



Weave

Business Integration Framework

Weave System Administrator Guide Excerpt Feb 2022

1. Developers Guide	3
1.1 What's new in 2.4 for developers	3
1.2 Eclipse	5
1.3 Third Party Application Integration	6
1.4 Building Custom Clients	14
1.5 Debugging client side errors	16

Developers Guide

This guide provides the information required to extend and modify the Weave libraries and also create new components Cohga has not yet implemented.

Weave is made up of many different components which contribute to the entire system. 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 to 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 choose 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.

Intended Audience

Assumptions/Prerequisites

What's new in 2.4 for developers

This page contains a list of some of the bundles that are new and that have been changed as part of Weave 2.4 since version 2.3.22.

New Cohga Bundles

Bundle	Description
com.cohga.client.dms.upload	Actions for uploading and attaching a document to entities
com.cohga.filemanager	API for working with temporary files
com.cohga.filemanager.impl	Implementation of file manager API
com.cohga.fileupload	API for uploading document to the server
com.cohga.fileupload.impl	Implementation of file upload API
com.cohga.server.dms.storage.db	A database implementation of document storage
com.cohga.server.dms.storage.file	A file system implementation of document storage
com.cohga.server.dms.upload	API for document upload
com.cohga.server.dms.upload.core	Implementation of document upload
com.cohga.server.entity.core	Implementation of entity API. Previously this was included in the entity API bundle
com.cohga.server.processor.upload	Base request processor implementation for a document upload handler
com.cohga.server.progress	API for providing control and feedback for long running tasks
com.cohga.server.password	API for password encoding/decoding
com.cohga.server.encryption	API for encryption/decryption
com.cohga.server.encryption.impl	Implementation of encryption/decryption API

Modified Cohga Bundles (excluding minor bug fixes)

Bundle	Description
com.cohga.client.weave.main	Updates to help with file uploading on client
com.cohga.server.dms.core	Provide API for formatting the output of the document list page
com.cohga.server.entity	Split into two bundles, this one is now just the API as well as adding a service interface, IEntityAccess, to provide information about a users access rights to a given entity id
com.cohga.server.oda	Updated for BIRT 2.6
com.cohga.server.processor.json	Provides some additional support methods that are common to a lot of request processors
com.cohga.server.report.birt	Updated for BIRT 2.6
com.cohga.server.script.init.toc	Added support for requests to get and create toc models on the fly

com.cohga.server.script.spatial.geometry	Minor internal changes
com.cohga.server.security	Removed support for OSGi user admin API and added support for dynamically processing updates to security.xml file
com.cohga.server.selection	Changed the selection building API
com.cohga.server.setup	Added support for using environment variables for substitution in config files
com.cohga.server.spatial.utils	Updated to use the new progress monitoring API
com.cohga.server.user.core	Removed support for OSGi user admin API
com.cohga.support	Added new cryptography, i18n, IO and general support classes

New Third Party Bundles

Bundle	Version	Description
org.apache.commons.fileupload	1.2.1	Low level file upload handling library, wrapped in simple API by the com.cohga.fileupload bundle
org.apache.commons.io	1.4.0	Low level file IO handling library, used by org.apache.commons.fileupload
java.xml.rpc	1.1.0	Java XML RPC library
javax.xml	1.3.4	Java XML library
javax.xml.soap	1.3.0	Java XML Soap Library
jackson	1.5.5	JSON parser and generator library, required for jettison and jersey
jersey	1.4.0	REST API implementation
jettison	1.1.0	A StAX implementation for JSON
javax.ws.rs	1.1.1	REST API library
org.hsqldb	2.0.0	HSQL DB Driver

Modified Third Party Bundles

Bundle	Version
com.vividolutions.jts	1.0.9 -> 1.0.10
org.apache.commoms.beanutils	1.7.0 -> 1.8.2
org.apache.commons.collections	3.2.0 -> 3.2.1
org.apache.commons.discovery	0.2.0 -> 0.4.0
org.apache.commons.lang	2.1.0 -> 2.4.0
org.apache.commons.pool	1.3.0 -> 1.5.4
org.apache.lucene	2.4.1 -> 2.9.4
org.geotools	2.5.7 -> 2.6.5
org.geotools.arcsde	2.5.7 -> 2.6.5
org.geotools.epsg.wkt	2.5.7 -> 2.6.5
org.geotools.oraclespatial	2.5.7 -> 2.6.5
org.geotools.postgis	2.5.7 -> 2.6.5
org.geotools.shapefile	2.5.7 -> 2.6.5
org.geotools.sqlserver	2.5.7 -> 2.6.5
org.geotools.wfs	2.5.7 -> 2.6.5
org.geotools.wms	2.5.7 -> 2.6.5
org.json	1.0.6 -> 1.0.7
org.opengis	2.2.0 -> 2.3.0
org.slf4j	1.3.1 -> 1.5.11

groovy	1.5.6 -> 1.7.5
birt	2.2.1 -> 2.6.1
eclipse	3.3.1 -> 3.6.1

Eclipse

Weave development doesn't require that you use the Eclipse IDE, but given the in-built support for OSGi development the core Weave development is generally performed using Eclipse and any examples shown here will be using Eclipse.

If you need to download Eclipse ensure that you download the version that includes support for plugin development.

Setting up Eclipse for Weave development

[This video](#) shows how to setup an Eclipse environment for development with Weave and [this one](#) is an update for Windows.

[Jetty development bundles for Eclipse](#)

Basic development options

```
-server
-Xms512m
-Xmx2048m

-Djava.net.preferIPv4Stack=true
-Djava.awt.headless=true
-Declipse.ignoreApp=true
-Dosgi.noShutdown=true
-Dosgi.console.enable.builtin=true
```

Setting Jetty parameters

```
-Dorg.osgi.service.http.port=8080
-Dorg.eclipse.equinox.http.jetty.context.path=/weave
-Dorg.eclipse.jetty.server.Request.maxFormContentSize=5000000
```

Some helpful Weave settings

```
-Dweave.cache.default=false
-Dweave.script.debug=true
```

Changing file locations

```
-Dconfig.location=/home/sforbes/projects/weave/platforms/platform-r24
/workspace/config.xml
-Dsecurity.location=/home/sforbes/projects/weave/platforms/platform-
r24/workspace/security.xml
-Dosgi.debug=/home/sforbes/projects/weave/platforms/platform-r24
/workspace/debug.options
```

Adding external libraries

```
-Djava.library.path=/usr/local/arcgis/arcscde10/dev/lib:/usr/local
/arcgis/arcscde10/sdeexe100/lib
```

Proxy Settings

```
-Dhttp.proxyHost=192.168.0.11
-Dhttp.proxyPort=3128
-Dhttp.nonProxyHosts=*.local|localhost|192.168.0.
*|arcserver10|arcserver93|arcserver|arcims10|geoserver
```

Third Party Application Integration

WeaveLink

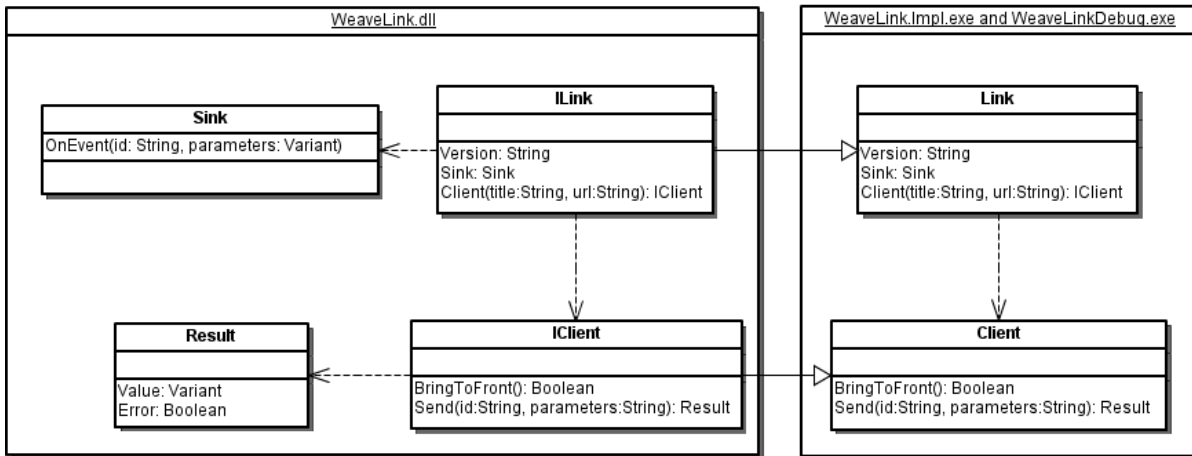
WeaveLink is a Windows COM API that can be used for communication between Weave and another application or web page. See [this page](#) for more information about installing and testing WeaveLink.

The WeaveLink COM API provides the following functionality:

- Locate an existing browser instance running Weave
- Create a new browser running Weave if an existing one can't be found
- Bring the Weave browser to the front of other windows
- Execute an action in the Weave client and return the results
- Listen for events fired by the Weave client

As you can see the actual functionality provided by the API is very limited, with most of the actual work being performed by the Weave client running in the browser. This simplified API means that new functions can be called without the need to upgrade the WeaveLink components on every client PC.

Unfortunately this also means that a certain amount of boiler plate code needs to be added to actually make use of this API, work is underway to provide a higher level API in Weave 2.4 that sits on top of the WeaveLink API and provides functionality in terms of entities, maps, etc, but this page documents the WeaveLink COM API.



Getting a link to the Weave client

⚠ Unless indicated otherwise all code samples below were written and tested using Visual Basic in Visual Studio 2005.

But it's possible to use the WeaveLink COM API in older and newer version of Visual Basic and Visual Studio, as well as Windows Scripting Host and even VBScript or JavaScript within a web page.

The first step when communicating with the Weave client is to create a `WeaveLink.ILink` object.

The basic interface for this class is contained in `WeaveLink.dll` with either `WeaveLinkImpl.exe` or `WeaveLinkDebug.exe` providing the implementation.

⚠ If you have a Weave instance available then `WeaveLinkImpl.exe` should be used, if you don't then `WeaveLinkDebug.exe` can be used to provide a test harness.

This class provides a way to:

- Obtain a reference to a Weave client
- Check what version of the `WeaveLink` API is being used (useful for weakly types languages)
- Obtain a reference to a `Sink` that can used to respond to events triggered in the Weave client

So to create the `ILink` object you would use something like the following

```
Dim link As WeaveLink.ILink
link = CreateObject("Weave.Link")
```

Then once you have a link you can obtain a reference to a Weave client by using its `Client` method.

```
Dim client As WeaveLink.IClient
client = link.Client("Weave HTML Client", "http://wrath:8080/weave/main.html")
```

This will locate a running instance of Internet Explorer that contains the Weave client based on the window title given as the first parameter, or start a new instance using the second parameter if it can't find one.

Communicating with the Weave client

Once we have a reference to the client we can then call actions that are provided by the JavaScript embedded in the client by using the `Send()` method of the `IClient` class.

The `Send()` method takes an action id and a parameter string, in JSON format, and passes that to the Weave client and returns the results of that action in a `Result` object.

The `Result` class provides an `Error` Boolean property that indicates if there was an error executing the action, if the `Error` property is `true` then the operation failed and the `Value` property of the `Result` will be a `String` describing the problem. If the `Error` property is `false` then the `Value` property will return an object that is the actual response from the action.

```
Dim response As WeaveLink.Result

response = client.Send("com.cohga.GetActiveEntity", "{}")
If response.Error Then
    MsgBox(response.Value, MsgBoxStyle.Exclamation, "Error")
Else
    Dim entity As String
    entity = response.Value
End If
```

```
response = client.Send("com.cohga.SetActiveEntity", "{entity:
'property'}")
If response.Error Then
    MsgBox(response.Value, MsgBoxStyle.Exclamation, "Error")
End If
```

From here it's just a matter of knowing what actions are available, what parameters they require and what they return.

Actions available in Weave 2.3

com.cohga.ZoomSelection

Parameters

name	type	optional	description
entity	string	yes	The id of the entity who's selection should be zoomed to, the active entity will be used if this parameter isn't supplied

Zoom to the selection of the `property` entity

```
response = client.Send("com.cohga.ZoomSelection", "{entity:
'property'}")
If response.Error Then
    MsgBox(response.Value, MsgBoxStyle.Exclamation, "Error")
End If
```

Zoom to the current selection of the active entity


```

response = client.Send("com.cohga.ZoomSelection", "{}")
If response.Error Then
    MsgBox(response.Value, MsgBoxStyle.Exclamation, "Error")
End If

```

com.cohga.SetSelection

Parameters

name	type	optional	description
entity	string	no	The id of the entity who's selection should be updated
ids	object or object[]	no	The id or array of ids to update the selection with
operator	'REPLACE', 'ADD', 'REMOVE', 'REFINE', 'CLEAR'	yes	Determines what will be done with the selection set relative to the current selection, the default if not set is 'REPLACE'
clearOthers	boolean	yes	If true the selections for all other entities will be cleared during the processing of this action, if false or not set then other entities selections will be untouched
active	boolean	yes	If true or not set then the active entity will also be changed to this entity, if set to false then the active entity won't be changed
zoom	boolean	yes	If true or not set then the map will also zoom to the new selection extent, if set to false then the map extent won't be changed

```

response = client.Send("com.cohga.SetSelection", "{entity:
'property', ids: [11000, 11001], clearOthers: true}")
If response.Error Then
    MsgBox(response.Value, MsgBoxStyle.Exclamation, "Error")
End If

```

```

response = client.Send("com.cohga.SetSelection", "{entity: 'roads',
ids: ['PENNY LANE', 'BOURBON STREET', 'ELECTRIC AVENUE'], active:
false, zoom: false, operator: 'ADD'}")
If response.Error Then
    MsgBox(response.Value, MsgBoxStyle.Exclamation, "Error")
End If

```

com.cohga.GetSelection


Parameters

name	type	optional	description
entity	String or String []	yes	The id, or array of id's, of the entity who's selection should be retrieved, if not set then all entities that the user has access to will be returned
returnids	Boolean	yes	If true or not set then the current list of id will also be returned, if set to false then only the current size of the selection will be returned

Return

name	type	optional	description
------	------	----------	-------------

entity	String	no	The id of the entity who's selection is being returned
size	Integer	no	The number of unique identifiers currently selected for the entity
ids	Object []	yes	The individual id's for the entity, not included if <code>returnids</code> is set to false

 Regardless of how many entities are requested this action will always return an array of object of the above format.

```

response = client.Send("com.cohga.GetSelection", "{entity:
'property', returnids: false}")
If Not response.Error And Not response.Value Is Nothing Then
    Dim count As Integer
    Dim i As Integer
    count = CallByName(response.Value, "length", CallType.Get)
    For i = 0 To count - 1
        Dim v As Object
        v = CallByName(response.Value, i, CallType.Get)
        Debug.Print(v.entity & " " & v.size)
    Next
End If

```

property 2

```

response = client.Send("com.cohga.GetSelection", "{entity:
['property', 'roads'], returnids: false}")
If Not response.Error And Not response.Value Is Nothing Then
    Dim count As Integer
    Dim i As Integer
    count = CallByName(response.Value, "length", CallType.Get)
    For i = 0 To count - 1
        Dim v As Object
        v = CallByName(response.Value, i, CallType.Get)
        Debug.Print(v.entity & " " & v.size)
    Next
End If

```

property 2
roads 6

```

response = client.Send("com.cohga.GetSelection", "{returnids: false}")
If Not response.Error And Not response.Value Is Nothing Then
    Dim count As Integer
    Dim i As Integer
    count = CallByName(response.Value, "length", CallType.Get)
    For i = 0 To count - 1
        Dim v As Object
        v = CallByName(response.Value, i, CallType.Get)
        Debug.Print(v.entity & " " & v.size)
    Next
End If

```

```

buildings 0
property 2
busstops 0
roads 6
drainage 0
development_applications 0
hydrants 0
suburbs 0

```

```

response = client.Send("com.cohga.GetSelection", "{entity: 'roads'}")
If Not response.Error And Not response.Value Is Nothing Then
    Dim count As Integer
    Dim i As Integer
    count = CallByName(response.Value, "length", CallType.Get)
    For i = 0 To count - 1
        Dim v As Object
        v = CallByName(response.Value, i, CallType.Get)
        Debug.Print(v.entity & " " & v.size)
        Dim ids As Object
        Dim idcount As Integer
        Dim j As Integer
        ids = CallByName(v, "ids", CallType.Get)
        idcount = CallByName(ids, "length", CallType.Get)
        For j = 0 To idcount - 1
            Debug.Print(CallByName(ids, j, CallType.Get))
        Next
    Next
End If

```

```
roads 6
ABBEY ROAD
BOURBON STREET
LONELY AVENUE
PENNY LANE
ELECTRIC AVENUE
GASOLINE ALLEY
```

com.cohga.SetActiveEntity

com.cohga.GetActiveEntity

com.cohga.BeginUpdate

com.cohga.EndUpdate

Actions available in Weave 2.4

com.cohga.ZoomSelection

com.cohga.SetSelection

com.cohga.GetSelection

com.cohga.SetActiveEntity

com.cohga.BeginUpdate

com.cohga.EndUpdate

com.cohga.ClearSelection

com.cohga.GetActiveEntity

com.cohga.GetEntities

com.cohga.GetEntity

com.cohga.GetExtent

com.cohga.SetExtent

com.cohga.SetLocation

com.cohga.GetLocation

com.cohga.GetScales


com.cohga.GetVersion

Parameters

name	type	optional	description
item	string or string []	yes	The id or, array of id's, of the item who's version should be return, if null then version number for all components will be returned

Return

name	type	optional	description
item	String	no	The name of the item
version	String	no	The version string, format will always be major[.minor[.release]], where major, minor and release will be numeric and minor and release are optional
major	Integer	no	The major version number of the item
minor	Integer	yes	The minor version number of the item
release	Integer	yes	The release version number of the components

 If one item is requested then only 1 object of this format will be returned, if multiple items (even if only 1 item requested finds a match) or all items are requested then an array of objects of this format will be returned.

```
response = client.Send("com.cohga.GetVersion", "{item: 'weave'}")
If Not response.Error Then
    Dim v As Object
    v = response.Value
    Debug.Print(v.Item & " " & v.Version & " " & v.Major & " " & v.
Minor & " " & v.Release)
End If
```

```
weave 2.4 2 4
```

```
response = client.Send("com.cohga.GetVersion", "{item: ['client',
'weave']}")
If Not response.Error And Not response.Value Is Nothing Then
    Dim count As Integer
    Dim i As Integer
    count = CallByName(response.Value, "length", CallType.Get)
    For i = 0 To count - 1
        Dim v As Object
        v = CallByName(response.Value, i, CallType.Get)
        Debug.Print(v.Item & " " & v.Version & " " & v.Major & " " &
v.Minor & " " & v.Release)
    Next
End If
```

```
client 2.1.1 2 1 1
weave 2.4 2 4
```

```

response = client.Send("com.cohga.GetVersion")
If Not response.Error And Not response.Value Is Nothing Then
    Dim count As Integer
    Dim i As Integer
    count = CallByName(response.Value, "length", CallType.Get)
    For i = 0 To count - 1
        Dim v As Object
        v = CallByName(response.Value, i, CallType.Get)
        Debug.Print(v.Item & " " & v.Version & " " & v.Major & " " &
v.Minor & " " & v.Release)
    Next
End If

```

```

weave 2.4 2 4
client 2.1.1 2 1 1
interop 2.1.0 2 1 0

```

Building Custom Clients

i This content is a work in progress and may or may not be accurate.

This page describes the components that make up what may become the next method of building clients for Weave.

The current Weave 2.x desktop client is developed using the Sencha Ext.js framework, 2.x in Weave 2.4 and 3.x in Weave 2.5, and as part of the migration to newer versions of the Sencha libraries, including Ext.js 4.x and later and Sencha Touch 2.x and later, a new method of providing the client content to browsers is required.

The new method of providing client content to the browser should address some or all of the following:

- Provisioning of third party content
 - Currently there is no standard method of including third party JavaScript libraries in Weave
 - This implementation should provide a way to package and publish third party JavaScript libraries
 - Currently some of the third party libraries that Weave uses must be changed before that can be published, e.g. paths to images must be changed
 - This implementation should minimise the customisation required to third party libraries before they can be used
 - Currently there is no standard method of versioning third party libraries
 - This implementation should provide a way to publish multiple versions of third party libraries
 - Currently there is no standard method of publishing third party components for use with the Sencha frameworks
 - This implementation should provide a way to package and publish components in a standard manner
- Provisioning of Weave client content
 - Currently the JavaScript code that makes up the Weave client has minimal dependency analysis, i.e. it uses a pre-defined list of folder names to order content, but there is no way to specify dependencies within those folders
 - Code in newer versions of the Sencha frameworks are more structured in the way they specify dependencies, this can be leveraged by Weave to ensure full dependency analysis is performed on the code supplied to the browser
- Separation of JavaScript code
 - Currently code from any bundle installed in a Weave instance will be pushed to the client when it starts, regardless of whether the code is used or not, e.g. if the Pathway bundle is installed every client that's started includes the Pathway JavaScript code, even if the client is not configured to expose the Pathway components.
 - This implementation should provide a way to separate code into different packages that can be included only if required
- Provisioning of debug versus production code
 - There is currently no standard method of supplying different development and production versions of code to the client, e.g. minified code versus commented code
 - This implementation should provide a means of publishing both development versions and production versions of JavaScript code
- Provisioning different types of clients
 - Currently there is only one client type that can be easily published, i.e. there is only one Weave client, all be it a very customisable one
 - This implementation should allow for the creation of other client types, like the Admin UI for example
- Embedding within other web pages
 - Currently it's not easy to embed the Weave client directly within another web page

- This implementation should help make it possible to embed Weave resources within another page, e.g. within a div element on a page
- Note that the primary issue to be addressed here is the inclusion of Weave JavaScript code and CSS styles into another web page, it does not necessarily imply the ability of the actual Weave components to be embedded in the page, that may require changing the code of the components, that is, this update is primarily intended to address the provisioning of the code to the client, not the code itself.
- This should be done by providing "standard" url's for retrieving Weave code and resources.

Provisioning of Third Party Content

Currently there are three ways that third party JavaScript libraries are pushed to the browser:

1. Third party libraries that Weave depends upon just to run are published by the `com.cohga.client.weave` bundle (note that some of the other resources for those libraries are published by the `com.cohga.client.weave.main` bundle), and these are directly included by the Weave client startup code when the browser loads.
2. Libraries embedded in client bundles, this code is currently extracted from the bundle and appended to a giant script sent to the browser after the third party libraries listed above.
3. Dynamically loaded JavaScript code, this code is downloaded on the fly when it's first required.

There are a number of issues with this:

1. The `com.cohga.client.weave` and `com.cohga.client.weave.main` bundles are too dependant upon each other and have become large and unwieldy
 - a. JavaScript is in one bundle CSS resources in another.
 - b. The `com.cohga.client.weave` bundle performs too many tasks. Currently this is the one bundle that's responsible for the entire loading of the client (with the `com.cohga.client.weave.main` providing primarily core Weave code and resources), which isn't necessarily a bad thing, but it also performs other function it does not need to, e.g. being the repository for the core libraries, providing the default "index" page.
2. Updates to the libraries are unnecessary complex
 - a. A minor bug fix to one of the core libraries could mean that one or both of the `com.cohga.client.weave` and `com.cohga.client.weave.main` bundles needs to be updated.
 - b. Some of the third party libraries must be changed before they can be published
 - c. Change to Weave code that uses a third party library embedded in the same bundle, or vis-versa, an update to the third party library in the bundle, requires that both things be updated when they don't need to be, i.e. the Weave code that uses a third party library should not require the third party library to be updated if it hasn't changed.
3. The release of an update to a third party library may break an existing clients
 - a. In certain circumstances it may be advantageous to tie a client to a well tested and consistent version of a library, rather than automatically updating it to the latest version when a semi-major update is applied, for example an update of a library from 4.2 to 4.3 may introduce new functionality, but it also may introduce new bugs, so it would be useful to specify that a client only uses the latest 4.2.x version of a library. Initially this may primarily be of use in customised clients, or when Weave is embedded in other applications.

The proposed solution to this is to package third party libraries into their own bundles, which must be individually included into the client as (and when) required. Note that at the moment we're just addressing how the code and resources will be published, not how the client will retrieve them.

The new implementation allows packaging of the libraries into their own bundles so they can be updated independently of any code that uses them and the bundle can include multiple versions of the third party library. Additionally it minimises the changes required to the libraries, with currently one exception being third party Ext.js components that do not use a distinct package name, e.g. two bundles, one providing Ext.ux.Grid and another providing Ext.ux.Table can cause issues because both libraries are trying to expose content at Ext.ux, these components will need to be changed before they can be published, for example by renaming them to Ext.ux.grid.Grid and Ext.ux.table.Table (which is actually more standard for Sencha components anyway).

Building a third party library bundle

Packaging a third party library into its own bundle has two requirements, firstly the content for the library is copied into the bundle at a specific location, and secondly a custom header is added to the MANIFEST.MF file.

The content for the library code and resources should be stored within a directory called 'content' within the bundle, additionally a sub-directory for each version of the library should be created under the content directory, this is where the code and resources are placed, generally exactly as it comes from the original download.

Third party library versions

The name of the "version" directory will specify the version number that the code and resources will be published as, and should follow the pattern X.Y.Z.Q, where X and Y are required and should be numbers representing the major and minor version of the library, Z is an optional (unless a qualifier is specified) release number for the library, and Q is an optional qualifier.

Some examples of valid library version numbers are:

- 2.2
- 2.2.1
- 2.2.1.rc5

These version numbers are not valid

- 2 – minor version number is always required

- 2.rc5 – minor version and release number required if a qualifier is specified, or alternatively "rc5" isn't a valid minor version number, it should be numeric
- 2.2.latest – release number required if a qualifier is specified, or alternatively "latest" isn't a valid release version number, it should be numeric

As an example assume that we wish to package up the jQuery library, and assume that the current version is 1.11.2, then the location of the jQuery library should be extracted to the directory `/content/1.11.2/` within our new jQuery library bundle. Then down the track a new version of the jQuery library is published and by the time we notice it's up to version 2.0.3, so the content for the new version should be extracted to the `/content/2.0.3` directory, and the existing 1.11.2 version left alone. Further if additionally versions 1.11.3 and 2.1.4 of jQuery are release they would be extracted to `/content/1.11.3` and `/content/2.1.4` respectively.

In addition to the directory name there's another location in a bundle that's related to the version, and that's the actual OSGi bundle version, as set in the MANIFEST.MF. The bundle version should always reflect the version number of the latest release of the third party library contained within the bundle. So in our jQuery example above the Bundle-Version header in the MANIFEST.MF file would start out as 1.11.2, then be updated to 2.0.3, then finally to 2.1.4.

Finally, the version numbers that are used in the directory names are the full version numbers that the content is published as, so it's possible for a client to retrieve the `jquery.js` file using the path `.../2.1.4/jquery.js` (as well as `.../2.0.3/jquery.js`, `.../1.11.3/jquery.js` and `.../1.11.2/jquery.js`). But, Weave also published the "latest" versions of the libraries using abbreviated version numbers, for example the 2.1.4 version of the `jquery.js` file is also available using the path `.../2.1/jquery.js` or `.../2/jquery.js`, and the 2.0.3 version available at `.../path/2.0/jquery.js`, and finally the 1.11.3 release at `.../1.11/jquery.js` or `.../1/jquery.js`.

This allows a client to reference a preferred version of the third party library but still receive bug fix updates without having to change the client itself, this is assuming that the ignored portion of the version number signifies a bug fix and not a whole new version, but this is generally dependant upon the library itself. So for example having our client include `.../2.1/jquery.js` mean that it will automatically receive bug fixes to the jQuery library if it's updated to 2.1.5. This is assuming the change from 2.1.4 to 2.1.5 signifies a bug fix release, it could be that a library used just X.Y version number and a change from 2.1 to 2.2 signifies a bug fix, and a change from 2.1 to 3.0 signifies new functionality, in which case you'd use `.../2/filename.js` to include the library.

Finally, every library is also published using "latest" as the version number, so regardless of the version number the newest version of the library will be available using "latest" rather than the version number, so for our jquery example version 2.1.4 of jquery would also be available at `.../latest/jquery.js`.

Third party library paths

I mentioned earlier that there are two requirements to publish a third party library, extracting the content to the correct location, and adding a custom header to the MANIFEST.MF file. The purpose of the custom header in the MANIFEST.MF file it to provide the base portion of the path that the library content will be published at. In the examples in the last section I presented the locations of the files as `.../2.1/jquery.js` for example, this header specifies what will be used for the `...` portion of that path.

The name of the header to include in MANIFEST.MF is X-Weave-Library and should specify a unique base path for the content, for example with the jQuery example it could be to `/jquery`

```
X-Weave-Library: /jquery
```

Weave will then publish the `/content` directory in our jQuery example bundle at `/jquery`, so the `jquery.js` file for version 2.1.4 would be available at `/jquery/2.1.4/jquery.js` (plus `/jquery/2.1/jquery.js`, `/jquery/2/jquery.js` and `/jquery/latest/jquery.js`)

Third party library content

So far we've just made a passing reference to the actual content of the third party library that gets extracted into the bundle, but it's important to understand what has to go into this directory, and this largely depends upon the library itself.

For the jQuery example it's pretty straight forward as there's really only two files that would be extracted into the bundle, the compressed and the uncompressed versions of the jQuery JavaScript files, almost exactly as they're downloaded from the jQuery web site. I say almost because jQuery, like a number of libraries, include the version number in the file name, so for example if I were to download the latest version of uncompressed jQuery library right now I'd get a file called `jquery-2.1.4.js`, and if I were to just include that file as it was into the bundle, in the `/content/2.1.4/` directory, the client would need to reference it as `/jquery/2.1.4/jquery-2.1.4.js`, which means that if the client just wants the latest version, or even the latest 2.1 version, then they'd have to ask for `/jquery/latest/jquery-2.1.4.js` or `/jquery/2.1/jquery-2.1.4.js`, and what happens when 2.1.5 is release? Now the client needs to be updated to ask for `/jquery/2.1/jquery-2.1.5.js`, defeating our whole mechanism of exposing multiple versions of the same library.

What this means is just that the `jquery-2.1.4.js` file, and `jquery-2.1.5.js` file after it, should be renamed to `jquery.js` once it's been extracted into the bundle. The same for the compressed version, `jquery-2.1.4.min.js`, which should be renamed to `jquery-min.js`, or something similar.

Note that there is currently no standard for how these files should be named, either from the original third parties themselves, or if they're renamed to confirm to the requirements of Weave. This may have to be addressed eventually so that client's can reference a library by name, and not have to search for the correct file name to include the library, especially when we're going to build the client web page automatically rather than hand crafting the HTML code, which at this point is time is how these clients are created.

Debugging client side errors

Tracking down client side errors will be different depending upon the browser client (and version) because of the varied level of support for debugging provided by the various browsers.

When tracking down an error on the client the first thing you should make sure of is that `<debug>true</debug>` is included in the client config.

Then for IE the trick is the developer console, which is under the Tools|Developer Console menu or F12.

This is for IE 8 & 9, for IE 7 and earlier it is trickier because you may need to install the Microsoft Office Script Debugger, which adds the required functionality to those browsers (actually I'm not sure about IE 7, it may have the developer console already).

From there you can go to the Script tab and look at the console. Hopefully that will contain more information to help you resolve the issue.

If it doesn't then hitting the "Start Debugging" button or pressing F5 will reload the page and it should then show the errors in the Console.

If it's an error at startup there will usually be two errors listed, the first one is the important error and you can click on the blue link in the error to go to the line that's giving the error which hopefully will provide some sort of clue.

Also, testing Firefox or Chrome will generally produce better error information than IE, so even if you need to use IE it can be useful to test this with FF or Chrome.