DESCRIPTION" is not the same as "Description"

### Custom geometry requirements

Unless explicitly set the types of geometry that a user can create for an entity, and also if the user can create an entity with no geometry, is defined by the underlying spatial tables, if you wish to override this then you do so with the `geometry` settings.

The `geometry` settings determine the minimum and maximum number of each geometry type (point, linestring and polygon) that the user can create, as well as the total number of geometry items (of any type) that the user can create.

This is done by specifying `point`, `linestring`, `polygon` and/or `geometry` items, and for each one specify the minimum and maximum value for each. If no `point`, `linestring` or `polygon` setting is specified then the user will not be able to create those geometry types, unless `geometry` is specified in which case the user can create all geometry types.

Some examples of why you would want to set these values are (these assume that you can't change the underlying spatial table):

Spatial table allows nil geometry but you wish to enforce geometry creation

```
<geometry minimum="1" />
```

or

```
<geometry>
  <polygon minimum="1" />
</geometry>
```

Spatial table allows multi part geometry but you wish to enforce creation of single part geometry

```
<geometry maximum="1" />
```

or

```
<geometry>
  <polygon maximum="1" />
</geometry>
```

The spatial engine supports multiple geometry types in a single table, for example Oracle Spatial, and you wish to limit the geometry

```
<geometry>
  <polygon />
</geometry>
```

You want the user to specify 2 geometries, including 1 polygon and either a point or a line.

```
<geometry minimum="2" maximum="2">
  <polygon minimum="1" maximum="1" />
  <linestring />
  <point />
</geometry>
```

You want the user to specify 2 geometries, including 1 polygon and either a point, line or polygon.

```
<geometry minimum="2" maximum="2">
  <polygon minimum="1" />
  <linestring />
  <point />
</geometry>
```

You want the user to specify 1 or 2 geometries, including 1 polygon and optionally a point or line.

```
<geometry maximum="2">
  <polygon minimum="1" />
  <linestring/>
  <point/>
</geometry>
```

You want the user to specify between 2 and four points.

```
<geometry>
  <point minimum="2" maximum="4" />
</geometry>
```

- i** As of version 0.38 it's possible to specify that multiple geometry types are supported, but only one should be used at a time. That is if the user creates a polygon then they can only create more polygons, even if the edit supports lines. This is done by setting the value for `exclusive` to `true` in the main geometry tag.

### Setting the geometry type to 'exclusive'

```
<geometry maximum="2" exclusive="true">
  <polygon/>
  <linestring/>
  <point/>
</geometry>
```

- i** If `minimum` is not specified it defaults to 0.  
If `maximum` is not specified there will be no upper limit.

### Restricting edit types

You can selectively disable a users ability to create, update or delete entities by setting `create`, `update` or `delete` to `false` in the edit config.

### Create an edit where users can only create new entities

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns="urn:com.cohga.server.config#1.0" xmlns:edit="urn:com.cohga.spatial.edit#1.0">
  <edit:config id="simple.edit">
    <entity>graffiti</entity>
    <label>Graffiti</label>
    <description>Report graffiti for removal</description>
    <update>false</update>
    <delete>false</delete>
  </edit:config>
</config>
```

## Custom attributes

If the default attribute editing setup isn't suitable you can directly specify the attributes that the user can edit and how they appear to the user.

 When any parameters are specified they will completely replace any parameters that may have been created by Weave


This is done by adding one or more `parameter` items to the edit configuration, where each `parameter` specified one input parameter. The format of the `parameter` items in an edit config are an extension of those available for search parameters.

Name	Type	Required	Default	Description
id	string	yes		A unique identifier for the parameter
label	string	yes		The prompt text displayed when user input the parameter value
column	string	no		The name of the column within the table that this parameter references
helptext	string	no		Additional text to display for the parameter to explain how to use the parameter
hidden	boolean	no	false	Hides the parameter from the parameter UI
alignment	'left', 'center', 'right', 'auto'	no	'auto'	How the items should appear in the UI
controltype	'listbox', 'checkbox', 'radiobutton', 'textbox' or 'textarea'	no	'textbox'	The suggested type of UI control to use when displaying the parameter
datatype	'any', 'date', 'time', 'datetime', 'integer', 'string'	no	'string'	The data type for the parameter
allownull	boolean	no	false	Whether a null value is allowed for this parameter
allowblank	boolean	no	true	Give the user the choice of an empty value in the listbox (as opposed to a null value)
allownewvalues	boolean	no	false	Allow the user to enter values not in the listbox already
defaultvalue	any	no		The default value of the parameter
dataset	<a href="#">ref urn:com.cohga.server.data.database#1.0:datadefinition</a>	no		Where to get the values for a listbox
labelcolumn	string	no		Column in the datadefinition that supplies the label of the value to show the user
valuecolumn	string	no		Column in the datadefinition that supplies the value of the value to use in the SQL
uppercase	boolean	no	false	Should the value be converted to upper case in the generated SQL
readonly	boolean	no	false	Can the user change the value
readonlyoninsert	boolean	no	false	Can the user change the value when a new entity is being created
readonlyonupdate	boolean	no	false	Can the user change the value when an entity is being edited
updatable	boolean	no	true	Can the underlying value ever be changed once set (implied readonlyonupdate if set to true)
value	any	formula or any		A value to insert into the database, provides a means of creating values beyond what the user enters (implied readonly if set)
persisted	boolean	no	false	Should the value the user chooses for the field become the default value for the field?
minvalue	any	no		The minimum value allowed for a field. Available from 2.5.28.
maxvalue	any	no		The maximum value allowed for a field. Available from 2.5.28.
minlength	integer	no		The minimum length allowed for a field. Available from 2.5.28.
maxlength	integer	no		The maximum length allowed for a field. Available from 2.5.28.
increment	integer	no		The increment to use for fields that support it, the units are dependant upon the field type. Available from 2.5.28 and currently only for time fields.

A few things to not about the properties that can be applied to parameters:

- `column` is optional, if not provided then the field will still appear, but an attempt to write the entered value into the underlying spatial table will not be made. The user entered value can still be used by other means that will be covered later.
- The `controltype` now supports 'textarea' which isn't available for search fields, it provides a multi-line text input field.
- `readonly`, `readonlyoninsert` and `readonlyonupdate` aren't available for search field either, they're special markers that allow you to alter the ability of the user to change the value in a field. These are client-side flags, and if set alters the display of the field so that the user cannot change the value, the field is still displayed, just not editable.

- `updatable` specifies that once a value is set it can't be changed, this is similar to `readonlyonupdate`, and in fact `readonlyonupdate` will be set to `true` if `updatable` is set to `false`, but it also provides additional checks on the server to ensure that the value is not updated.
- `value` can be used to set a value explicitly by directly supplying the value, or there are a number of formulas that are available.
  - `entity()` The name of the entity type being edited.
  - `userid()` The current username.
  - `ip()` The current users IP address.
  - `datetime()` The current date/time.
  - `operation()` The type of operation being performed will be one of the strings 'create', 'update' or 'delete'.
  - `nextval()` The value used for the column will be the previous largest value from the column plus one, the underlying column must be numeric.
  - `geometry()` A WKT (Well Known Text) representation of the geometry if available, otherwise `null`.
  - `id()` The value for the key field for the spatial table will be used if available, otherwise `null`.
  - `auto()` The value for the field is auto-generated by the database.
  - `count()` The number of geometry objects.
  - `area()` The area of the geometry.
  - `length()` The length of the geometry.
  - `guid()` Generate a new globally unique id.

 When geometry is being written to multiple spatial tables the `count()`, `area()` and `length()` functions will behave differently when used in an edit config item versus an edit audit item. The value will only contain the count, area and length for the geometries that match the table that the geometry is being written to, not the total, when used in an edit config item. But, when used in audit config item then count, area and length will be the total of all the geometries (if there are more than one).

### Custom parameter examples

#### Auditing changes

To record who created and who modifies an entity use:

#### Auditing changes example

```
<edit:config id="custom.edit">
  <entity>graffiti</entity>
  <label>Grafitti</label>
  <description>Report graffiti for removal</description>
  <geometry>
    <point minimum="1" maximum="1"/>
  </geometry>
  <parameter id="description">
    <label>Description</label>
    <controlType>textarea</controlType>
    <column>DESCRIPTION</column>
  </parameter>
  <!-- Hidden parameters to record audit information -->
  <parameter id="createdby">
    <hidden>true</hidden>
    <column>CREATEDBY</column>
    <value>userid()</value>
    <updatable>false</updatable>
  </parameter>
  <parameter id="createdon">
    <hidden>true</hidden>
    <column>CREATEDON</column>
    <value>datetime()</value>
    <updatable>false</updatable>
  </parameter>
  <parameter id="modifiedby">
```

```

    <hidden>true</hidden>
    <column>MODIFIEDBY</column>
    <value>userid()</value>
  </parameter>
  <parameter id="modifiedon">
    <hidden>true</hidden>
    <column>MODIFIEDON</column>
    <value>datetime()</value>
  </parameter>
</edit:config>

```

In this example the audit information is stored in the spatial table itself, so the table must already contain the CREATEDBY, CREATEDON, MODIFIEDBY and MODIFIEDON fields and they need to be character fields for the 'by' fields and timestamps for the 'on' fields.

Also, the about configuration assumes that there is an additional column that's used to identify the records, the id column, but that the column is auto-generated by the database and is not user editable.

Setting an id field

Specifying an field as an id field to be created by using the maximum value from the column.

### Id column example

```

<edit:config id="custom.edit">
  <entity>graffiti</entity>
  <label>Grafitti</label>
  <description>Report graffiti for removal</description>
  <geometry>
    <point minimum="1" maximum="1"/>
  </geometry>
  <parameter id="id">
    <hidden>true</hidden>
    <column>ID</column>
    <value>nextval()</value>
  </parameter>
  <parameter id="description">
    <label>Description</label>
    <controlType>textarea</controlType>
    <column>DESCRIPTION</column>
  </parameter>
</edit:config>

```

This example sets an id field that's hidden from the user and it's created from the existing values in the table. This is not an optimal solution to generating new key values, and it would be much better to use the functionality provided by the underlying database (sequences, auto-generate, identity, etc).

Drop down lists

Specifying that the value for a field is chosen from a list. Here we have a static list, status, included directly in the configuration, and a dynamic list populated from a data definition, reporter.

### Drop down list example

```

<edit:config id="custom.edit">
  <entity>graffiti</entity>
  <label>Grafitti</label>

```



```

<description>Report graffiti for removal</description>
<geometry>
  <point minimum="1" maximum="1"/>
</geometry>
<parameter id="id">
  <hidden>true</hidden>
  <column>ID</column>
  <value>nextval()</value>
</parameter>
<parameter id="description">
  <label>Description</label>
  <controlType>textarea</controlType>
  <column>DESCRIPTION</column>
</parameter>
<parameter id="status">
  <label>Status</label>
  <controlType>listbox</controlType>
  <column>STATUS</column>
  <defaultValue>N</defaultValue>
  <list value="N" label="New" />
  <list value="V" label="Verified" />
  <list value="S" label="Scheduled" />
  <list value="R" label="Removed" />
</parameter>
<parameter id="reporter">
  <label>Reporter</label>
  <controlType>list-box</controlType>
  <dataSet>staff</dataSet>
  <allowNewValues>true</allowNewValues>
  <column>REPORTEDBY</column>
</parameter>
</edit:config>

```

#### Check Boxes and Radio Buttons

##### Check box and radio button example

```

<edit:config id="custom.edit">
  <entity>graffiti</entity>
  <label>Grafitti</label>
  <description>Report graffiti for removal</description>
  <geometry>
    <point minimum="1" maximum="1"/>
  </geometry>
  <parameter id="id">
    <hidden>true</hidden>
    <column>ID</column>
    <value>nextval()</value>
  </parameter>
  <parameter id="checkbox">
    <label>Check Box</label>
    <controlType>checkbox</controlType>


```

```

    <column>CHECKBOX</column>
    <trueValue>Y</trueValue>
    <falseValue xsi:nil="true"/>
</parameter>
<parameter id="radiobutton">
  <label>Radio Button</label>
  <controlType>radiobutton</controlType>
  <column>RADIOBUTTON</column>
  <defaultValue>value2</defaultValue>
  <list>
    <label>Label 1</label>
    <value>value1</value>
  </list>
  <list>
    <label>Label 2</label>
    <value>value2</value>
  </list>
  <list>
    <label>Label 3</label>
    <value>value3</value>
  </list>
</parameter>
</edit:config>

```


#### Writing to other tables

 Currently only writing new records to a separate database table is supported, that is you can not currently update an existing record in an external database table.

So far all attribute values entered by the user have been written directly to the underlying spatial table, but it's also possible to write values, including those entered by the user and those available via value formulas, to another database table. This can be done by creating an `audit` edit configuration item and attaching it to an existing edit configuration item.

The `audit` configuration specified a `datasource` and `table` to write the values to plus a list of parameters that correspond to the columns in the table.

The columns can derive their values either from the values entered by the user as a parameter in the original `edit` configuration or as a formula using a `value` tag or as a hard coded value.

 Parameters in edit configurations can be setup with or without a `column` attribute. If there is a `column` specified then the value will be written to the corresponding column in the underlying spatial table, if there is no `column` specified then no value will be written during this phase. However, the value the user enters for the parameter, in both cases, is available to be written as part of an `audit` configuration.

#### Writing audit information to a separate table

```

<edit:config id="custom.edit">
  <entity>graffiti</entity>
  <label>Grafitti</label>
  <description>Report graffiti for removal</description>
  <geometry>
    <point minimum="1" maximum="1"/>
  </geometry>
  <parameter id="id" hidden="true" label="Id" column="ID" value="
auto()"/>

```

```

    <parameter id="description" label="Description" controlType="
textarea"/>
  </edit:config>

  <edit:audit id="custom.audit">
    <mode>create,update,delete</mode> <!-- specify when audit records
are written. create,update,delete is the default so this isn't really
needed here -->
    <edit>custom.edit</edit>
    <datasource>datasource.main</datasource>
    <table>EDIT_AUDIT</table>
    <parameter column="ID" value="id()"/>
    <parameter column="DESC" parameter="description"/>
    <parameter column="GEOM" value="geometry()"/> <!-- at the moment
this is text, not actual Geometry, in WKT format -->
    <parameter column="USER" value="userid()"/>
    <parameter column="MODIFIED" value="datetime()"/>
  </edit:audit>

```

#### Starting from a URL

The edit process can be started by including either a 'create' or an 'edit' parameter as part of the url when starting the client

```
http://localhost:8080/weave/edit.html?create=custom.edit
```

create will initiate the creation of a new entity, using the configuration specified by the value of the create parameter.

```
http://localhost:8080/weave/edit.html?edit=custom.edit&id=1234
```

edit will initiate the editing of an existing entity, using the configuration specified by the value of the edit parameter and using the entity identified by the id parameter. Also, the client will zoom to the extent of the selected entity.

Both of these URL's also allow you to specify values to be used for the parameter in the edit.

```
http://localhost:8080/weave/edit.html?create=custom.
edit&description=Type%20your%20description%20here
```

#### Localisation

It's possible to localise the text used in the edit panels so they're displayed in the local language of the user, this localisation is performed using the standard localisation mechanism outlined at [Internationalisation and Localisation](#).

The edit panel provides a set of resource id's that it understands and if they're set for the users locale they will be used instead of the defaults values.

id	default
edit.create.text	Create
edit.create.tooltip	Create a new entity
edit.create.notify	Entity successfully created
edit.update.text	Update

edit.delete.tooltip	Edit the selected entity
edit.delete.notify	Entity successfully edited
edit.delete.text	Delete
edit.delete.tooltip	Delete the selected entity
edit.delete.notify	Entity successfully deleted
edit.submit.text	Submit
edit.submit.tooltip	Submit changes
edit.cancel.text	Cancel
edit.cancel.tooltip	Cancel changes
edit.polygon.text	Polygon
edit.polygon.tooltip	Add a polygon
edit.line.text	Line
edit.line.tooltip	Add a line
edit.point.text	Point
edit.point.tooltip	Add a point
edit.modify.text	Modify
edit.modify.tooltip	Click on an item to modify it
edit.remove.text	Remove
edit.remove.tooltip	Remove the currently selected item
edit.reset.text	Reset
edit.reset.tooltip	Reset the form fields
edit.import.text	Import
edit.import.tooltip	Import geometry from another entity
edit.import.error.tooManyGeometries	There are too many geometries in the source
edit.import.error.tooManyPoints	There are too many points in the source
edit.import.error.tooManyLinestrings	There are too many lines in the source
edit.import.error.tooManyPolygons	There are too many polygons in the source

The following resources item can be used to replace the text for all of the items

```

<client:resources>
  <edit>
    <create>
      <text>Create</text>
      <tooltip>Create a new entity</tooltip>
      <notify>Entity successfully created</notify>
    </create>
    <update>
      <text>Update</text>
      <tooltip>Edit the selected entity</tooltip>
      <notify>Entity successfully edited</notify>
    </update>
    <delete>
      <text>Delete</text>

```

```

        <tooltip>Delete the selected entity</tooltip>
        <notify>Entity successfully deleted</notify>
    </delete>
    <submit>
        <text>Submit</text>
        <tooltip>Submit changes</tooltip>
    </submit>
    <cancel>
        <text>Cancel</text>
        <tooltip>Cancel changes</tooltip>
    </cancel>
    <polygon>
        <text>Polygon</text>
        <tooltip>Add a polygon</tooltip>
    </polygon>
    <line>
        <text>Line</text>
        <tooltip>Add a line</tooltip>
    </line>
    <point>
        <text>Point</text>
        <tooltip>Add a point</tooltip>
    </point>
    <modify>
        <text>Modify</text>
        <tooltip>Click on an item to modify it<
/tooltip>
    </modify>
    <remove>
        <text>Remove</text>
        <tooltip>Remove the currently selected item<
/tooltip>
    </remove>
    <reset>
        <text>Reset</text>
        <tooltip>Reset the form fields</tooltip>
    </reset>
    <import>
        <text>Import</text>
        <tooltip>Import geometry from another entity<
/tooltip>
        <error>
            <tooManyGeometries>There are too many
geometries in the source</tooManyGeometries>
            <tooManyPoints>There are too many
points in the source</tooManyPoints>
            <tooManyLinestrings>There are too
many lines in the source</tooManyLinestrings>
            <tooManyPolygons>There are too many
polygons in the source</tooManyPolygons>
        </error>
    </import>
</edit>
</client:resources>

```

**i** It's possible, as of version 0.38 of the edit bundle to specify when, on create, update and/or delete, the audit record is written by specifying a 'mode' value in the audit config as a comma separated list of those three values. e.g `<mode>create</mode>` would just write an audit record when an new entity is created.

The default, if not specified, is `<mode>create,update,delete</mode>`.

## Create New Button

As of release 2.5.21 there is an additional button that can be added to a tool bar or status bar to immediately initiate the creation of a new object.

There's not much to it, it's just a button that can be added to a toolbar and when click immediately starts the creation of a new spatial object of a specific type, and is configured with the type of object to create so you don't have to switch the active entity.

It either takes an entity, in which case the default edit for the entity is used,

```
<item action="weave.edit.createNew" entity="grafitti"/>
```

or it takes an entity and a specific edit, in which case that edit will be used,

```
<item action="weave.edit.createNew" entity="grafitti" edit="grafitti.edit"/>
```

## Overriding Edited Table

It's possible to override the table that Weave will update when the user submits an edit, rather than just using the table associated with the entity in the spatial mapper. This helps when working with views that can't be updated but the table that underlies the view can be.

To tell Weave to update a different table than the one specified in the spatial mapper you need to set a `spatialEngine`, `table` and `key` in the edit config.

```
<edit:config id="edit.view">
  <entity>grafitti</entity> <!-- this entity is backed by a DB view
that can't be edited directly -->
  <spatialEngine>oracle</spatialEngine> <!-- set the spatial engine
that contains the table that should be edited instead, could be the
same as the spatial mapper, but might not be -->
  <table>GRAFITTI</table> <!-- set the name of the table in the
spatial engine that should be edited instead, this should be
different than the spatial mapper, since that's the whole point of
this -->
  <key>OID</key> <!-- set the column that uniquely identifies the
rows in the table that should be used instead, generally this would
be the same as the spatial mapper -->
  <label>Grafitti</label>
  <description>Report grafitti for removal</description>
  <geometry>
    <point minimum="1" maximum="1"/>
  </geometry>
  <parameter id="id" hidden="true" label="Id" column="ID" value="
auto()"/>
  <parameter id="description" label="Description" controlType="
textarea"/>
```

```
</edit:config>
```

## Non-spatial Editing Introduction

The non-spatial editing bundle supports three types of editing

1. Standalone tables
  - These tables have no link to any entities and the user will be able to access all rows, with the ability to create, update and delete rows.
  - These will generally be lookup tables, but could also be used for creating, updating and deleting non-spatial entities.
  - They will be presented to the user as a grid.
  - Initially editing of the values for an individual record will be done in a separate window, with a form showing the values to edit, rather than being performed directly within the grid because of limitations in the current grid editing capabilities in Ext 3.4, specifically relating to cascading parameters (list parameters where the list content depends on values being set in other fields).
  - There can be two types of client configurations for this editor:
    - One that provides the user with a list box so they can choose what edit to perform, for editing lookup tables.
    - And another where the editor displayed in the view is pre-set (via a config option) and the view will then only display rows for that edit.
2. Related tables
  - These are tables that are related to an entity in a one to many relationships.
  - The content for display will change based on the currently selected entity.
  - They will be presented to the user as a grid.
  - These are tables that can contain zero or more rows that relate to the currently selected entity (spatial or non-spatial).
  - There can be two types of configuration for this editor:
    - One that provides the user with a list box so they can choose what edit to perform if there is more than one editor configured for the current entity.
    - And another where the editor displayed in the view is pre-set (via a config option) and the view will then only display the rows for that edit that are associated with the currently selected entity. This would be used to create a more customised editing client, where for example there a number of tabs available that each corresponds to editing a different part of the current entity.
3. Entity tables
  - These are tables that are related to a entity in a one to one relationship.
  - They will be presented to the user as a form.
  - These are tables that can contain zero or one rows that relate to the currently selected entity (spatial or non-spatial).
  - To the user there will be no difference if there exists a corresponding row or not, beyond the fact that if there is a corresponding row that the content of that row will be displayed, as opposed to empty (or default) values being displayed if the corresponding row does not exist. Other than that the user will have the option to change the values and submit the results. On the server if the corresponding row exists it will be updated, if it does not exist it will be created.
  - When the source entity is a spatial entity this type of editor will be a related non-spatial (this should NOT be the table underlying the spatial entity itself, unless you know what you're doing, and if you're using ArcSDE they almost certainly should NOT be, because of versioning) that can be linked to the entity via a one-to-one attribute join. If the entity is a spatial entity and direct create, update and delete operation are required then the spatial editing bundle should be used.
  - When the source entity is a non-spatial entity this type of editor can be either a related non-spatial table or a table that directly represents the entity. Since there is no concept of a non-spatial entity specific table in Weave that corresponds to the spatial table (or layer) for a spatial entity practically there is no difference between a related non-spatial table and a table that directly represents the entity.

There is also the option to create an entity specific edit panel that combines entity and related edits, generally representing a single entity table and one or more related tables.

### Configuration

#### Client

##### Simple grid editing

There are a number of client panels available, the first and simplest is the "simple edit grid", this provides a panel similar to the data grid, in that it shows a grid of records and a list box to allow switching between different "editors".

From there the user can select and edit a single row.

The only config options for the simple grid panel are `entity`, to restrict the panel to only showing "related" editors associated with the indicated entity.

`type` to restrict the panel to only showing editors of the specified type ("table" and "related" are the available options, more on those later). Finally `edit` is also available to restrict the display to a single specific edit config, or to refine the edits included when one of the other options is used.

If none of `entity`, `type` or `edit` are specified then the result would be the same as if `type` were set to "related".

#### Example simple editor grid showing only 'table' type editors

```
<view id="editor.panel.simplegrid">
  <label>Tables</label>
  <location>center</location>
  <type>table</type>
</view>
```

#### Example simple editor grid showing only 'related' type editors

```
<view id="editor.panel.simplegrid">
  <label>Tables</label>
  <location>center</location>
  <type>related</type>
</view>
```

#### Example simple editor grid showing only editors associated with the 'property' entity

```
<view id="editor.panel.simplegrid">
  <label>Property</label>
  <location>center</location>
  <entity>property</entity>
</view>
```

#### Example simple editor grid showing only the 'edit.property' editor

```
<view id="editor.panel.simplegrid">
  <label>Edit</label>
  <location>center</location>
  <edit>edit.property</edit>
</view>
```

#### Example simple editor grid showing only a subset of available editors

```
<view id="editor.panel.simplegrid">
  <label>Edit</label>
  <location>center</location>
  <edit>edit.property.owners</edit>
  <edit>edit.property.occupiers</edit>
  <edit>edit.property.notices</edit>
</view>
```



### Example combined editor based on a single edit configuration, that allows viewing and editing records on the fly

```
<view id="editor.panel.combinedgrid">
  <label>Property Owners</label>
  <location>east</location>
  <edit>edit.property.owners</edit>
  <labelAlign>left</labelAlign>
  <enableCreate>>true</enableCreate>
  <enableDelete>>true</enableDelete>
</view>
```

#### Simple entity editing

In addition to the simple editor grid there is also the editor panel, this panel is intended to support editing of an entity editor in a form (along with additional related editors)

### Example entity editor form

```
<view id="editor.panel">
  <label>Edit</label>
  <location>center</location>
  <edit>edit.property</edit>
</view>
```

In its simplest form above the view presents a form for a single edit (the panel can only be setup to edit a single entity). In addition to this you can also specify additional edit tag to add more forms or grids to the view, e.g.

### Extended entity editor form

```
<view id="editor.panel">
  <label>Edit</label>
  <location>center</location>
  <edit>edit.property</edit>
  <edit>edit.property_owners</edit>
</view>
```

In this example the edit.property\_owners editor is a related editor config for the property entity (the edit.property editor is an entity editor config for properties), and so the editor panel will display the property form then the property owners grid.

## Server

To have something to edit a server editor config must be created, and currently there are three different types of edits that you can configure (as previously mentioned).

#### Direct Table Editing

The most basic edit you can create is a 'table' edit, which directly represents a single table where the users can edit all rows in the table.

### Example table editor config

```
<?xml version="1.0" encoding="UTF-8"?>

<config      xmlns="urn:com.cohga.server.config#1.0"
            xmlns:editor="urn:com.cohga.server.editor#1.0">

    <editor:table id="lutAction">
        <label>Action</label>
        <datasource>ora_prod</datasource>
        <table>lutAction</table>
        <key>Action_ID</key>
    </editor:table>

</config>
```

In this example we've created a table editor and the user will be able to select it from the list using the name 'Action'. The table the editor will be updating is the 'lutAction' table in the 'ora-prod' database, and the table must have a column called 'Action\_ID' that contains the unique keys for the rows in the table.

Note that you don't need to specify the key column if the table contains a single primary key column, Weave will determine that and use that column to identify rows.

By default the editor in the client will display all columns for the user, and allow them to edit all columns (except the key column). You can specify which columns you want to expose by adding 'parameter' tags to the editor config (in a similar manner to the spatial editing configuration, and the attribute search configuration), this also allows you to explicitly specify how you want Weave to expose the column to the user, for example if you need to set a list of values that the user can enter for a particular column since Weave will just create a basic representation of the columns if they're not explicitly specified.

Note, you can create a default configuration for a table from the osgi console using the 'editor' command, e.g.

```
osgi> editor conf ora_prod lutAction
```

will produce something like:

#### lutAction example conf

```
<editor:table id="lutAction">
    <label>LutAction</label>
    <datasource>ora_prod</datasource>
    <table>lutAction</table>
    <parameter id="action_id" label="Action" column="Action_ID"
dataType="integer" autogenerated="true" hidden="true"/>
    <parameter id="enablingprovision_id" label="
EnablingProvision" column="EnablingProvision_ID" dataType="integer"/>
    <parameter id="order" label="Order" column="Order" dataType="
integer"/>
    <parameter id="action" label="Action" column="Action"/>
    <parameter id="landstatus_id" label="LandStatus" column="
LandStatus_ID" dataType="integer"/>
    <parameter id="casetype_id" label="CaseType" column="
CaseType_ID" dataType="integer"/>
    <parameter id="interesttype_id" label="InterestType" column="
InterestType_ID" dataType="integer"/>
    <parameter id="comment" label="Comment" column="Comment"/>
    <parameter id="shorttitle" label="ShortTitle" column="
ShortTitle"/>
    <parameter id="options" label="Options">
        <parameter id="primaryplaceofresidence" label="
PrimaryPlaceOfResidence" column="PrimaryPlaceOfResidence" dataType="
```

```

boolean" controlType="check-box"/>
        <parameter id="wholeproperty" label="WholeProperty"
column="WholeProperty" dataType="boolean" controlType="check-box"/>
        <parameter id="nottobelicensed" label="
NotToBeLicensed" column="NotToBeLicensed" dataType="boolean"
controlType="check-box"/>
        <parameter id="permanentworks" label="PermanentWorks"
column="PermanentWorks" dataType="boolean" controlType="check-box"/>
    </parameter>
</editor:table>

```

which you can cut and paste into your config file and update to reflect the changes you'd like, for example changing the landstatus\_id so that it selects the value from a list.

### Looking up land status

```

    <parameter id="landstatus_id" label="Land Status" column="
LandStatus_ID" dataType="integer">
        <dataset>landstatus</dataset>
        <valuecolumn>landstatus_id</valuecolumn>
        <labelcolumn>landstatus</labelcolumn>
        <pagesize>0</pagesize>
    </parameter>

```

It's also possible to have Weave generate the parameters and just override those that you want to change in the config, rather than having to specify each and every parameter if you just want to change one, by setting 'parameters' to true in the config.

```

<editor:table id="lutAction">
    <label>LutAction</label>
    <datasource>ora_prod</datasource>
    <table>lutAction</table>
    <parameters>true</parameters>
    <parameter id="landstatus_id" label="Land Status" column="
LandStatus_ID" dataType="integer">
        <dataset>landstatus</dataset>
        <valuecolumn>landstatus_id</valuecolumn>
        <labelcolumn>landstatus</labelcolumn>
        <pagesize>0</pagesize>
    </parameter>
</editor:table>

```

#### Entity Table Editing

You can also define an editor that represents a non-spatial entity

### Sample Entity Editor Config

```

<?xml version="1.0" encoding="UTF-8"?>

<config          xmlns="urn:com.cohga.server.config#1.0"

```

```

xmlns:editor="urn:com.cohga.server.editor#1.0">

<editor:entity id="tblDocument">
  <entity>documents</entity>
  <label>Document</label>
  <datasource>ora_prod</datasource>
  <table>tblDocuments</table>
  <key>Document_ID</key>
</editor:entity>

</config>

```

Here we're setting up an entity editing configuration, you can still use the 'editor conf' command to generate the base configuration, but then you need to change the type from 'table' to 'entity' and set the 'entity' attribute to the name of the entity that the table rows represent.

#### Related Table Editing

The final type of editor is the related table editor, this allows you to edit rows in a table that related in a one-to-many relationship to an entity.

#### Related Table Editor

```

<?xml version="1.0" encoding="UTF-8"?>

<config      xmlns="urn:com.cohga.server.config#1.0"
  xmlns:editor="urn:com.cohga.server.editor#1.0">

  <editor:related id="tblIncidents">
    <entity>documents</entity>
    <label>Incident</label>
    <datasource>ora_prod</datasource>
    <table>tblIncidents</table>
    <key>Incident_ID</key>
    <entitykey>Document_ID</entitykey>
  </editor:related>

</config>

```

This is a simple related table editor. In this example we're editing Incident records that are related to the Document entity. We need to define two keys for a related table, one that uniquely identifies the row to be edited, the 'key' attribute, which is optional and will be set if there's a single column primary key on the table, but there's also the 'entitykey' column, this column should contain the values that link the records in this table to the rows in the parent entity table.

**Note** that if you manually specify the parameters for a related editor you must specify the value for the entitykey parameter as the formula entitykey(), plus you should also set it as readonly on update and hidden.

This will ensure that when the record is written the value for the column that links to the parent entity will have the correct value, i.e. the key value of the selected parent entity.

Once the user selects an entity they will be able to edit all related records that contain the same 'entitykey' value as the selected entities key value.

#### Autogenerated values

Some parameters are autogenerated by the database and for some databases nothing needs to be configured for this to happen (aside from specifying that the column is autogenerated), but some databases require additional information in some circumstances, for example, Oracle and Postgres use something they call Sequences to generate a sequence of values, which is where the value to be inserted into the database column comes from. If you're trying to create a record that contains a column with a value that's supposed to be generated based on a sequence then you need to specify the name of the sequence that the database should use when creating the record, which can be done by setting the `sequence` attribute for the parameter.

### Setting a sequence for a parameter

```
<parameter name="id" column="ASSET_ID" autogenerated="true" sequence="
ASSET_ID_SEQ" hidden="true"/>
```

#### Enabling and Disabling create, update and/or delete

As of Weave 2.6.5, or version 1.23.5 of the `com.cohga.server.editor` plugin, it's possible to declare that a particular edit configuration supports, or does not support, create, update and/or delete operations.

By default, all three operations are allowed for a given edit configuration, but you can disable create, update and/or delete for a specific edit by setting the `creatable`, `updatable`, or `deletable` flags (you can also use `create`, `update` and `delete` which are equivalent settings).

### Disabling creation and deletion for a given edit config

```
<?xml version="1.0" encoding="UTF-8"?>

<config      xmlns="urn:com.cohga.server.config#1.0"
            xmlns:editor="urn:com.cohga.server.editor#1.0">

    <editor:related id="tblIncidents">
        <entity>documents</entity>
        <label>Incident</label>
        <datasource>ora_prod</datasource>
        <table>tblIncidents</table>
        <key>Incident_ID</key>
        <entitykey>Document_ID</entitykey>
        <creatable>>false</creatable>
        <deletable>>false</deletable>
    </editor:related>

</config>
```

## Weave Console

The Weave console (or osgi console) is a command prompt available for interacting with the Weave server.

The console is available directly, if Weave is started using `startup.cmd` (on Windows) or `startup.sh` (on Linux), or it is available via `telnet` if Weave is started as a background service (i.e. a Windows service or a Linux daemon). If you have the Admin UI installed it is also available via the Console shortcut.

OSGi console when Weave is manually started on Windows (`startup.cmd`)

```

C:\Windows\system32\cmd.exe
com.cohga.server.dms.provider.db: BundleEvent STARTED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCustomizer@6e16ffd]
Added script source from bundleentry://216.fwk1890191192/client/css/actions/icon.s.css <com.cohga.client.weave.ClientResourceManager> [weave-dynamic-source-builder-001]
com.cohga.client.dms.upload: BundleEvent STARTED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCustomizer@6e16ffd]
com.cohga.server.dms.upload.core: ServiceEvent REGISTERED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCustomizer@6e16ffd]
com.cohga.server.dms.upload.core: BundleEvent STARTED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCustomizer@6e16ffd]
com.cohga.server.dms.storage.db: ServiceEvent REGISTERED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCustomizer@6e16ffd]
com.cohga.server.dms.storage.db: BundleEvent STARTED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCustomizer@6e16ffd]
com.cohga.server.geocode.arcgis: ServiceEvent REGISTERED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCustomizer@6e16ffd]
com.cohga.server.geocode.arcgis: BundleEvent STARTED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCustomizer@6e16ffd]
org.eclipse.osgi: BundleEvent STARTED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCustomizer@6e16ffd]
org.eclipse.osgi: FrameworkEvent STARTED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCustomizer@6e16ffd]
org.eclipse.osgi: FrameworkEvent STARTLEVEL CHANGED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCustomizer@6e16ffd]
2014-10-28 16:37:48.228:INFO:weave:Framework started
2014-10-28 16:37:48.229:INFO:weave:Weave Bridge Servlet initialized
2014-10-28 16:37:49.826:INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:8080

osgi> _

```

OSGi console via Telnet when running as a service on Windows (telnet localhost 9001)

```

Telnet localhost

osgi> _

```

OSGi console when manually started on Linux (startup.sh)

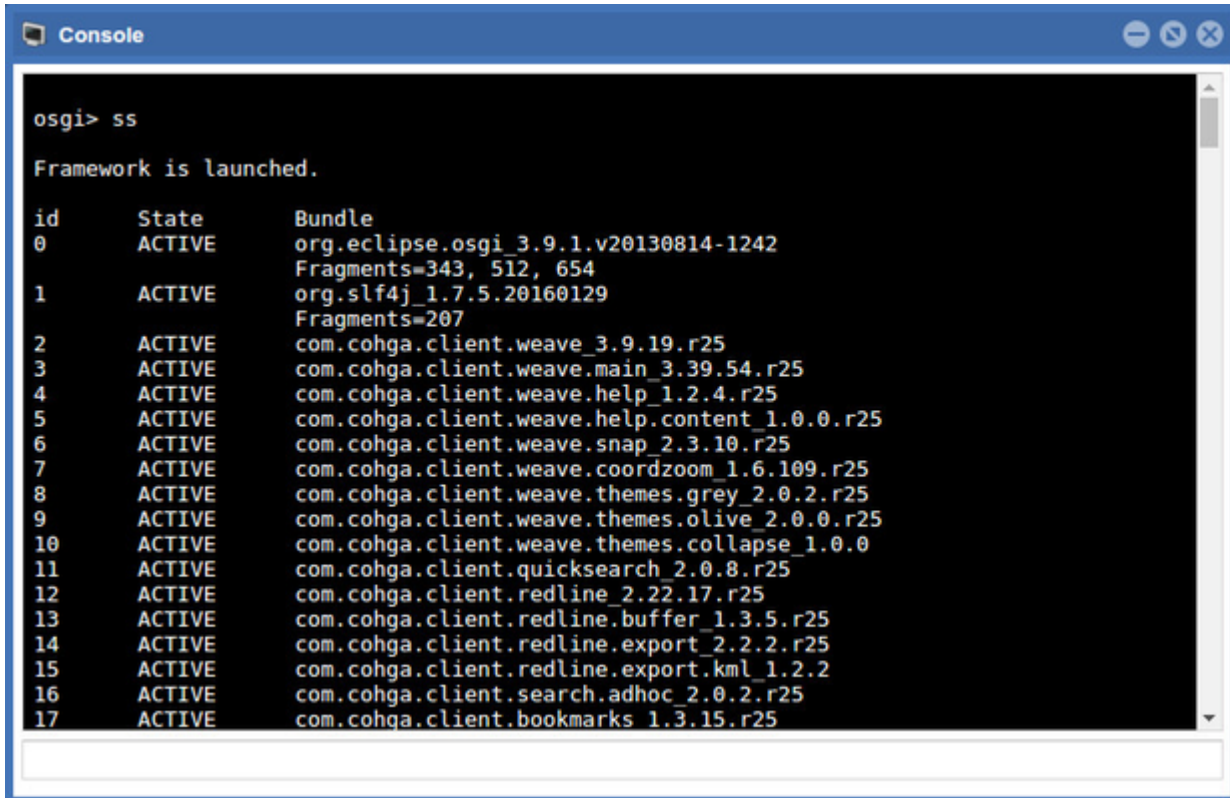
```

File Edit View Search Terminal Help
TrackerCustomizer@6b2efcaa]
Added content in com.sencha.extjs to /sencha/extjs <com.cohga.weave.client.content.ContentManager> [Framework
Event Dispatcher]
com.sencha.extjs: BundleEvent STARTED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCust
omizer@6b2efcaa]
Added content in com.sencha.extjs.desktop to /sencha/extjs/desktop <com.cohga.weave.client.content.ContentMan
ager> [Framework Event Dispatcher]
com.sencha.extjs.desktop: BundleEvent STARTED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTra
ckerCustomizer@6b2efcaa]
Added content in com.sencha.extjs.menu to /sencha/extjs/menu <com.cohga.weave.client.content.ContentManager>
[Framework Event Dispatcher]
com.sencha.extjs.menu: BundleEvent STARTED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTracke
rCustomizer@6b2efcaa]
org.eclipse.osgi: BundleEvent STARTED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerCust
omizer@6b2efcaa]
org.eclipse.osgi: FrameworkEvent STARTED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.LogTrackerC
ustomizer@6b2efcaa]
org.eclipse.osgi: FrameworkEvent STARTLEVEL CHANGED <com.cohga.slf4j.LogTrackerCustomizer> [com.cohga.slf4j.
LogTrackerCustomizer@6b2efcaa]
2014-10-28 16:46:15.920:INFO:weave:Framework started
2014-10-28 16:46:15.920:INFO:weave:Weave Bridge Servlet initialized
2014-10-28 16:46:15.960:INFO:oejs.AbstractConnector:Started SelectChannelConnector@0.0.0.0:8080

osgi>

```

Console accessed via the Admin UI



```

Console
osgi> ss

Framework is launched.

id      State      Bundle
0       ACTIVE    org.eclipse.osgi_3.9.1.v20130814-1242
         Fragments=343, 512, 654
1       ACTIVE    org.slf4j_1.7.5.20160129
         Fragments=207
2       ACTIVE    com.cohga.client.weave_3.9.19.r25
3       ACTIVE    com.cohga.client.weave.main_3.39.54.r25
4       ACTIVE    com.cohga.client.weave.help_1.2.4.r25
5       ACTIVE    com.cohga.client.weave.help.content_1.0.0.r25
6       ACTIVE    com.cohga.client.weave.snap_2.3.10.r25
7       ACTIVE    com.cohga.client.weave.coordzoom_1.6.109.r25
8       ACTIVE    com.cohga.client.weave.themes.grey_2.0.2.r25
9       ACTIVE    com.cohga.client.weave.themes.olive_2.0.0.r25
10      ACTIVE    com.cohga.client.weave.themes.collapse_1.0.0
11      ACTIVE    com.cohga.client.quicksearch_2.0.8.r25
12      ACTIVE    com.cohga.client.redline_2.22.17.r25
13      ACTIVE    com.cohga.client.redline.buffer_1.3.5.r25
14      ACTIVE    com.cohga.client.redline.export_2.2.2.r25
15      ACTIVE    com.cohga.client.redline.export.kml_1.2.2
16      ACTIVE    com.cohga.client.search.adhoc_2.0.2.r25
17      ACTIVE    com.cohga.client.bookmarks_1.3.15.r25

```

⊗ Providing telnet access presents security implications as no username or password is required to connect so telnet may not be enabled by default. If it is not enabled then it will need to be manually enabled in order to provide access.

## Summary of useful console commands

### Useful console commands

Command	Description
start	Starts a bundle given an ID or symbolic name
stop	Stops a bundle given an ID or symbolic name

<b>install</b>	Adds a bundle given a URL for the current instance
<b>uninstall</b>	Removes a bundle given a URL for the current instance
<b>update</b>	Updates a bundle given a URL for the current instance
<b>active</b>	Lists all active bundles in the current instance
<b>headers</b>	List the headers for a bundle given an ID or symbolic name
<b>ss</b>	Lists a short status of all the bundles registered in the current instance
<b>services &lt;filter&gt;</b>	Lists services given the proper filter
<b>diag</b>	Runs diagnostics on a bundle given an ID or symbolic name
<b>dump</b>	Generates a Weave support dump
<b>ustorage</b>	Work with information persisted for users
<b>storage</b>	Work with information persisted for the system

There are many other OSGi commands available. To get a list of all the commands, just type 'help' at the OSGi prompt ( e.g. osgi>help ) in the Weave console.

Note that starting and stopping bundles (the `start` and `stop` commands above) can also be done via the Admin UI using the Bundles shortcut.

Id ↑	Name	Version	Status
0	org.eclipse.osgi	3.9.1.v20130814-1242	ACTIVE
1	org.sif4j	1.7.5.20160129	ACTIVE
2	com.cohga.client.weave	3.9.19.r25	ACTIVE
3	com.cohga.client.weave.main	3.39.54.r25	ACTIVE
4	com.cohga.client.weave.help	1.2.4.r25	ACTIVE
5	com.cohga.client.weave.help.content	1.0.0.r25	ACTIVE
6	com.cohga.client.weave.snap	2.3.10.r25	ACTIVE
7	com.cohga.client.weave.coordzoom	1.6.109.r25	ACTIVE
8	com.cohga.client.weave.themes.grey	2.0.2.r25	ACTIVE
9	com.cohga.client.weave.themes.olive	2.0.0.r25	ACTIVE
10	com.cohga.client.weave.themes.collapse	1.0.0	RESOLVED
11	com.cohga.client.quicksearch	2.0.8.r25	ACTIVE
12	com.cohga.client.redline	2.22.17.r25	ACTIVE
13	com.cohga.client.redline.buffer	1.3.5.r25	ACTIVE
14	com.cohga.client.redline.export	2.2.2.r25	ACTIVE

At the bottom of the UI, there are checkboxes for **INSTALLED**, **RESOLVED**, **STARTING**, and **ACTIVE**. A **Start the selected bundle** button is also present.

### Enabling telnet access

If Weave was installed as a service in Windows without telnet access and you want to enable it then you need to edit the `service\conf\wrapper.conf` file.

Telnet access is enabled by setting the `osgi.console` property to the port number that you want to provide telnet access on.

The `wrapper.conf` file may already contain a line, that is commented out with a `#`, that can be edited to provide telnet access, e.g.

```
# set the console port for telneting into Weave.
# wrapper.java.additional.20=-Dosgi.console=<port>
```



Removing the # at the start of the second line and replacing <port> with the port number will enable telnet access on that port.

Alternatively the line may already be enabled, in which case you should already be able to telnet to Weave on the port number listed by <port>

```
wrapper.java.additional.20=-Dosgi.console=9001
```

 The default value for the telnet port is 9001, but can be changed during installation.

If the `wrapper.conf` file does not contain the `osgi.console` line at all then it should be added, paying special attention to the number just before the equals sign, to make sure it is one more than the highest value used in any other additional settings. e.g. 20 in the above examples.

After this line has been added or changed in `wrapper.conf` you will need to restart Weave to enable the telnet access. Once that is done you should be able to telnet directly into Weave and access the osgi console. e.g.


```
C:> telnet localhost 9001
osgi>
```

When you have finished accessing the osgi console use the `disconnect` command to close the connection.

```
osgi> disconnect
Disconnect from console? (y/n; default=y) y

Connection to host lost.

C:>
```

 Do not use `close` when connected to the osgi console via telnet as this command will shutdown Weave!

 You may not have telnet installed on the server or client PC. The following link contains information on how to [Install Telnet Client](#)

## UTF-8 and Internationalization

By default Windows may not correctly display UTF-8 characters in the OSGi console.

To fix this you need to change the default font for the Command Prompt to a true type font (for example Consolas or Lucinda Console) and then execute `chcp 65001`

This should be done before running `startup.cmd`, from the same command prompt window where you just entered the `chcp` command.

Alternatively you could edit the `startup.cmd` file to include the `chcp 65001` command and set the command prompt as the default (this has been done in recent releases of Weave).

## Weave Metadata Commands

### The metadata commands

The Weave OSGi console provides three commands that provide information about things that Weave interacts with, namely data sources, spatial engines and map engines.

By using the `dbmd` (DataBase MetaData), `memd` (Map Engine MetaData) and `spmd` (SPatial engine MetaData) commands you can see what databases, map engines and spatial engines Weave is connected to and find out information about the items within those connections.

When used by themselves these commands list the registered datasources, map engines and spatial engines.

```
osgi> dbmd
Datasource
```

```

-----
datasource.dbf
datasource.manningham
datasource.sqlserver
dms.documents
storage
system.datasource

```

```

osgi> memd
Map Engine
-----
mapengine.edit
mapengine.overview
mapengine.raster
mapengine.raster.ags10
mapengine.raster.arcims
mapengine.selection
mapengine.selectionOld
mapengine.upload
mapengine.vector

```

```

osgi> spmd
Spatial Engine
-----
spatialengine.edit.shapefile
spatialengine.manningham
spatialengine.oracle
spatialengine.postgis
spatialengine.sqlserver
spatialengine.wfs

```

Each metadata command also has sub-commands that perform specific actions, and the latest detailed syntax of each command can be seen by typing 'help' at the OSGi prompt.

### dbmd - The database metadata command

The `dbmd` command has the following sub-commands:

- `list` - List all the available datasources (the default if no sub-command is given) or list the tables that are contained within a `datasource`
- `desc` - Describes a table contained within a `datasource`
- `conf` - Generate a data definition XML template based on a table.

### memd - The map engine metadata command

The `memd` command has the following sub-commands:

- `list` - Lists the available map engines
- `desc` - Describes a map engine
- `toc` - Generates a ToC model XML template based on a map engine
- `reset` - Clears metadata for a map engine (only available in 2.5.26).

### spmd - The spatial engine metadata command

The `spmd` command has the following sub-commands:

- `list` - List all the available datasources (the default if no sub-command is given) or list the spatial layers that are contained within a spatial engine
- `desc` - Describe a spatial layer contained within a spatial engine
- `crs` - List the coordinate reference systems of each spatial layer or a single spatial layer in a spatial engine
- `count` - Counts the number of rows in a spatial layer
- `conf` - Produces sample configuration XML files for a spatial engine.

## Server Status

The Weave server status page is available at the following location:

`http://<server>:<port>/weave/server/status`

It includes a range of different details about the internals of the running server.

The status page provides links to various monitoring pages at the top and at the bottom provides 2 additional links to export the data contained in the page to an xml file.

One link exports the current page content and the other exports the detail from all of the monitoring pages.

### ArcGIS Connections

For connections to ArcGIS Server that have pooling enabled this page shows details about the connections Weave has created.

Column	Description
Map Engine	The id of the ArcGIS Server map engine
Created	Total number of connection Weave has opened to ArcGIS Server
Destroyed	Total number of connections Weave has closed to ArcGIS Server
Existing	How many connection Weave currently has open to ArcGIS Server
Max. Existing	The maximum number of connections Weave has open to ArcGIS Server at one time
Borrowed	Total number of times a connection has been taken from the pool to be used
Returned	Total number of times a connection has been returned to the pool after use
Invalidated	Total number of times a connection has been invalidated, usually because of an error when being used
In Use	How many connections are currently borrowed from the pool
Max. In Use	The maximum number of connections borrowed from a pool at one time

### User Details

Gives a brief overview of a users usage of the server.

Column	Description
User	The userid of the user
Request Count	The total number of requests the users browser has made to the server
Last Request	The date and time of the last request made by the users browser

### Request Details

Gives a brief overview of the different requests that the users browsers have made to the server

Column	Description
Request	The id of the request
Count	The total number of requests of this type the users browser has made to the server

### Pending Requests

Gives a overview of what requests are currently being performed by the server

Column	Description

User	The userid of the user who made the request
Request	The id of the request
IP	The IP address of the users browser
Age	The number of milliseconds since the request was submitted from the users browser

### Hourly Response

Provides an historical overview of how many requests are made and how long responses are taking broken down by how many hours ago the request was made.

This is particularly useful to determine if the server is currently experiencing performance issues, as it provides clear evidence if recent responses are taking longer than responses from previous hours.

Each column in the table, except for the first, provides a count of the number of requests and the average response time in milliseconds, for a one hour period, starting from the hour immediately preceding the time that the page was generated and covering one hour intervals for 24 hours after that.

### Response History

Provides an historical overview of how many requests are made and how long responses are taking covering periods ranging from the previous minute up to the previous week (up to a maximum of 50,000 requests).

This is particularly useful to determine if the server is currently experiencing performance issues, as it provides clear evidence if recent responses are taking longer than usual.

Each column in the table, except for the first, provides a count of the number of requests and the average response time in milliseconds, for a particular period starting from when the request was made. That is, the first data column provides the count and average time for the previous minute, the second data column provides the count and average time for the previous 5 minutes, up to the last data column with provides the count and average time for the previous 7 days. But there's a maximum of 50,000 data points that are collected, so if you exceed that number within a period less than 7 days then the columns after that time will only show a total of 50,000 and the average for those.

### Quantile Distribution

Provides a breakdown of response times, grouping the data so that there is an equal (or as equal as possible) request count in each group.

In this view the total number of requests are divided by 4 and the minimum and maximum response times gathered, starting from fastest to slowest, until enough samples (i.e the total divided by 4) are collected to determine the the minimum and maximum response times for that group. That is, based on the number of requests, the first row will display the minimum and maximum response times for the fastest 25% of the requests, the second row for the next 25% of the requests, up to the last row that shows the response time range for the slowest 25% of the requests. So that in this grouping the data is broken down so that the requests count between collection are equal.

This lists the minimum and maximum response times in a way that can provide a guide to the distribution of response times.

### Equal Distribution

Provides a breakdown of response times, grouping data so that there is equal (or as equal as possible) time period between each groups minimum and maximum response times.

In this view the minimum and maximum response times are broken up into four equal length periods, then the number of requests that took that long to respond are totaled for each period, showing the Count, and the number of requests that fall into that group out of the total requests made are shown in the Percent. So that in this grouping the data is broken down so that the time periods between collection are equal.

This lists the minimum and maximum response times in a way that can provide a guide to the distribution of response times.

### Hourly Requests

Provides an overview of how many requests are made during each hour of the day. This is a running total that accumulates as long as the server is running. Each row in the table represents a 1 hour period starting from midnight.

### Weekly Requests

Provides an overview of how many requests are made during each hour of the day for each day of the week. This is a running total that accumulates as long as the server is running. Each row in the table represents a 1 hour period starting from midnight for each day of the week.

### Response Distribution

Provides a quick overview of the distribution of response times.

Column	Description
--------	-------------

Response Time	The range, in seconds, that the row covers
Count	The total number of requests that occurred within the response time range
Percent	The total number of requests that have been handled within the maximum response range
	A "graphical" representation of the distribution

Note that the Percent column is cumulative as you go down the rows. That is the second row shows the percent of total requests that were handled in 10 seconds or under, not the percentage of total requests that were handled within 5 to 10 seconds, and the third row shows the percent of requests handled in 15 seconds or under, not those handled between 10 and 15 seconds.

### Timing Summary

This is a detailed breakdown of the timing for a whole range of different components. The first column provides a Label that indicates what is being timed, which can include things like the response times when communicating with other systems, the response times for a particular user, browser or IP address. Most of these are internal measurements and not intended for general consumption.

A detailed explanation of what the various columns in the table represent can be found at <http://jamonapi.sourceforge.net/>

### Bundle Status

Provides an overview of the state of the OSGi bundles that make up the Weave instance.

### Config Items

Provides an overview of the configuration items that the Weave instance currently has registered. Note that this provides no guarantee that the configured item is correct and working, just that the configuration (usually from a config.xml file) has been read and registered with the system.

### Indexes

Provides an overview and some tools to operate on the Quick Search indexes that have been configured for Weave. This is the same stuff provided by the OSGi command prompt with its `is`, `ix` and `ib` commands.

### JDBC Connections

For connections to databases that have pooling enabled this page shows details about the connections Weave has created.

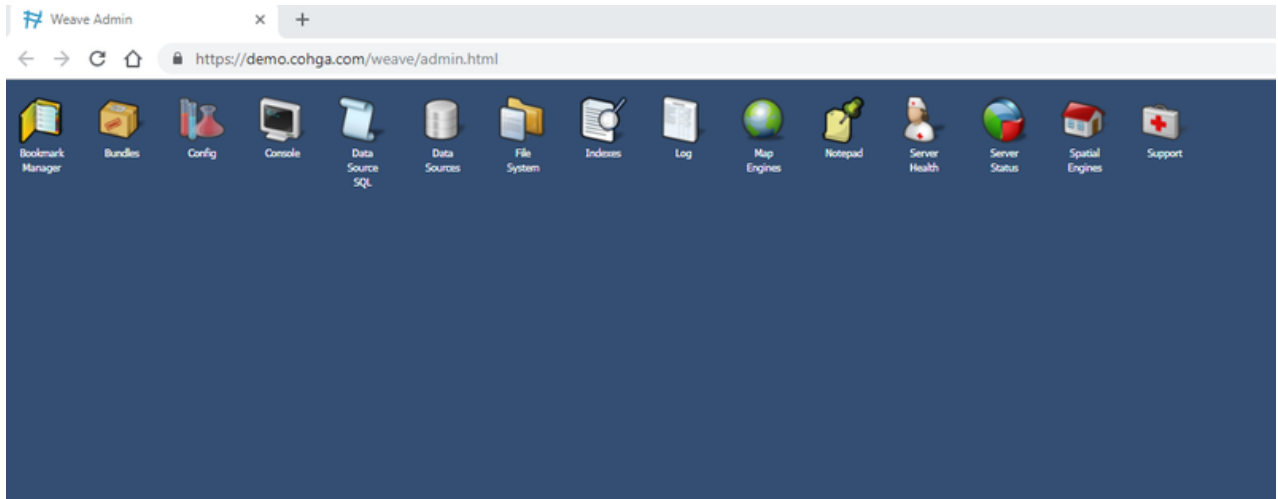
Column	Description
Datasource	The id of the datasource
Created	Total number of connection Weave has opened to the database
Destroyed	Total number of connections Weave has closed to the database
Existing	How many connection Weave currently has open to the database
Max. Existing	The maximum number connections Weave has open to the database at one time
Borrowed	Total number of times a connection has been taken from the pool to be used
Returned	Total number of times a connection has been returned to the pool after use
In Use	How many connections are currently borrowed from the pool
Max. In Use	The maximum number of connection borrowed from a pool at one time

## Administration Tool

### Accessing the Administration Tool

The Weave *Administration Tool* is a web-based application which allows you, as the Weave Administrator, to view and edit Weave configuration files. The *Administration Tool* is accessed via this URL:

```
<hostname or IP address>:<port>/weave/admin.html
(where Weave itself is run at: http://<hostname or IP address>/weave/main.html)
```



### The Tools

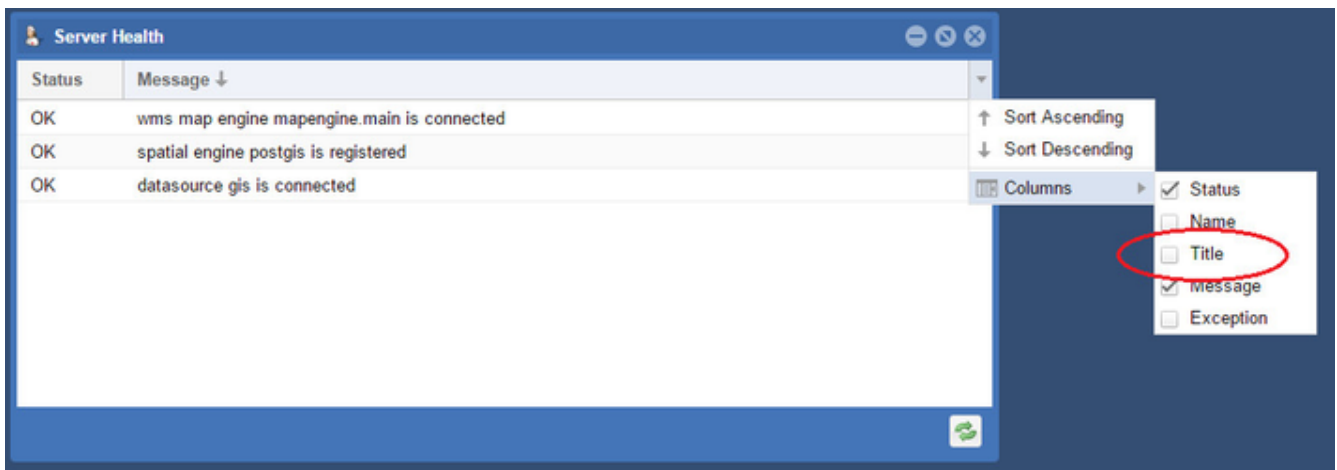
The tools available through the *Administration Tool* are described in the following sections:

- [Bookmark Manager Tool](#)
- [Bundles Tool](#)
- [Config Tool](#)
- [Console Tool](#)
- [Data Source SQL Tool](#)
- [Data Sources Tool](#)
- [File System Tool](#)
- [Indexes Tool](#)
- [Log Tool](#)
- [Map Engines Tool](#)
- [Notepad Tool](#)
- [Server Health Tool](#)
- [Server Status Tool](#)
- [Spatial Engines Tool](#)
- [Support Tool](#)

Most of the tools available through the *Administration Tool* are also available through the [Weave Console](#). However, unlike the Weave Console, the *Administration Tool* is always available and not reliant on you having access to the server Weave is installed on (the Weave server). If you are able to run a telnet session to the Weave server, you will get the same access as what you get via the *Console* window in the *Administration Tool*.

### Customising the Admin Tool Windows

In some of the windows in the Admin Tool you can add/remove columns by clicking on the column heading and selecting/deselecting column names from the list displayed as shown below.



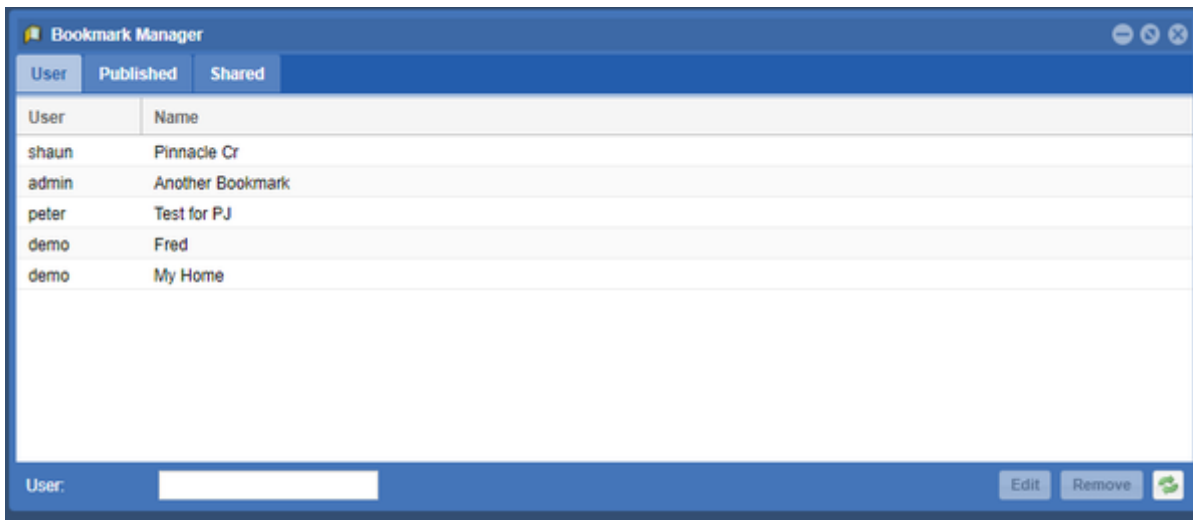
### Bookmark Manager Tool

This tool is available for versions of Weave from 2.5.29.

The *Bookmark Manager Tool* opens a window that allows you to manage Bookmarks. This tool provides the same functions as the [Client Action Book Manager](#) provides through the button on the Weave client toolbar.

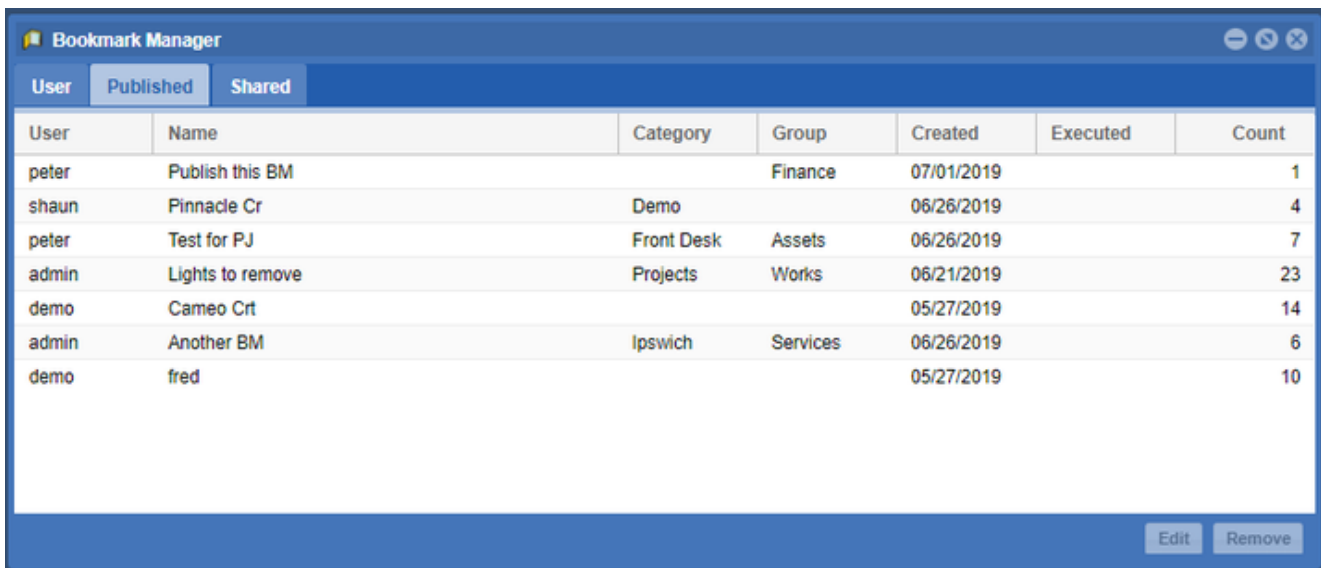
*User*

This displays the list of Bookmarks created by the *user* you are currently logged in as.



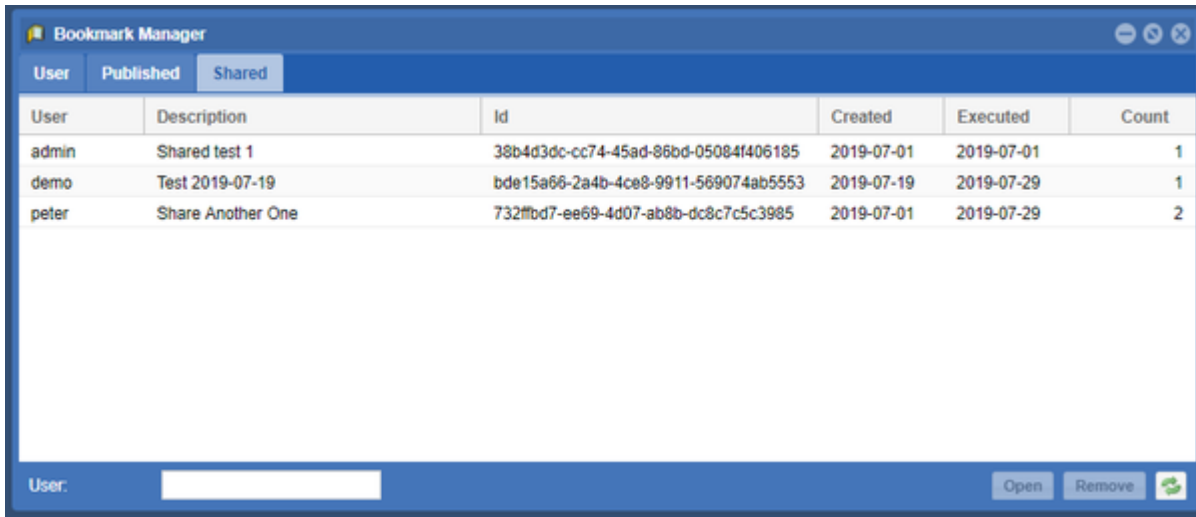
*Published*

This displays the list of Bookmark that have been *published*. These Bookmarks can be edited or deleted. This tool also displays the number of times the Bookmark has been used and when it was last executed.



*Shared*

This displays the list of Bookmark that have been *shared*. These Bookmarks can be opened or deleted. This tool also displays the number of times the Bookmark has been used and when it was last executed.



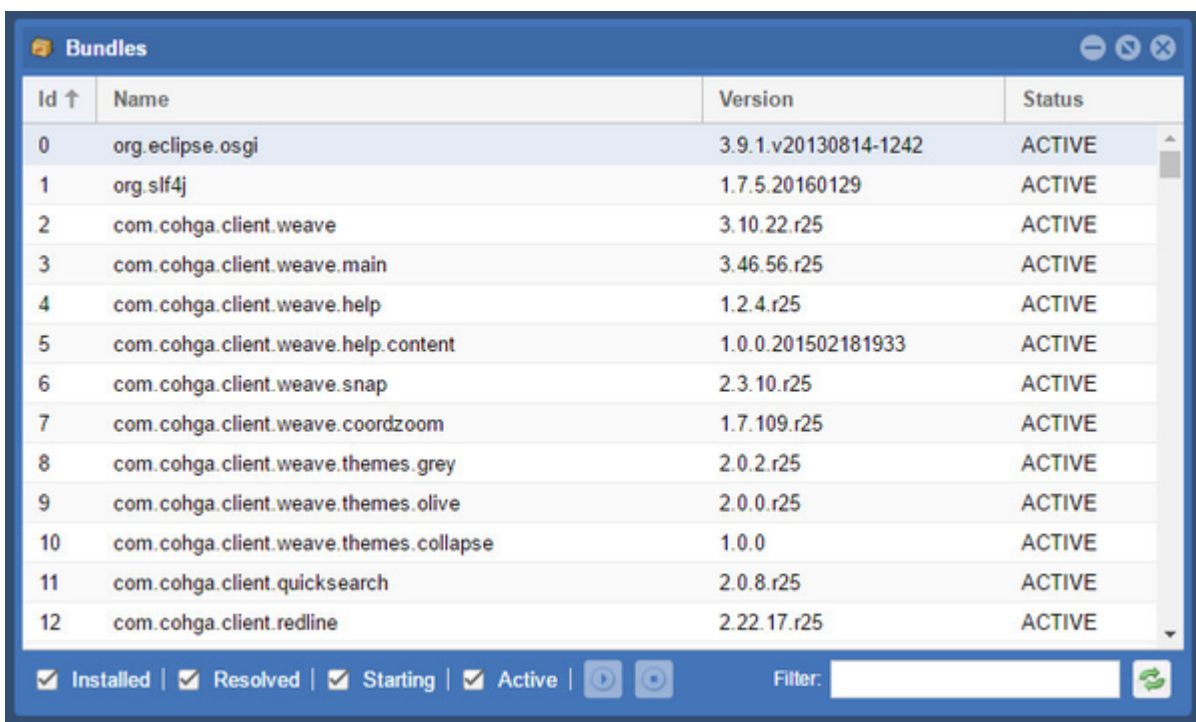
The screenshot shows a window titled "Bookmark Manager" with three tabs: "User", "Published", and "Shared". The "User" tab is active, displaying a table with the following data:

User	Description	Id	Created	Executed	Count
admin	Shared test 1	38b4d3dc-cc74-45ad-86bd-05084f406185	2019-07-01	2019-07-01	1
demo	Test 2019-07-19	bde15a66-2a4b-4ce8-9911-569074ab5553	2019-07-19	2019-07-29	1
peter	Share Another One	732fbd7-ee69-4d07-ab8b-dc8c7c5c3985	2019-07-01	2019-07-29	2

At the bottom of the window, there is a "User:" label followed by a text input field, and three buttons: "Open", "Remove", and a refresh icon.

## Bundles Tool

The *Bundles Tool* lists all bundles in the Weave installation, where bundles are small packets of code that together make up the Weave system. This tool gives the version of the bundle and the status of the bundle. It is useful for managing bundles if you are running customised bundles or have been provided a replacement bundle.



The screenshot shows a window titled "Bundles" with a table listing various bundles. The table has columns for "Id", "Name", "Version", and "Status".

Id ↑	Name	Version	Status
0	org.eclipse.osgi	3.9.1.v20130814-1242	ACTIVE
1	org.slf4j	1.7.5.20160129	ACTIVE
2	com.cohga.client.weave	3.10.22.r25	ACTIVE
3	com.cohga.client.weave.main	3.46.56.r25	ACTIVE
4	com.cohga.client.weave.help	1.2.4.r25	ACTIVE
5	com.cohga.client.weave.help.content	1.0.0.201502181933	ACTIVE
6	com.cohga.client.weave.snap	2.3.10.r25	ACTIVE
7	com.cohga.client.weave.coordzoom	1.7.109.r25	ACTIVE
8	com.cohga.client.weave.themes.grey	2.0.2.r25	ACTIVE
9	com.cohga.client.weave.themes.olive	2.0.0.r25	ACTIVE
10	com.cohga.client.weave.themes.collapse	1.0.0	ACTIVE
11	com.cohga.client.quicksearch	2.0.8.r25	ACTIVE
12	com.cohga.client.redline	2.22.17.r25	ACTIVE

At the bottom of the window, there are several filter buttons: "Installed", "Resolved", "Starting", and "Active", each with a checked checkbox. There are also two circular buttons (Play and Stop) and a "Filter:" text box with a refresh icon.

The *Installed*, *Resolved*, *Starting* and *Active* buttons on the bottom of the window can be used to restrict the display to bundles of a specific *Status*. You can also narrow down the list of bundles displayed by using the *Filter* box.

Use the *Refresh* button to get the latest list of bundles if any have been added to the Weave installation folder.

The *Play* and *Stop* buttons are used to start or stop the selected bundle.

There are six possible states for a bundle as shown in the *Status* column:

- *Installed* means the bundle has been copied to the correct location (the `..\weave\platform\plugins` directory) and OSGi knows about it, but it hasn't been started yet, either because it hasn't been added to `config.ini` file or because there's something preventing it from starting (e.g. a missing dependency).
- *Resolved* means the bundle has been installed and has all it needs to be used, it is ready to be *started*.
- *Starting* means the bundle is in the process of getting up and running (and should be available soon).
- *Active* means the bundle has been resolved and started.
- *Fragment* bundles extend the functionality of other bundles and cannot be started.



- *Lazy bundles* are started automatically when/if they're needed.

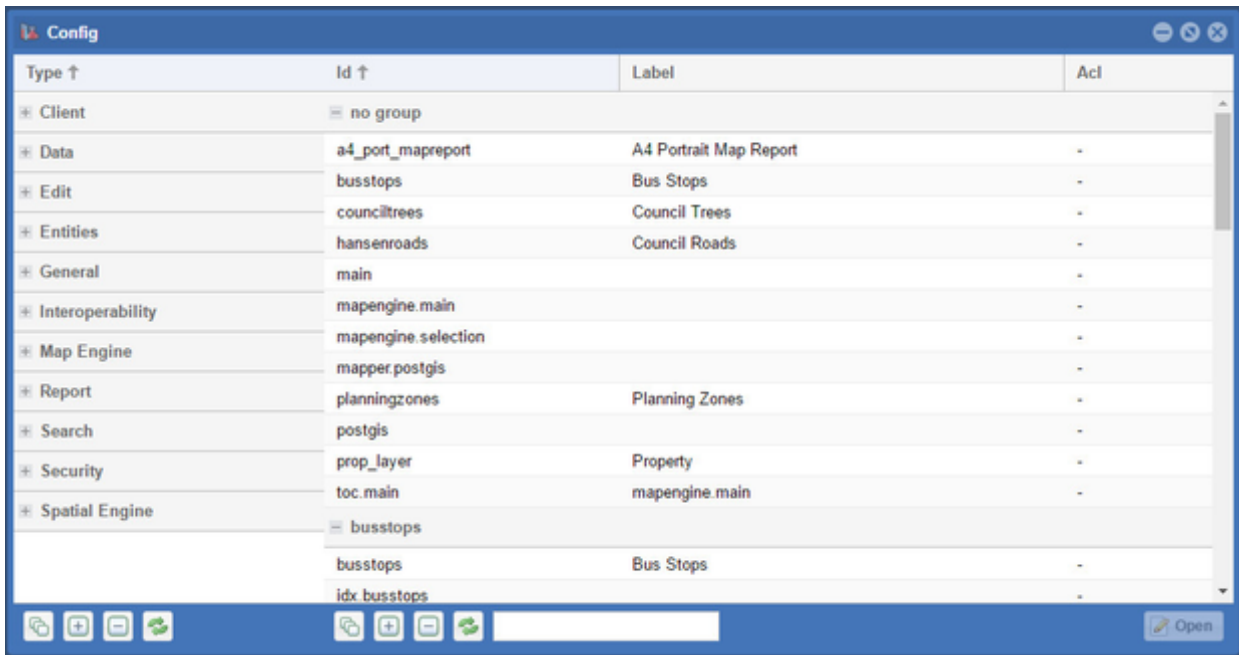
Bundles that are listed in the `config.ini` file (in the `..\weave\platform\configuration` folder) will start when Weave starts.

## Config Tool

The *Config Tool* opens a window that allows you to view and edit configuration items for Weave.

### Configuration Items List

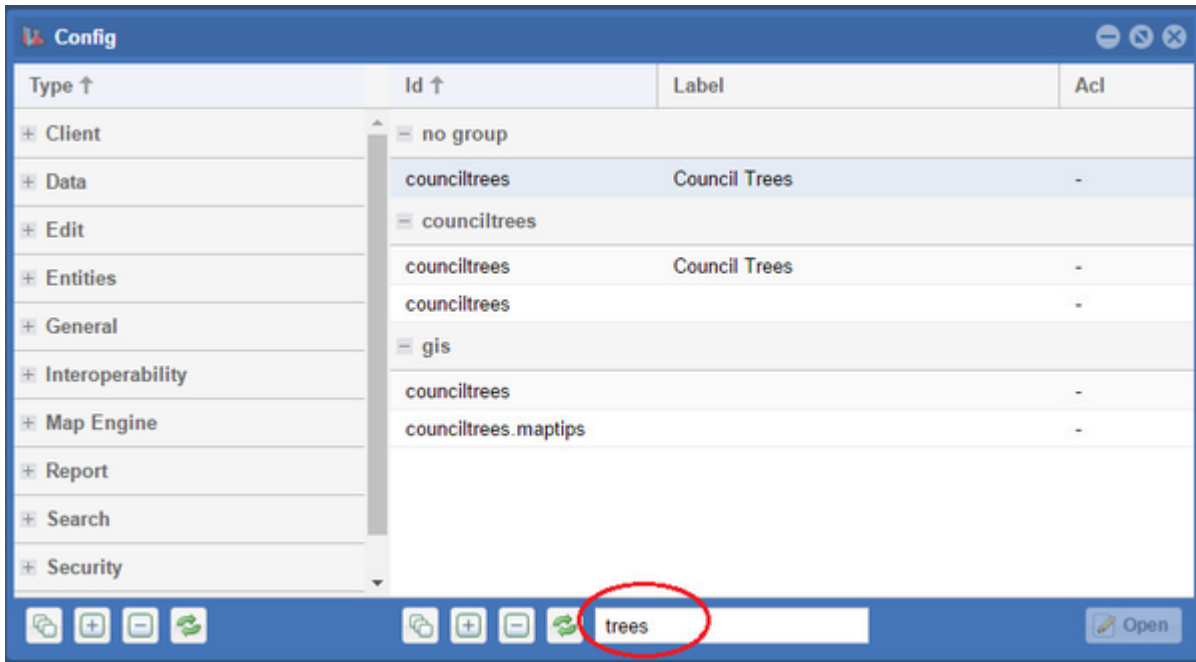
The configuration items are listed in groups, based on their function, and are listed in the left panel. The individual items are listed in the right panel.



In the left panel, use the *Group/Ungroup*, *Expand*, *Collapse* and *Refresh* buttons on the bottom of the window to change the appearance of the list.



The right panel shows the items from the group after you have expanded one/all groups and highlighted an item. The right panel has the same *Group/Ungroup*, *Expand*, *Collapse* and *Refresh* buttons to change the appearance of the list of items. It has the extra option of the *Filter* text box so you can restrict the items listed based on your filter text. This is useful where you have many reports, searches, entities, etc.



### Editing Configuration Items

This tool is available for versions of Weave from 2.5.20.

If you click on an item from the left side, in the *Type* column, the items that have been configured will be listed in the right side in the *Id* column. If you click on one of these items, the *Open* button in the bottom right becomes active and pressing it allows you to view the contents of the configuration file for that item.

Pressing the *Open* button or double clicking the item name will open the XML file that contains the details about that item. From here you'll be able to edit anything in that particular file, not just the item that you originally selected. The editor is simply a means to access the file. From here:

- The window that opens is a simple editor and it gives you the option of editing anything in that file.
- The following buttons help with your editing (they have shortcuts using Function Keys):
  - Comment - This button toggles the commenting around the current element and can be used as a quick way to disable /enable some XML.
  - Validate - Checks the XML syntax of the content, warning you if there are XML related issues with the content, which need to be fixed (saving also performs an XML validation before the content is written to disk, so you don't need to manually validate before saving the file).
  - Format - Adjusts the formatting in the content to tidy up the XML.
- The following Shortcuts are also available:
  - Fullscreen (F9) - Maximises the size of the editing window.
  - Find (Ctl-F) - Finds the occurrence of text within the editing window. You will first need to click into the editing window otherwise this will run a 'Find' in the browser window, and not the editing window.
  - Next (Ctl-G) - Finds the next occurrence of the search text in the editing window. You will first need to click into the editing window otherwise this will run a 'Next' in the browser window, and not the editing window.

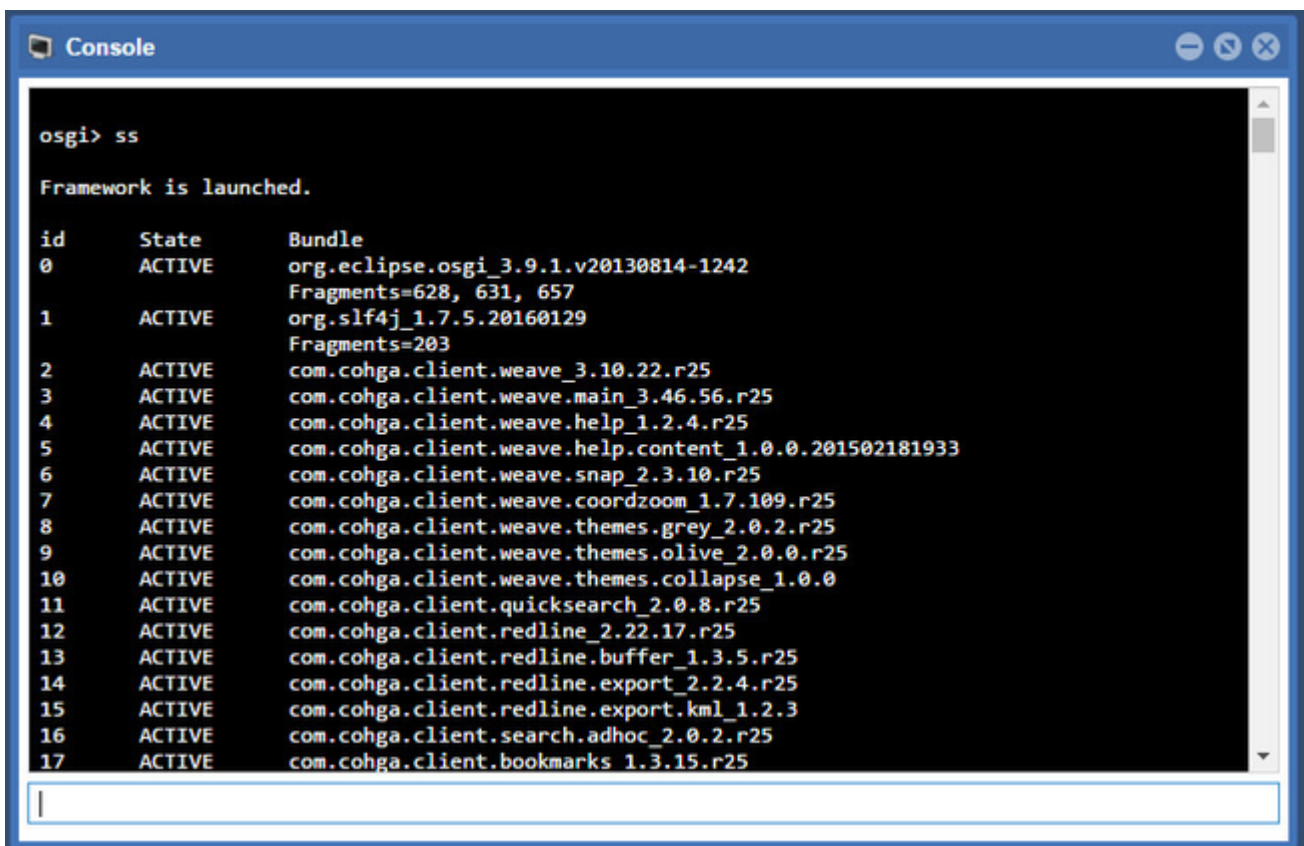


### Console Tool

The *Console Tool* opens the OSGi console and allows you to run OSGi and Weave commands. This is similar to the Weave server console window that is available when Weave is not running as a service but has been started by manually running the `c:` `\<weave_folder>\startup.cmd` file on the Weave server.

e.g. `ss`

to display the installed status of bundles (short status).



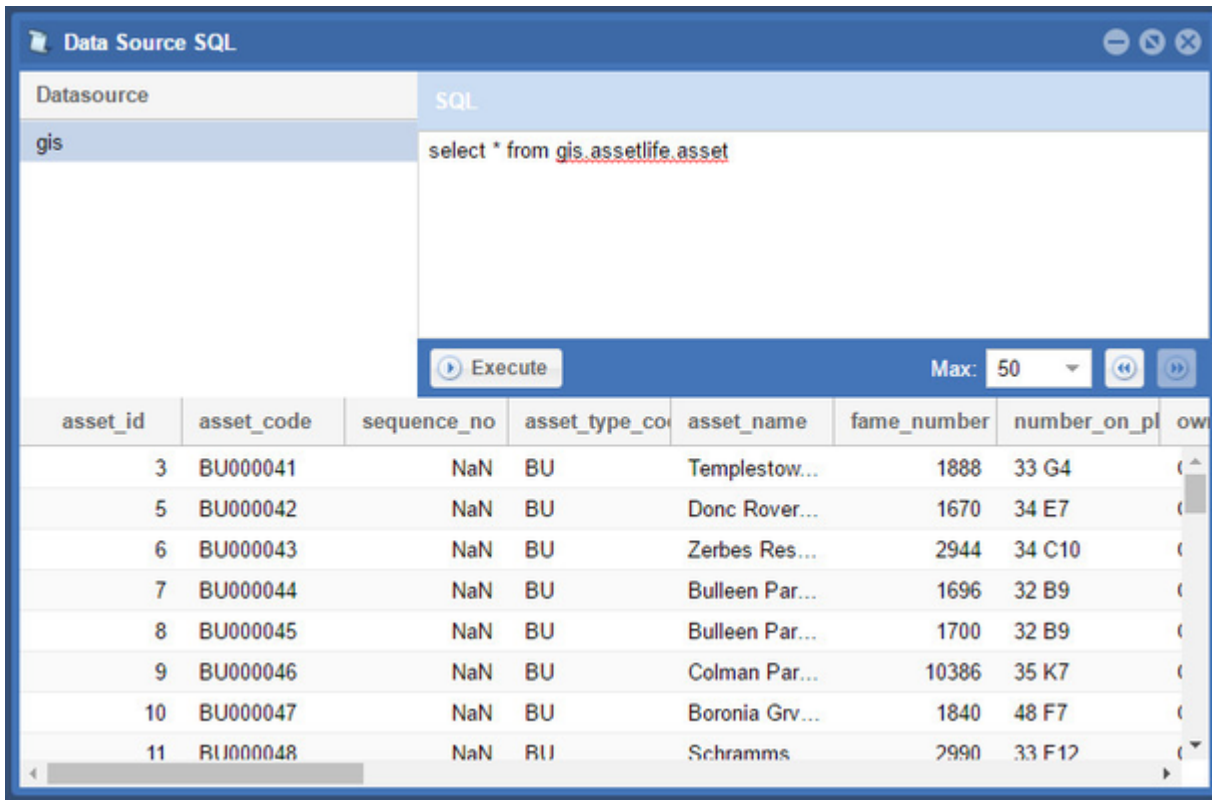
Refer to other pages in the Weave Administration Guide for details about the [Weave Console](#) and [Weave Metadata Commands](#).

### Data Source SQL Tool

The *Data Source SQL Tool* allows you to run SQL statements to query data in the datasource(s) (which are listed in the [Data Sources Tool](#)) to ensure that your query is correct and will return the correct results.

e.g. `select * from <database.schema.tablename>`

`select * from gis.assetlife.asset`



### Using the Data Source SQL Tool for Debugging

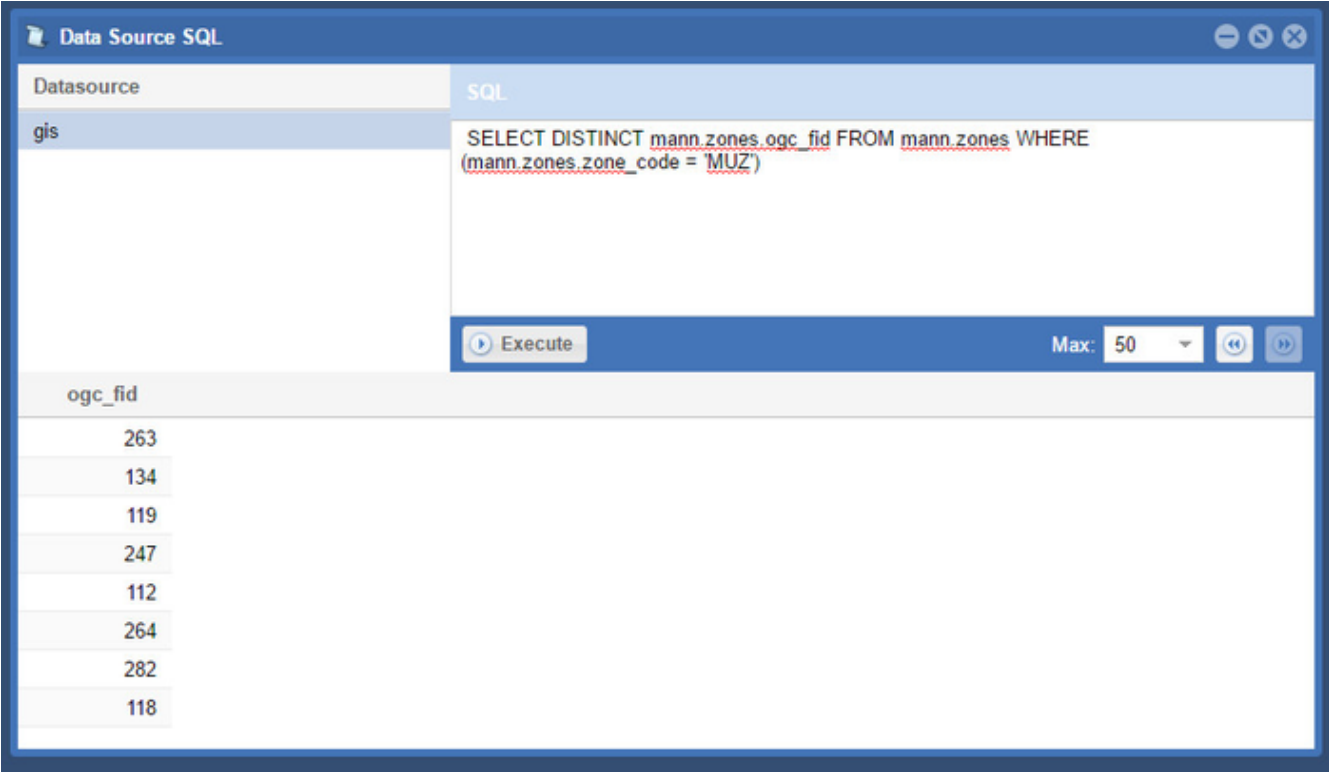
This tool is useful if you're not getting the results you were expecting in a search, data grid, etc. Use it to run the SQL command that is being sent to Weave (and has been constructed from the content of your XML files) and then check the results of this query.

To get the SQL that is being sent to Weave, open your Weave log file (e.g. `C:\weave\logs\weave.log`), search for "NativeSQL" in the file and copy the SQL command that follows this search term (remember to search from the bottom of the log file and search for the "previous" occurrence if you're looking for the last instance of the term "NativeSQL"). And enter the values for parameters if they do not appear in the log file. An example of this is shown below:

From `weave.log` file:

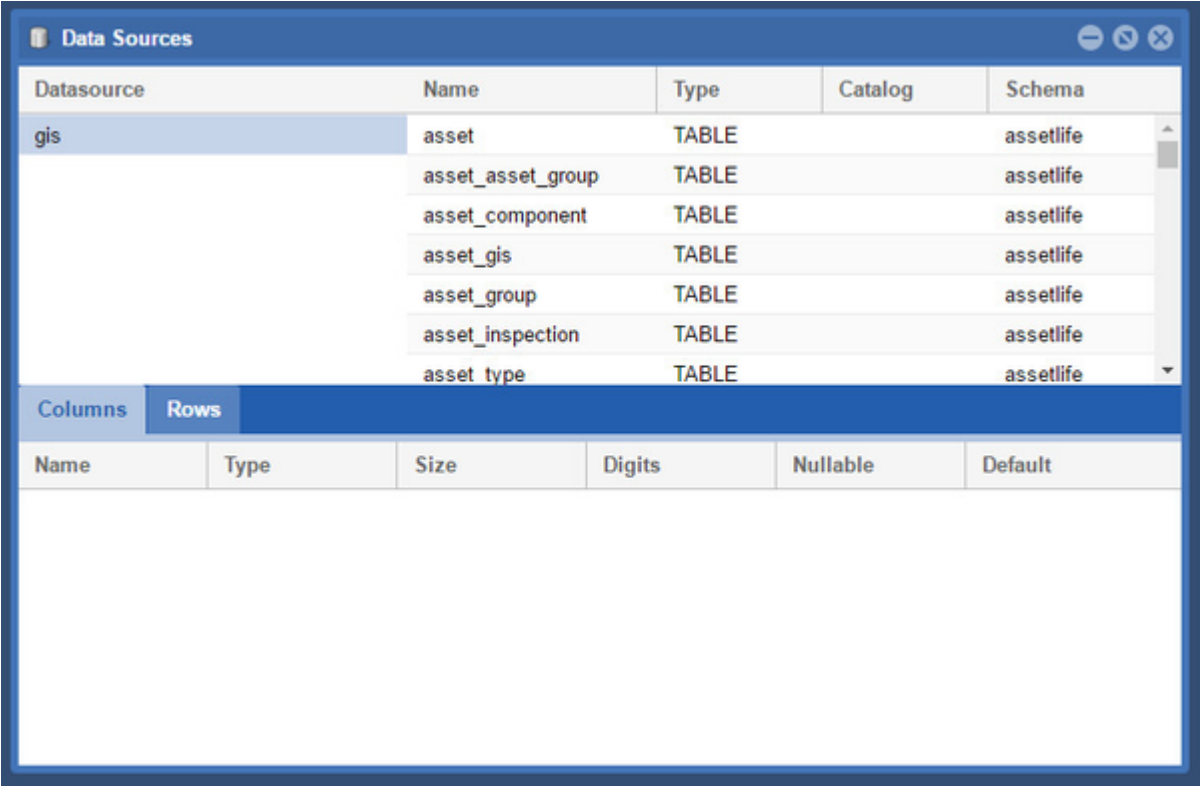
```
16:18:54,397 DEBUG [qtp228750133-114] com.cohga.server.search.
database.internal.attribute.AttributeSearchProvider "NativeSQL:
SELECT DISTINCT mann.zones.ogc_fid FROM mann.zones WHERE (mann.zones.
zone_code = ?)"
16:18:54,402 DEBUG [qtp228750133-114] com.cohga.server.search.
database.internal.attribute.BaseSearchProvider "Result=[263, 134,
119, 247, 112, 264, 282, 118]"
```

The same query applied in the *Data Source SQL Tool*:



Data Sources Tool

The *Data Sources Tool* lists the registered Data Sources. If you click on an item from the *Datasource* list in the left panel, the list of tables in that Data Source will be displayed in the right panel.



If you click on one item (i.e. table or view) from this list, the columns from that table will be displayed in the *Columns* section in the bottom panel of the window. If you click on the *Rows* tab, the first 10 records from that table will be displayed (as a sample of all the records in the table).

Datasource	Name	Type	Catalog	Schema
gis	wildfire	TABLE		mann
	yvw_hydrants	TABLE		mann
	yvw_sewer_pipe	TABLE		mann
	yvw_water_pipe	TABLE		mann
	yvwsewered	TABLE		mann
	zones	TABLE		mann
	Mineral_development	TABLE		public

ogc_fid	shape	zone_num	status	zone_code	mun_num	amd_num	am
1	[...]	6050	g	PPRZ	0	C0	
5	[...]	6050	g	PPRZ	0	C0	
6	[...]	6050	g	PPRZ	0	C0	
7	[...]	7160	g	PUZ1	0	C0	
8	[...]	706	n	R17	0	C0	

From Weave 2.5.22, it is possible to filter on *Name*, *Catalog* or *Schema* to reduce the entries returned in the right panel thus making it easier to find the entry you are interested in. The example below shows a filter on *Name*.

Datasource	Name	Catalog	Schema	Type
datasource.dbf	bushfireprone		mann	TABLE
gis	busroutes		mann	TABLE
	busstops		mann	TABLE

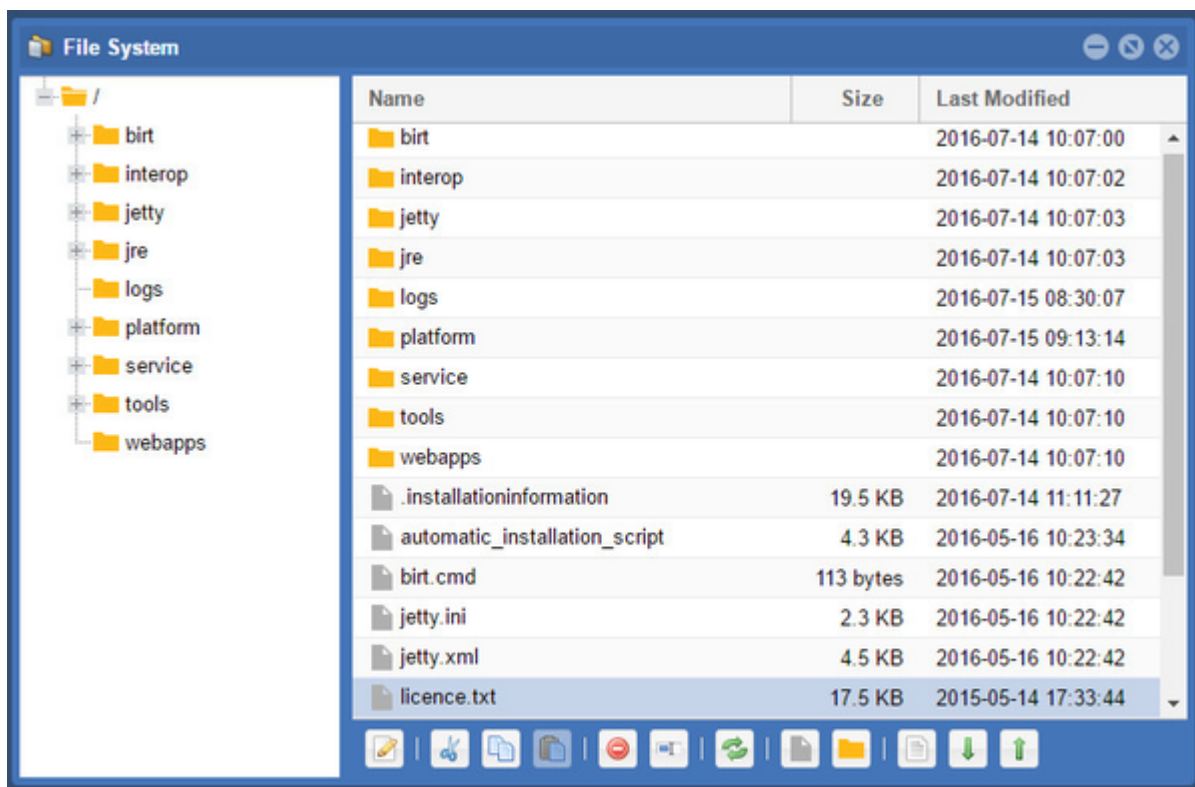
  

Filter:  Catalog:  Schema:

Name	Type	Size	Digits	Nullable	Default
------	------	------	--------	----------	---------

### File System Tool

The *File System Tool* opens a window that displays the current file system and allows you to create folders and files using the buttons along the bottom of the window. It uses the Weave installation folder as its starting point. The folders and files you see through this tool are what you would see if using Windows Explorer (or its equivalent for other operating systems) on the Weave server.

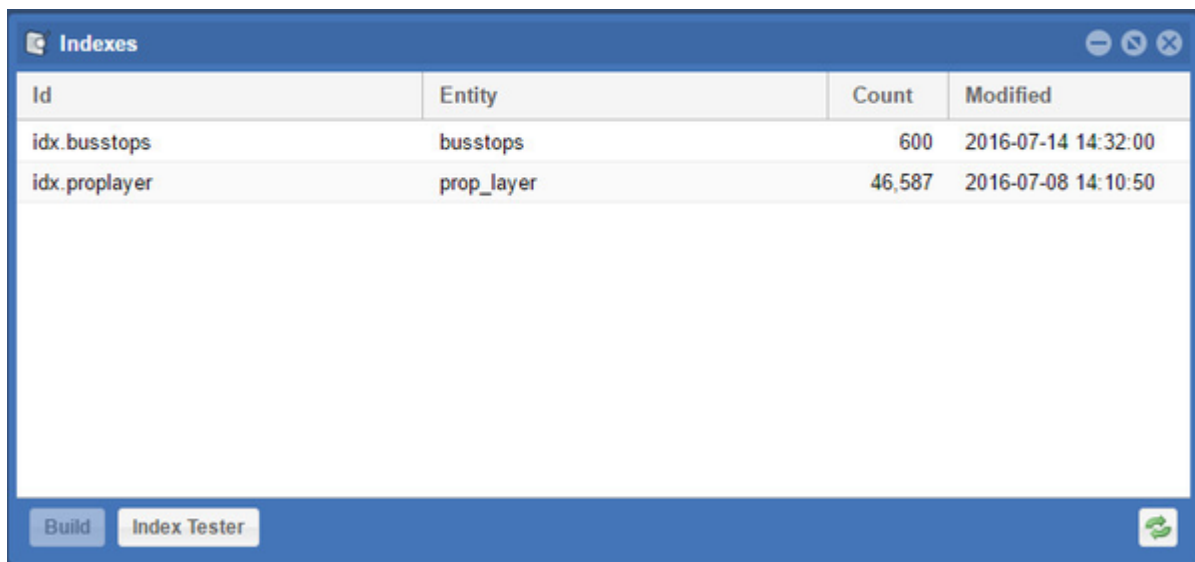


## Indexes Tool

This tool is available for versions of Weave from 2.5.20.

The *Indexes Tool* opens a window that shows you the status of indexes used in your Weave installation. It also allows you to build/rebuild and test these indexes.

The Indexes window displays the id of the index, the entity that it uses, the number of records in the index and the modification time of the index.



### Building an Index

To build or rebuild an index, select the index from the list and press the *Build* button. The index is built in the background and you will get a message stating that the index is in the process of being built. Press the *Refresh* button to get an update on the index build time and date.

### Testing an Index

To test an index, press the *Index Tester* button. The testing window is displayed and lets you enter search text or an entity id. You can also specify which index you want to search or it will, by default, use all indexes. Search results will begin to appear as you type in the *Text* or *Entity Id* text fields.

When testing your search through the *Index Tester* keep in mind that it is not exactly the same as the search that takes place through the *Quick Search* tool in the Weave client. When accessing the index through the *Quick Search* tool, the tool adds characters to the search string to ensure the best search result. So in order to mimic the *Quick Search* tool, you will need to add the wildcard (\*) or fuzzy (~) characters to your search string as or when needed. The examples below show the results from an index test with simple text followed by those that equate to what is done in the *Quick Search* tool.

1. Exact string search

Score	Index	Entity	Entity Id
-------	-------	--------	-----------

- Exact search returns no matching results.

2. Character substitution string search



The screenshot shows the 'Index Tester' window. The search criteria are: Text: 'cara\*', Entity Id: 'Entity key', Index: 'All Indexes', and Entity: 'All Indexes'. The results table is as follows:

Score	Index	Entity	Entity Id
1.00	idx.proplayer	prop_layer	32693
Plan No.: LP43090 pid: 32693 Address: 8 Carawatha Rd DONCASTER			
1.00	idx.proplayer	prop_layer	103806
Plan No.: PS545734 pid: 103806 Address: 2/6 Carawatha Rd DONCASTER			
1.00	idx.proplayer	prop_layer	32828
Plan No.: LP43090 pid: 32828 Address: 12 Carawatha Rd DONCASTER			


- The \* (asterisk) character is used to represent a number of characters. Adding a wildcard character suffix returns records that start with the search string

### 3. Approximate string match search

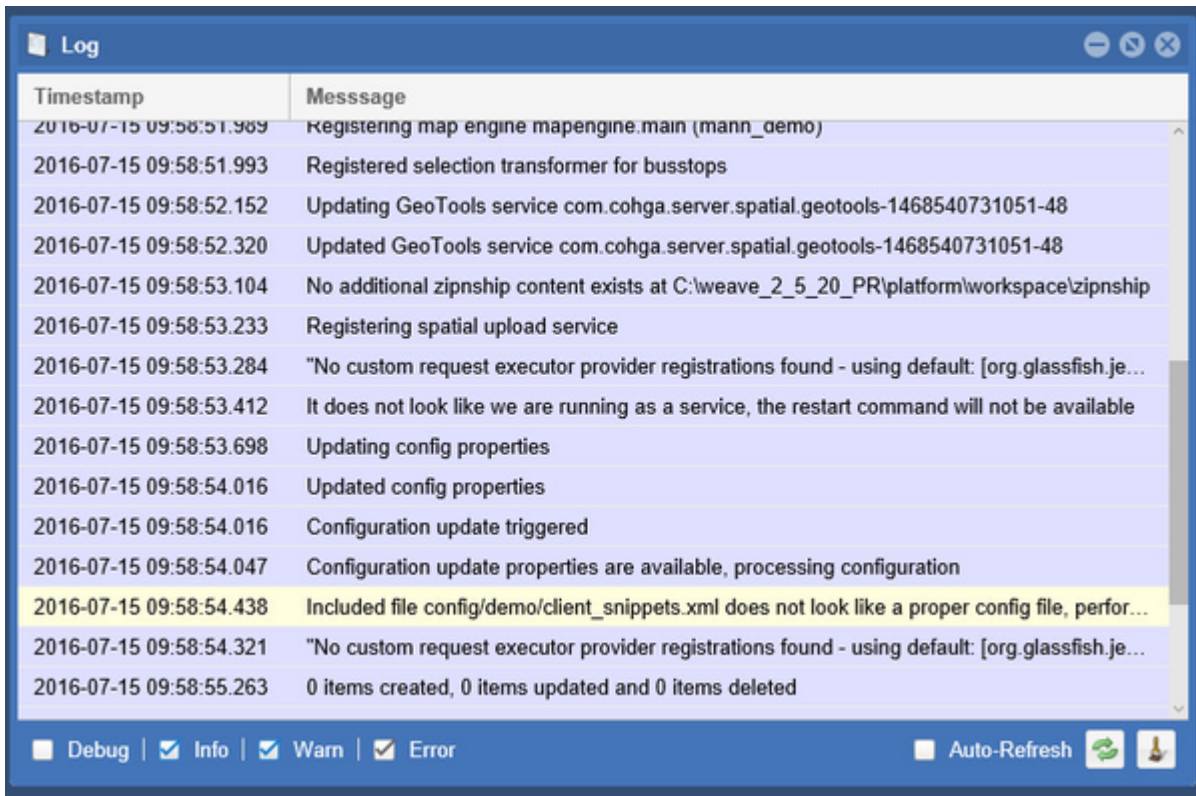
The screenshot shows the 'Index Tester' window. The search criteria are: Text: 'cara~|', Entity Id: 'Entity key', Index: 'All Indexes', and Entity: 'All Indexes'. The results table is as follows:

Score	Index	Entity	Entity Id
2.64	idx.proplayer	prop_layer	31632
Plan No.: LP70092 pid: 31632 Address: 4 Tara Crt DONCASTER			
2.64	idx.proplayer	prop_layer	31339
Plan No.: LP70092 pid: 31339 Address: 8 Tara Crt DONCASTER			
2.64	idx.proplayer	prop_layer	31432
Plan No.: LP70092 pid: 31432 Address: 7 Tara Crt DONCASTER			

- The ~ (tilde) character is used for approximate string matching (also known as fuzzy string searching). It is the technique of finding strings that match a pattern approximately (rather than exactly). It returns records that "sound like" the search string.

 For more details on creating and using indexes refer to this [page](#) in the Weave System Administrator Guides.

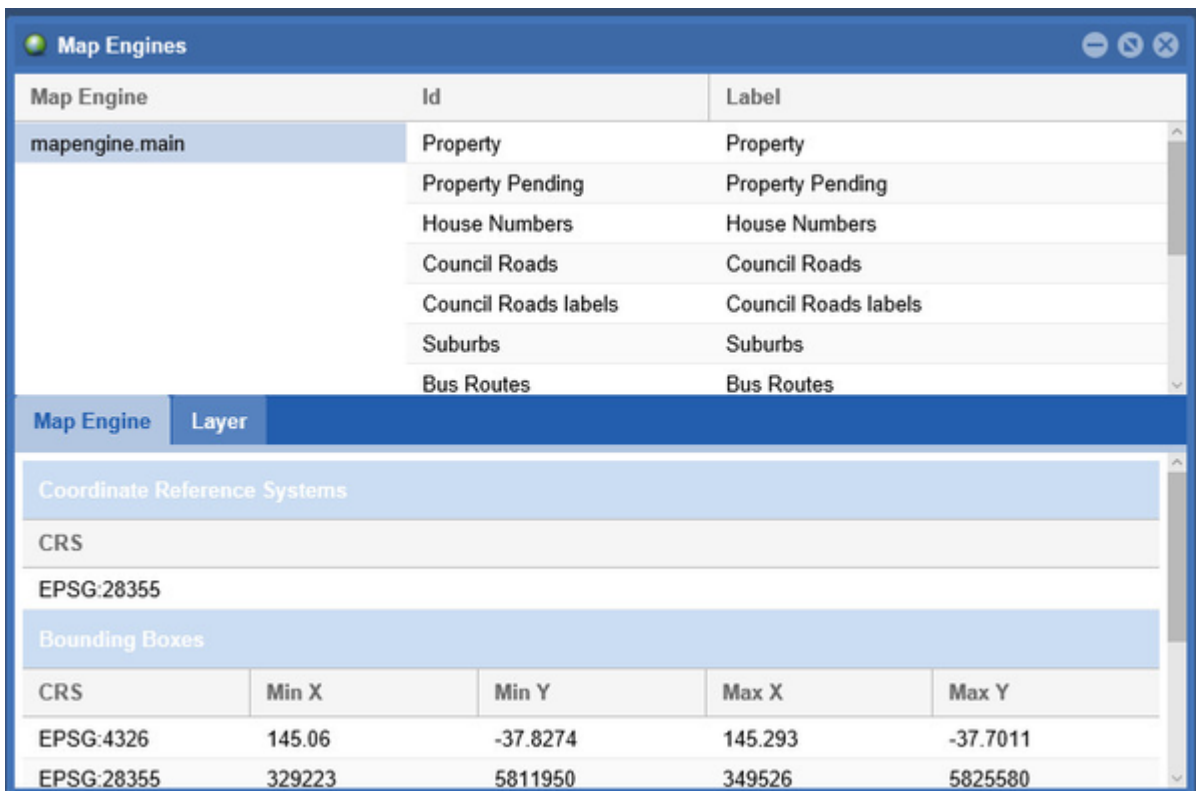
The *Log Tool* opens a window that displays Weave's logging messages. The four buttons along the bottom left of the window, *Debug*, *Info*, *Warn*, *Error*, control the level of error messages that are displayed in the window. These messages are also written to the Weave log files stored in the `c:\<weave_folder>\logs` folder.



### Map Engines Tool

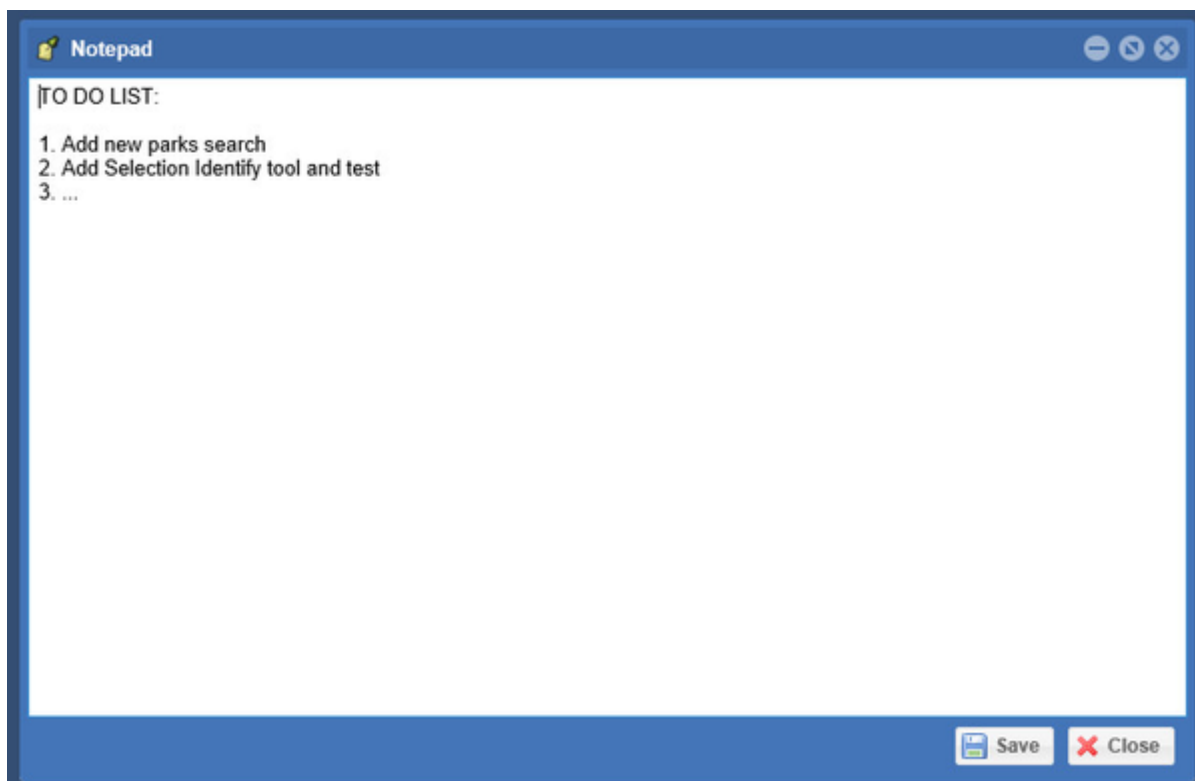
The *Map Engines Tool* lists the registered Map Engines sources. If you click on an item from the *Map Engine* list in the left panel, the list of layers in that Map Engine will be displayed in the right panel. The *Map Engine* tab on the bottom of the window gives you details of that engine; the *Layer* tab gives you details about the Map Layer that's highlighted in the top right panel.

This tool also has a `Reset` button (only available in 2.5.26). This tells Weave to clear any cached information it may have on a back-end map service and is useful when you change the service and need Weave to take note of the update.



### Notepad Tool

The *Notepad Tool* opens a text editor that can be used for keeping notes on your Weave configuration. This can be useful for sharing notes if you have more than one Weave System Administrator or for recording notes for yourself. The notes are stored on the Weave server.



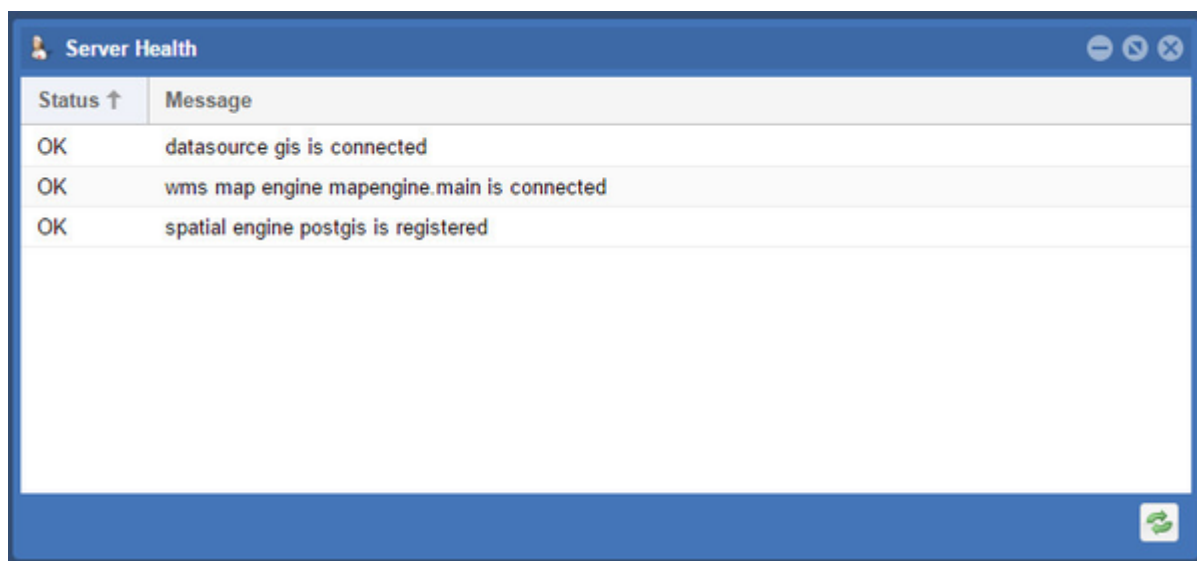
### Server Health Tool

This tool is available for versions of Weave from 2.5.20.

The *Server Health Tool* opens a window that shows you the status of the components of your Weave installation - Data Sources, Map Engines and Spatial Engines.

The Status column can be:

- OK = The item is working as expected.
- WARN = The item is not currently available, but it's not necessarily failed yet.
- CRITICAL = The item has failed and is not currently available.



### Sample Server Health Output

As an example of how the item status works imagine that we have an attribute search that requires a database connection before it can be used. During startup Weave reads the current configuration and processes both the attribute search and the database connection that it relies upon at the same time. The attribute search can not be said to be available until it knows that the database it's associated with is alive and accepting connections, but database connections can take a number of seconds to become available. During this period both the attribute search and the database connection will have a status of WARN, since we don't know if there's a problem establishing the database connection or if it's just taking time to establish the connection.

If the database connection is eventually successfully established then the new connection will be registered with Weave and the database's connection status will change to OK. Additionally, since the attribute search has been waiting on the database connection to be established before it can be registered, it now changes its status to OK and is also registered with the system.

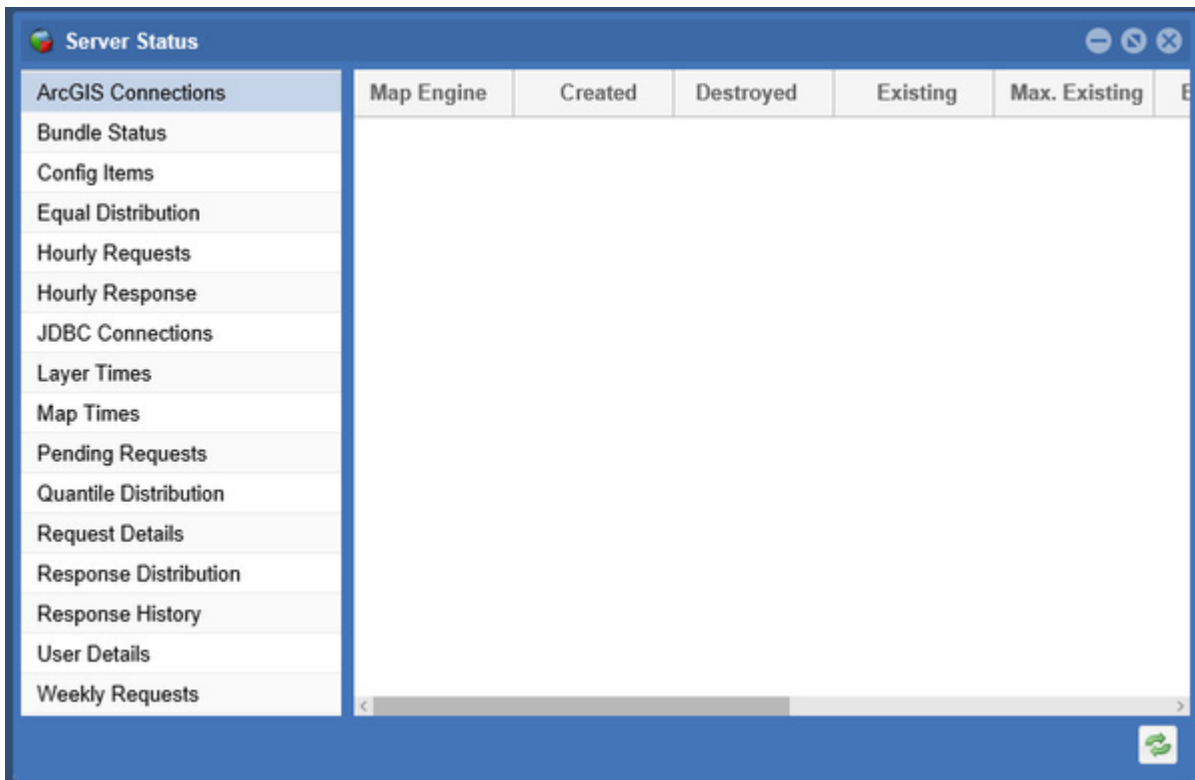
If however the database connection fails, either because the database is down or because the connection was not configured correctly, then the database's connection status will change to CRITICAL and the database connection will not be registered with Weave. At this stage the attribute search will still be listed as WARN, because it's still waiting on the database connection to be available, however the attribute search is setup with a timeout to ensure that if it waits too long for the database connection then it will change to CRITICAL.

If the problem with the database was a temporary one and the database comes back on-line, then the status of the database connection should change to OK and attribute search should detect the availability of the database connection that it's relying upon and also change to OK. And if the database goes down again, the database connection status will change to CRITICAL and it will be unregistered, the attribute report will detect the loss of its database connection and change to WARN until it times out and then will switch to CRITICAL too, unless the database comes back on-line within the timeout period.

## Server Status Tool

This tool is available for versions of Weave from 2.5.20.

The *Server Status Tool* displays details about the internals of the Weave server by a number of different measures. This tool is useful for monitoring the performance of your Weave installation.



There are a number of items that are reported and these are listed in the left panel of the window, click on one of these items and the monitoring details are shown in the right panel. The items that are reported are briefly explained below, for more details on the values reported refer to the [Server Status](#) page.

- ArcGIS Connections
  - Shows details about the connections Weave has created for connections to ArcGIS Server that have pooling enabled. Refer to this [page](#) for more details.
- Bundle Status
  - Provides an overview of the state of the OSGi bundles that make up the Weave instance.
- Config Items
  - Provides an overview of the configuration items that the Weave instance currently has registered.
  - If an item is listed here, it does not guarantee that the configured item is correct and working, just that the configuration (usually from a `config.xml` file) has been read and registered with the system.

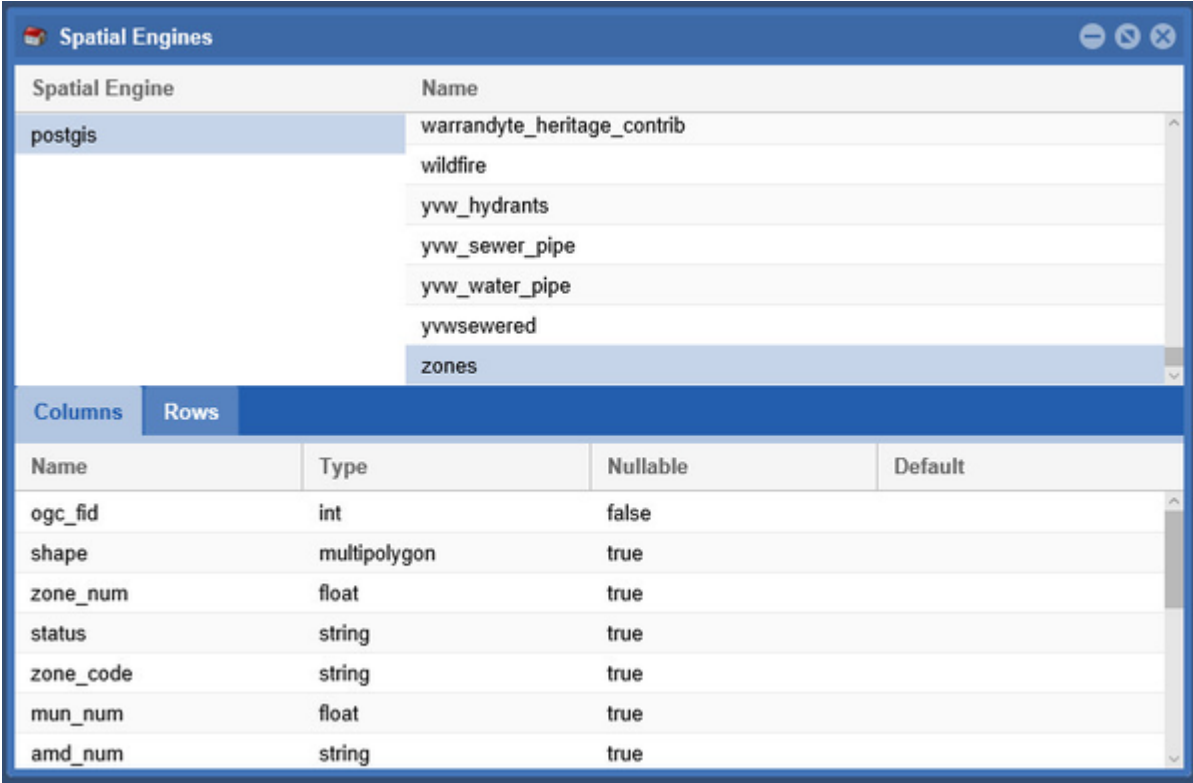
- Items are listed alphabetically by Id with upper case listed before lower case characters.
- Equal Distribution
  - Provides a breakdown of response times (in milliseconds), grouping data so that there is equal (or as equal as possible) time period between each group's minimum and maximum response times. Refer to this [page](#) for more details.
- Hourly Requests
  - Provides an overview of how many requests are made during each hour of the day. Refer to this [page](#) for more details.
- Hourly Response
  - Provides an historical overview of how many requests are made and how long responses are taking broken down by how many hours ago the request was made. Refer to this [page](#) for more details.
- JDBC Connections
  - Shows details about the connections Weave has created for connections to databases that have pooling enabled. Refer to this [page](#) for more details.
- Map Times
  - Shows the average response time (in milliseconds) for a request sent to a Map Engine to generate a map image.
- Layer Times
  - Shows the average response time (in milliseconds) for a request sent to a Map Engine to generate a map image for the Map Engine that contains this layer. Note that this is not how long it takes to draw the individual layer, rather it's how long the entire map image, including all layers, takes to draw, it's basically averaging the Map Times value against the layers.
- Pending Requests
  - Gives a overview of what requests are currently being performed by the server. Refer to this [page](#) for more details.
- Quantile Distribution
  - Provides a breakdown of response times, grouping the data so that there is an equal (or as equal as possible) request count in each group. Refer to this [page](#) for more details.
- Request Details
  - Gives a brief overview of the different requests that the users' browsers have made to the server. Refer to this [page](#) for more details.
- Response Distribution
  - Provides a quick overview of the distribution of response times (in seconds). Refer to this [page](#) for more details.
- Response History
  - Provides an historical overview of how many requests are made and how long responses are taking for periods ranging from the previous minute up to the previous week (up to a maximum of 50,000 requests). Refer to this [page](#) for more details.
- User Details
  - Gives a brief overview of a user's usage of the server. Refer to this [page](#) for more details.
- Weekly Requests
  - Provides an overview of how many requests are made during each hour of the day for each day of the week. Refer to this [page](#) for more details.

Some of the values are reset when the Weave server is restarted and some are persisted across restarts.

This tool provides similar details to those that can be accessed through a browser at: `http://<server>:<port>/weave/server/status`. More information on the Server Status web page can be found on the [Server Status](#) page.

## Spatial Engines Tool

The *Spatial Engines Tool* lists the registered Spatial Engines. If you click on an item from the *Spatial Engine* list in the left panel, the list of layers in that Spatial Engine will be displayed in the right panel. If you click on one item (i.e. layer) from this list, the columns from that layer will be displayed in the *Columns* section in the bottom section of the window. If you click on the *Rows* tab, the first 10 records from that layer will be displayed (as a sample of all the features in the layer).



The screenshot shows a window titled "Spatial Engines" with a list of engines and a detailed view of the "postgis" engine's columns.

Spatial Engine	Name
postgis	warrandyte_heritage_contrib
	wildfire
	yvw_hydrants
	yvw_sewer_pipe
	yvw_water_pipe
	yvwsewered
	zones

Columns		Rows	
Name	Type	Nullable	Default
ogc_fid	int	false	
shape	multipolygon	true	
zone_num	float	true	
status	string	true	
zone_code	string	true	
mun_num	float	true	
amd_num	string	true	

## Support Tool

The *Support Tool* generates a zip file of relevant files from the Weave server. The file is stored in the Downloads area on your Weave server. This file can be sent with your Weave Support Request to Cohga Support ( [support@cohga.com](mailto:support@cohga.com) ) if you are having a Weave issue that you can not resolve.

## System Metrics

As of version 2.6.4 Weave supports metrics, which are counters, gauges and histograms that can be used to monitor the performance of the Weave server.

The metrics can be published to one of four different metrics "databases", [Prometheus](#), [InfluxDB](#), [Datadog](#) and [JAMon](#), or not published at all (which removes the overhead of collecting the metrics if you don't intend to use them).

### Long Term Monitoring

Prometheus, InfluxDB and Datadog provide long term monitoring because they use a database and the metrics are stored in the database so they are not lost after a service or system restart.

These tools provide basic graphing, but are primarily for collecting the metrics over a long period; another tool would be used to display and monitor that information, for example Grafana <https://grafana.com/>. A basic dashboard for Grafana that uses Prometheus is available for [import to Grafana from here](#).

### Short Term Monitoring

JAMon does short term monitoring but not monitoring of your system over the long term. The monitoring is described as short term because once the Weave services are restarted, the metrics will be lost as they are cleared after a restart.

The JAMon metrics registry is for sites that don't have one of the proper metrics registries but want to be able to at least see what the metrics are doing. If you want to make use the metrics to their full potential, you should use one of the other registries.

## Installation

If you perform an upgrade to 2.6.4 from a previous 2.6 release then the metrics will not be installed.

You need to perform a clean install of 2.6.4 and choose to install one of the metrics database providers during installation from the available Extensions, or, after you upgrade to 2.6.4 re-run the 2.6.4 installer (not the updater), uncheck the Main Components and then choose the metrics database provider you wish to utilise from the available Extensions.

**i** There may be an issue with the installer and you should also click on the System Metrics check box to ensure all the components are installed.

**i** At the moment you can only expose the metrics to a single metrics database, even if you manually install more than one of the available extensions.

**Select Installation Packages**

**Select the packs you want to install:**  
**Note: Grayed packs are required.**

Package	Size
Main Components	0 bytes
Extensions	0 bytes
Document Management System	145.29 KB
Geocoding	29.04 KB
Vehicle Tracking	62.79 KB
<b>System Metrics</b>	<b>577.02 KB</b>
System Metrics Core	40.12 KB
JAMon	525.83 KB
Datadog	536.9 KB
InfluxDB	536.57 KB
Prometheus	601.25 KB

The selected package has the following dependencies and/or excludes

**Description**  
Export system metrics to an external metrics database

**Total space required:** 1,012.69 KB  
**Available space:** 13.34 GB

(Made with IzPack - <http://izpack.org/>)

Previous Next Quit

## Configuration

### JAMon

JAMon is an internal metrics database that stores the metrics and makes them available via the existing [Weave server status page](#) (under the Timing Summary page). JAMon requires no configuration.

### Prometheus

Out of the box, the Prometheus database requires no configuration to start it working, and it then presents the available metrics for collection from the `/weave/metrics` URL. To utilise the metrics you need to point your Prometheus server to access that URL to periodically collect the metrics from the Weave server.

**i** If you're going to use a `security.xml` file from a previous version of Weave it may not include an entry for the Prometheus `/metrics` endpoint, if this is the case add the following entry to the `filterInvocationDefinitionSource` property in the `filterChainProxy`, after `PATTERN_TYPE_APACHE_ANT` but before `**=httpSessionContextIntegrationFilter,logoutFilter,...`

```
/metrics=#NONE#
```

Other configuration options:

- descriptions
  - Should meter descriptions be sent to Prometheus? Default is `true`, set to `false` to minimise the amount of data sent on each scrape
- step
  - The step size to use in computing windowed statistics like `max`. The default is 1 minute. To get the most out of these statistics, align the step interval to be close to your scrape interval.
  - The formats accepted are based on the ISO-8601 duration format `PnDTnHnMn.nS` with days considered to be exactly 24 hours, e.g

- "PT20.345S" -- parses as "20.345 seconds"
- "PT15M" -- parses as "15 minutes" (where a minute is 60 seconds)
- "PT10H" -- parses as "10 hours" (where an hour is 3600 seconds)
- "P2D" -- parses as "2 days" (where a day is 24 hours or 86400 seconds)
- "P2DT3H4M" -- parses as "2 days, 3 hours and 4 minutes"

## Datadog

Datadog must at least be configured to provide an `apiKey` and `applicationKey`, but it also supports a number of other configuration options.

```
<config xmlns="urn:com.cohga.server.config#1.0"
  xmlns:datadog="urn:com.cohga.server.metric.datadog#1.0">

  <datadog:config>
    <apiKey>INCLUDE_YOUR_API_KEY_HERE</apiKey>
    <applicationKey>INCLUDE_YOUR_APPLICATION_KEY_HERE<
/applicationKey>
  </datadog:config>

</config>
```

Other configuration options:

- `hostTag`
  - The tag that will be mapped to "host" when shipping metrics to datadog, default is no tag
- `uri`
  - URL to push metrics to, default is `https://app.datadoghq.com`
- `descriptions`
  - Should meter descriptions be sent to Datadog? Default is `true`, set to `false` to minimise the amount of data sent on each scrape
- `step`
  - The step size (reporting frequency) to use. The default is 1 minute.
  - The formats accepted are based on the ISO-8601 duration format `PnDTnHnMn.nS` with days considered to be exactly 24 hours, e.g
    - "PT20.345S" -- parses as "20.345 seconds"
    - "PT15M" -- parses as "15 minutes" (where a minute is 60 seconds)
    - "PT10H" -- parses as "10 hours" (where an hour is 3600 seconds)
    - "P2D" -- parses as "2 days" (where a day is 24 hours or 86400 seconds)
    - "P2DT3H4M" -- parses as "2 days, 3 hours and 4 minutes"
- `enabled`
  - If publishing is enabled. Default is `true`.
- `numThreads`
  - The number of threads to use with the scheduler. The default is 2 threads.
- `connectTimeout`
  - The connection timeout for requests to the backend. The default is 1 second.
- `readTimeout`
  - The read timeout for requests to the backend. The default is 10 seconds.
- `batchSize`
  - The number of measurements per request to use for the backend. If more measurements are found, then multiple requests will be made. The default is 10,000.

## InfluxDB

InfluxDB must be configured to at least provide a `userName` and `password`. InfluxDB also supports a number of other configuration options.

```
<config xmlns="urn:com.cohga.server.config#1.0"
  xmlns:influx="urn:com.cohga.server.metric.influxdb#1.0">

  <influx:config>
    <userName>INCLUDE_YOUR_USERNAME_KEY_HERE</enabled>
    <password>INCLUDE_YOUR_PASSWORD_KEY_HERE</password>
  </influx:config>
```



```
</config>
```

Other configuration options:

- db
  - The db to send metrics to. Defaults to "mydb".
- consistency
  - Sets the write consistency for each point. The Influx default is 'one'. Must be one of 'any', 'one', 'quorum', or 'all'. Only available for InfluxEnterprise clusters.
- retentionPolicy
  - Influx writes to the DEFAULT retention policy if one is not specified.
- retentionDuration
  - Time period for which influx should retain data in the current database (e.g. 2h, 52w).
- retentionReplicationFactor
  - How many copies of the data are stored in the cluster. Must be 1 for a single node instance.
- retentionShardDuration
  - The time range covered by a shard group (e.g. 2h, 52w).
- uri
  - The URI for the Influx backend. The default is `http://localhost:8086`
- compressed
  - if metrics publish batches should be GZIP compressed, default is `true`.
- autoCreateDb
  - if Micrometer should check if db exists before attempting to publish metrics to it, creating it if it does not exist.
- step
  - The step size (reporting frequency) to use. The default is 1 minute.
  - The formats accepted are based on the ISO-8601 duration format `PnDTnHnMn.nS` with days considered to be exactly 24 hours, e.g.
    - "PT20.345S" -- parses as "20.345 seconds"
    - "PT15M" -- parses as "15 minutes" (where a minute is 60 seconds)
    - "PT10H" -- parses as "10 hours" (where an hour is 3600 seconds)
    - "P2D" -- parses as "2 days" (where a day is 24 hours or 86400 seconds)
    - "P2DT3H4M" -- parses as "2 days, 3 hours and 4 minutes"
- enabled
  - If publishing is enabled. Default is `true`.
- numThreads
  - The number of threads to use with the scheduler. The default is 2 threads.
- connectTimeout
  - The connection timeout for requests to the backend. The default is 1 second.
- readTimeout
  - The read timeout for requests to the backend. The default is 10 seconds.
- batchSize
  - The number of measurements per request to use for the backend. If more measurements are found, then multiple requests will be made. The default is 10,000.

## Reverse Proxy

If there is a reverse proxy used in front of the Weave server then additional configuration must be setup for Weave to tell it that there is a reverse proxy. This is so that Weave can process the various headers provided by the proxy to determine the information required about the actual user rather than the proxy server (IP Address, HTTP vs HTTPS, etc).

### Jetty 9

The default installation of Weave includes the Jetty 9 Web Application Server and "running" Weave involves running Jetty and Jetty then runs the Weave application, `weave.war`. So enabling reverse proxy support for a default Weave instance means enabling reverse proxy support for Jetty 9.

To enable reverse proxy support for Jetty 9 you should edit the file `...\\weave\\jetty_base\\etc\\jetty.xml` and add the following lines to the `httpConfig` item (before the closing `</New>` tag).

```
<Call name="addCustomizer">
  <Arg><New class="org.eclipse.jetty.server.
ForwardedRequestCustomizer"/></Arg>
</Call>
```

e.g.

```

    <New id="httpConfig" class="org.eclipse.jetty.server.
HttpConfiguration">
    <Set name="secureScheme"><Property name="jetty.httpConfig.
secureScheme" default="https" /></Set>
    <Set name="securePort"><Property name="jetty.httpConfig.
securePort" deprecated="jetty.secure.port" default="8443" /></Set>
    <!-- more settings here that have been excluded for brevity -->
    <Set name="responseCookieCompliance"><Call class="org.eclipse.
jetty.http.CookieCompliance" name="valueOf"><Arg><Property name="
jetty.httpConfig.responseCookieCompliance" default="RFC6265"/></Arg><
/Call></Set>
    <Set name="multiPartFormDataCompliance"><Call class="org.
eclipse.jetty.server.MultiPartFormDataCompliance" name="valueOf"
><Arg><Property name="jetty.httpConfig.multiPartFormDataCompliance"
default="RFC7578"/></Arg></Call></Set>
    <Call name="addCustomizer">
    <Arg><New class="org.eclipse.jetty.server.
ForwardedRequestCustomizer"/></Arg>
    </Call>
</New>

```



The Jetty forward request customizer relies upon the X-Forwarded-Host and X-Forwarded-Proto headers to be set correctly in the reverse proxy

### Docker

When Weave is run as a Docker container it does the reverse of the default installation. Rather than running Weave within Jetty it runs Jetty within Weave (by including additional Jetty related plugins in the `...\weave\platform\plugins` directory) and this configuration (out of the box) does not use the same configuration files used when running Weave embedded in Jetty. So to make it easier to perform the configuration of the embedded Jetty plugins to correctly parse the required reverse proxy headers, Weave provides a custom plugin to perform the required changes. Currently this has to be enabled manually by you but that may change in the future.

To configure Weave to enable the reverse proxy customizer you should set the system property `org.eclipse.equinox.http.jetty.customizer.class` to the value `com.cohga.jetty8.ReverseProxyCustomizer` (note `jetty8` is correct, the embedded version of Jetty is Jetty 8, not Jetty 9).

### Other

If you're running Weave embedded within a different Web Application Server (i.e. not Jetty 9), you will have to examine the documentation for that Web Application Server to determine if/how it can be configured to correctly parse the proxy headers sent by the reverse proxy.