WoodWing Software
Enterprise Web Services Guide

Version 3.2.0

Introduction

Before Enterprise 7, a System Integration Guide was available for Enterprise 4. That document has now been discontinued in its current form and split up into two components:

- **Web Services Guide** (this document)**.** This guide provides in-depth information about how client applications interact with the Enterprise Server system through Web Services. It looks from the 'outside' to Enterprise Server and explains concepts, interfaces, operations, data structures, etc.
- **Server Plug-ins.** This is not a guide but a bundle of presentations, practices and examples, explaining how to develop plug-ins. It looks from the 'inside' of Enterprise Server and describes the concept of plug-ins and how to customize behavior of Web Services and business logic.

The Web Services Guide intends not to be complete but to give a good starting point for integrating systems with Enterprise. The explained concepts provide basic ideas and behavior which are concealed underneath Enterprise's abstract entities. This theory is frequently diversified by concrete examples to visualize how things actually work.

This guide can be used for Enterprise 5 or higher. In some places, it refers to the Server Plug-ins technology, for which Enterprise 6 (or higher) is required.

Required knowledge

We assume the reader of this guide is familiar with:

- The functionality of the Enterprise system. For more information, check the Smart Connection User Guide.
- The configuration of the Enterprise system. For more information, check the Enterprise Server Admin Guide.
- Web Services, SOAP and WSDL. For more information, see http://www.w3.org/2002/ws/.
- PHP language. For more information, see: www.php.net.

Revision history

Revision	Description
v1.0.0	Initial writing. Up-to-date for Enterprise 7.
v2.0.0	Updates for Enterprise 7.4, 7.6 and 8.0.
v3.0.0	Updates for Enterprise 8.2 and 9.0.
v3.0.1	Added notes for Java SOAP clients.
v3.1.0	Updates for Enterprise 9.1, 9.2, 9.3, 9.4 and 9.5.
v3.2.0	Updates for Enterprise 9.7 and 9.8

Conventions used in this manual

Throughout this guide, Smart Connection is sometimes abbreviated as SC, while Content Station is sometimes abbreviated as CS.

The following special styling is used:

Italic	Literal reference. Typically used to refer to a term in an example as used literally.

colored	Literal reference. Typically used to refer from one example to another as used literally.
[n]	Numeric reference. Typically referring to a specific numeric bullet used in an example.
“quoted”	Literal text. Typically used literally in English GUI.
underlined	Hyperlink reference to web site, paragraph or chapter.
...	Used in examples to indicate there is more information, which is not relevant, and therefore hidden.
->	Path separator used in references locating data in hierarchically structured data trees, such as SOAP. For example Hello -> World refers to “hi” in this fragment: <Hello> <World>hi</World> </Hello>
- -	Description of a feature that was introduced since the specified version. In this example, since Enterprise v6.1.
Mixed Case	A name, term or entity that is considered to be known to the reader, such as Enterprise or WoodWing. Note that SOAP elements (as defined in WSDL files) are also mixed case and therefore literally used in this guide.

Support

To discuss any Web Services related issues, visit WoodWing’s Community site: http://community.woodwing.net/forums/enterprise-development/webservices (log-in required). If you require further support for Enterprise, visit the WoodWing support web site at http://www.WoodWing.com/support and follow the directions for submitting questions.

Table of Contents

Enterprise and Web Services

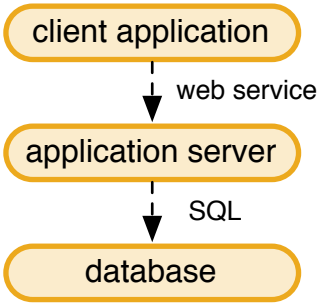
Web Services are a keystone of the Enterprise architecture. Many services are exposed by Enterprise Server and are defined in documents (WSDLs). Client applications communicate with Enterprise Server through services. A client can be any of the Enterprise applications such as InDesign or Content Station, but could also be your custom client applications that you are about to develop in order to integrate Enterprise. This section explains how this all comes together and gives an introduction of Enterprise services: a good starting point for integrators.

System Architecture

This chapter provides a global overview of the Enterprise architecture, followed by how Web Services fit into the big figure.

3-tier architecture

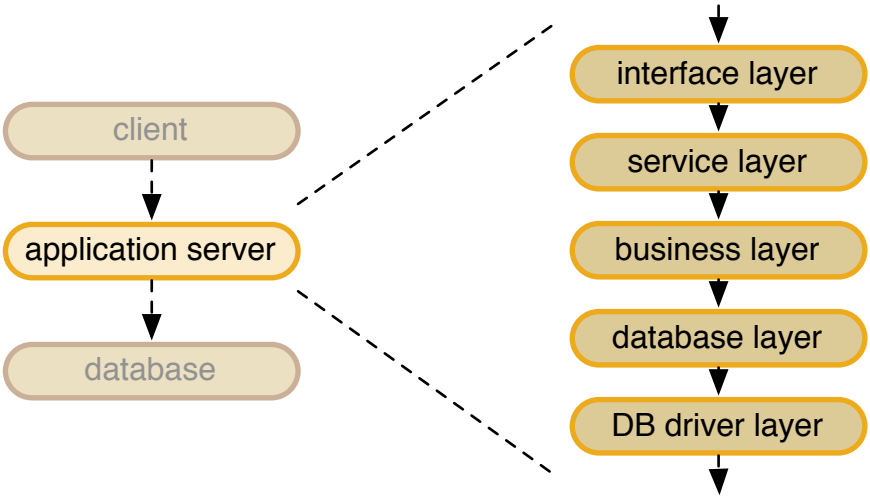
Enterprise uses a 3-tier architecture:



Users interact with *client applications*. They can use Content Station, Smart Connection InDesign / InCopy or a web browser running a web application. But also Smart Mover and Smart Connection for InDesign Server as well as third-party systems can act as a client when talking to the *application server*. All of these clients can request the *application server* through HTTP to run any of its Web Services. Enterprise Server is the *application server* and has all business logic to determine which actions are allowed, which triggers other actions, etc. The server connects through a database-independent module to the underlying SQL *database* and possibly file server to store the binary files.

Application server layers

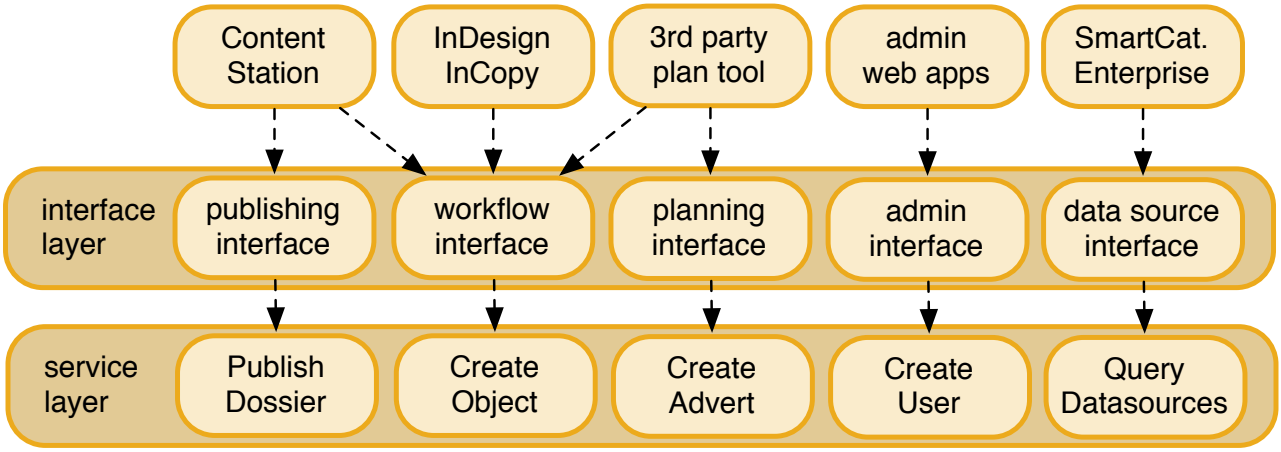
Zooming into the *application server* reveals the following layers:



An incoming Web Service request fired from a client arrives at the *interface layer*. This layer interprets the incoming structure and transforms it into a PHP object structure. The object structure is passed onto the *service* layer. The actual implementation is done at the *business layer*. This uses the *database layer* to transform the object structure into 'generic' SQL statements. The *DB driver layer* is responsible for translating the SQL statements into dedicated SQL for a specific database flavor, such as MySQL, MSSQL or Oracle.

Client applications calling Web Services

Client applications connect to one or many interfaces through which they fire Web Service requests. At the application server, a Web Service comes into action. The following overview shows how that is done.



For example, an InDesign user has created a new document and is about to Check-In the layout. The Smart Connection plug-ins in *InDesign* are connected to the Enterprise Server through the *workflow interface*. On the user's Check-In command, they fire a CreateObjects request through the *workflow interface*. The server invokes the *CreateObjects* service to store the layout.

Protocols [since 8.0]

Enterprise Server 7 (and earlier versions) supports the SOAP protocol. Because this protocol has some performance- and integration disadvantages, Enterprise Server 8 supports two additional protocols (AMF and JSON) that can be used instead of SOAP.

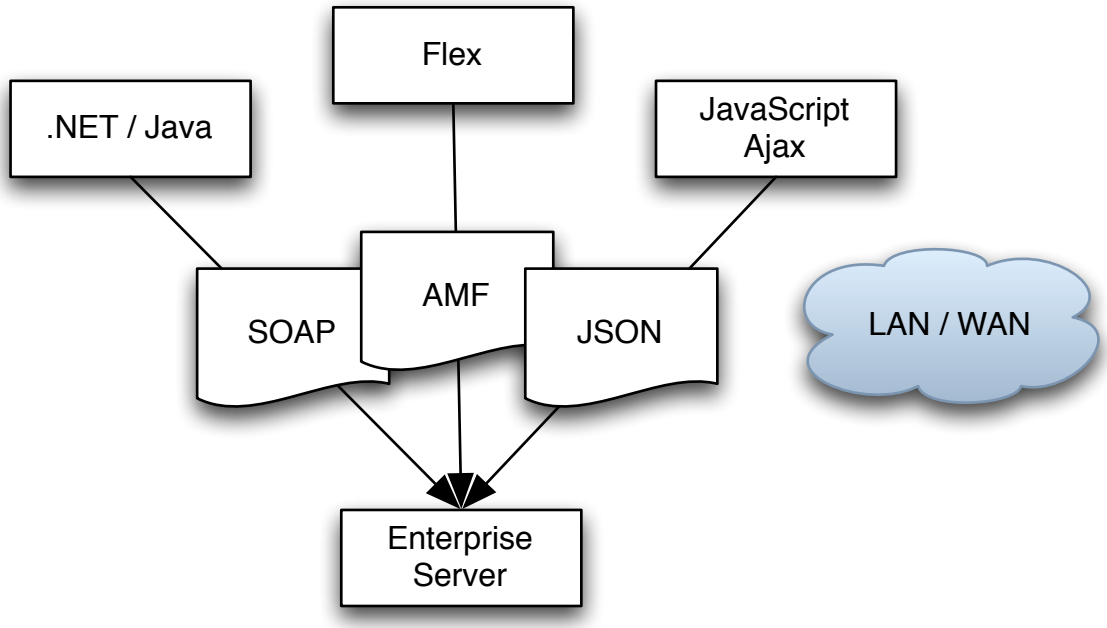
SDK documentation vs WSDL files

The SOAP protocol is defined in WSDL files. Even though the requests, responses and data structures sent between client and server are essentially the same as for AMF and JSON protocols (only wrapped differently), the WSDL files are designed for SOAP and therefore they are not suitable for validation of AMF and JSON. And, when developing a new AMF or JSON client, it would be odd to use WSDL files to find out how requests and responses should look like. Therefore, Enterprise Server 8 ships documentation that is independent of the protocol and does no longer require WSDL knowledge. For each interface (workflow, planning, admin, etc) there is an HTML page available containing full definition of all services and data structures. All definitions are hyperlinked to ease lookups. The documentation entry point can be found here:

http://localhost/Enterprise/sdk/doc/interfaces/index.htm

Protocol choice

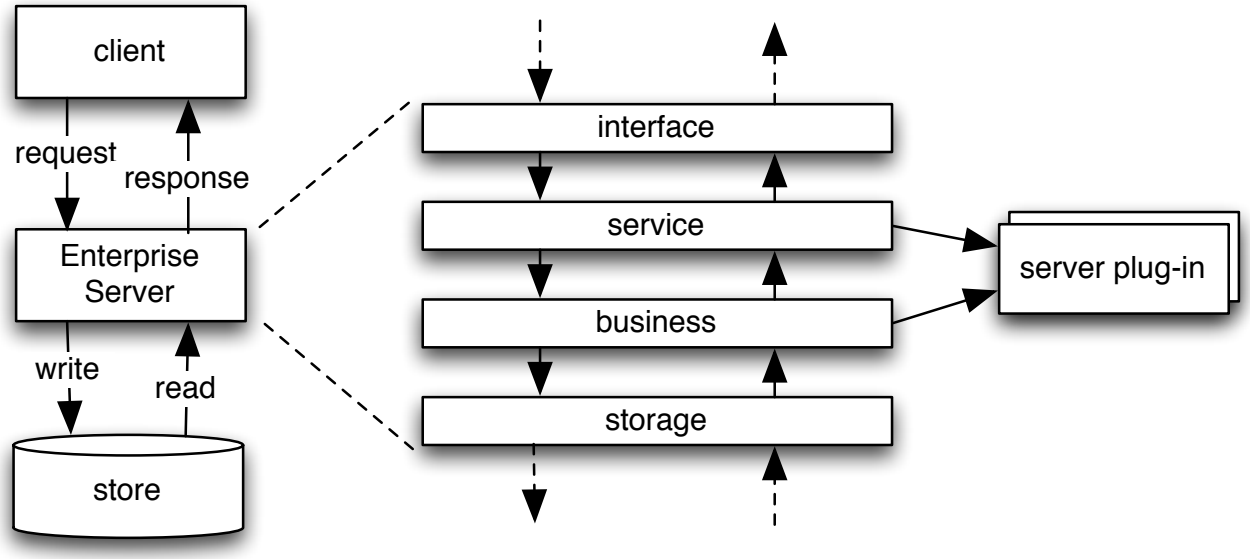
Besides performance, the ease of integration can be important too. For a JavaScript module running in web browser, it is hard to deal with SOAP due to lack of decent libraries. For a Flex client, SOAP is possible, but AMF is much more suitable. The figure below shows a bunch of programming languages with their most obvious choice.



See Client Application for more details about client applications integrating with Enterprise through specific protocols.

Protocol abstraction

On the server side, it is the responsibility of the interface layer to support protocols (SOAP, AMF and JSON). This is done for all interfaces (workflow, planning, admin, etc). In the figure below, you can see the concept of a client talking to the server. In the middle, the architectural layers of the server are shown. On the very top, where client requests arrive and the server responses depart, the interface layer is positioned. This layer unwraps the protocol notation of incoming requests and creates PHP request objects and data classes that are passed through the service layer underneath. On the way back, it wraps a protocol structure around the PHP response classes and data classes taken from the service layer before it gets sent out to client.



On the far right of the figure, it shows that server plug-ins are called from service- and business layers. This shows that server plug-ins do not have to know about the protocols; this is taken care of by the server (at the interface layer).

DIME and Transfer Server

The SOAP protocol is specified in the WSDL files. The workflow and planning WSDL files still specify DIME for some services, such as for the CreateObjects workflow service:

```
	<operation name="CreateObjects">
		<soap:operation soapAction="urn:SmartConnection#CreateObjects"/>
		<input>
			<dime:message layout="http://schemas.xmlsoap.org/ws/2002/04/dime/closed-
layout" wsdl:required="false"/>
			<soap:body use="literal"/>
		</input>
		<output>
			<soap:body use="literal"/>
		</output>
	</operation>
```

This does not imply that DIME -has- to be used. Instead, the File Transfer Server can be used, as explained in next chapter. For DIME, the Content element of the file attachment is used. To send files through the File Transfer Server, the FileUrl element is used. No matter which of the two methods is used, the server plug-ins use the FilePath attribute of the Attachment object. (Same for PHP web/admin applications.) The WSDL files show that FilePath and FileUrl are introduced since v8 as an alternative for Content:

```
<complexType name="Attachment">
<all>
        <element name="Rendition" type="tns:RenditionType"/>
        <element name="Type" type="xsd:string"/>
        <element name="Content" type="tns:AttachmentContent" ...
        <element name="FilePath" type="xsd:string" ...
        <element name="FileUrl" type="xsd:string" ...
</all>
</complexType>
```

For these three elements, there is always one in use while the other two are nullified.

Migration of Enterprise 7 integrations

The impact of the introduction of the new protocols (AMF and JSON) to your Enterprise 7 integrations are discussed in this chapter.

SOAP clients

There is no real need to migrate your SOAP clients to support the AMF or JSON protocols because there are no plans to drop the SOAP protocol. However, there are some reasons why you might want to consider AMF or JSON instead of SOAP:

- **AMF**: Because AMF is significantly faster than SOAP, you might want to migrate your SOAP client to AMF for the sake of performance. It is possible to mix both protocols to ease migration processes. So you can simply start with one service doing AMF while the others are still doing SOAP.
- **JSON**: Until today, there seems to be no mature SOAP library available for JavaScript (using Ajax). Building your integrations with Enterprise Server upon a shaky library could result into stability issues. A much better choice for this particular language is JSON.

If you need to upload or download files and you choose for AMF or JSON, you implicitly choose for the Transfer Server, since DIME is supported for SOAP only.

Server Plug-ins

Note that Server Plug-ins do not have to deal with the new protocols at all. SOAP, AMF and JSON requests and responses travel through the very same service classes using the very same request-, response- and data classes. And, no matter which file transfer technique clients are using (DIME or Transfer Server) the Attachment data class will have the FilePath filled.

Transfer Server [since 8.0]

Enterprise Server 7 (and earlier versions) supports the SOAP protocol through which files can be sent as DIME attachments. Because DIME has some performance- and integration disadvantages, Enterprise Server 8 supports a technique that can be used instead of DIME: the Transfer Server. This chapter explains how to *integrate* the Transfer Server. (How to *deploy* the Transfer Server can be found in the Admin Guide.)

Why no longer DIME?

Sending DIME attachments through SOAP implies that all request- and response data must be sent through one HTTP connection as one long binary stream. This has some important disadvantages:

- For remote workers (longer distances), the throughput of the traffic safe HTTP protocol drops significantly due to packet loss and round trip times.
- Sending file attachments along with messages, results in heavy memory consumption on both sides; client and server. A server claiming most of the memory to serve one client request could result into swapping out other server processes serving other clients. Memory swapping (to disk) leads to significant and unpredictable performance drops.
- DIME is superseded by MTOM and therefore it got more or less dropped by 3rd-parties (such as .NET and Java). This makes it hard for integrators to use such 3rd-party tools to build a client talking to Enterprise Server. However, MTOM has the same disadvantages as mentioned above. And, both protocols are bound to SOAP, which claims the need to introduce an alternative file transfer technique when introducing alternative protocols.

Introduction

Enterprise Server 8 can act as the Transfer Server. (How to set up and configure this is explained in the Admin Guide.) The Transfer Server handles file uploads and downloads. This way, files travel through a different HTTP connection than the service requests and responses. The Transfer Server by itself is already quite a bit faster than DIME and has a much lower memory footprint, but moreover it allows the clients to implement various performance improvements that really make a difference, such as:

- Upload/download files asynchronously and/or in parallel
- Skip download of locally cached files
- Report errors from the service -before- upload/download (large) files
- Use standard libraries (SOAP parsers, AMF, etc) that do not handle DIME

System administrators can set up the Enterprise Server on one machine and the Transfer Server on another. And, there can be multiple Enterprise Servers and multiple Transfer Servers, all working together serving a group of client machines. This way, advanced load balancing is possible with use of HTTP server dispatchers. The Enterprise Server tells clients which Transfer Server to use by returning the feature 'FileUploadUrl' in the LogOnResponse. (The value is taken from the HTTP_FILE_TRANSFER_REMOTE_URL option in the configserver.php file.)

The Transfer Server needs a folder to temporarily store files that are sent between clients and servers. This is called the Transfer Folder. Note that the Enterprise Server needs access to the temp files as well.

File operations / HTTP methods

The client uses HTTP requests to upload, download or delete files in the Transfer Folder. There is only one entry point http://<FileUploadUrl>/transferindex.php that accepts the following HTTP request methods to upload, download or delete a file:

- HTTP PUT: upload
- HTTP GET: download
- HTTP DELETE: delete

For clients that do not support these HTTP methods, they can use POST and give an additional parameter 'httpmethod' in the URL like this:

```
http://123.123.123.123/transferindex.php?...&httpmethod=DELETE
```

Regardless of the HTTP method used, each request must have a ‘fileguid’ and a ‘ticket’ parameter:

```
http://123.123.123.123/transferindex.php?fileguid=<GUID>&ticket=<TICKET>
```

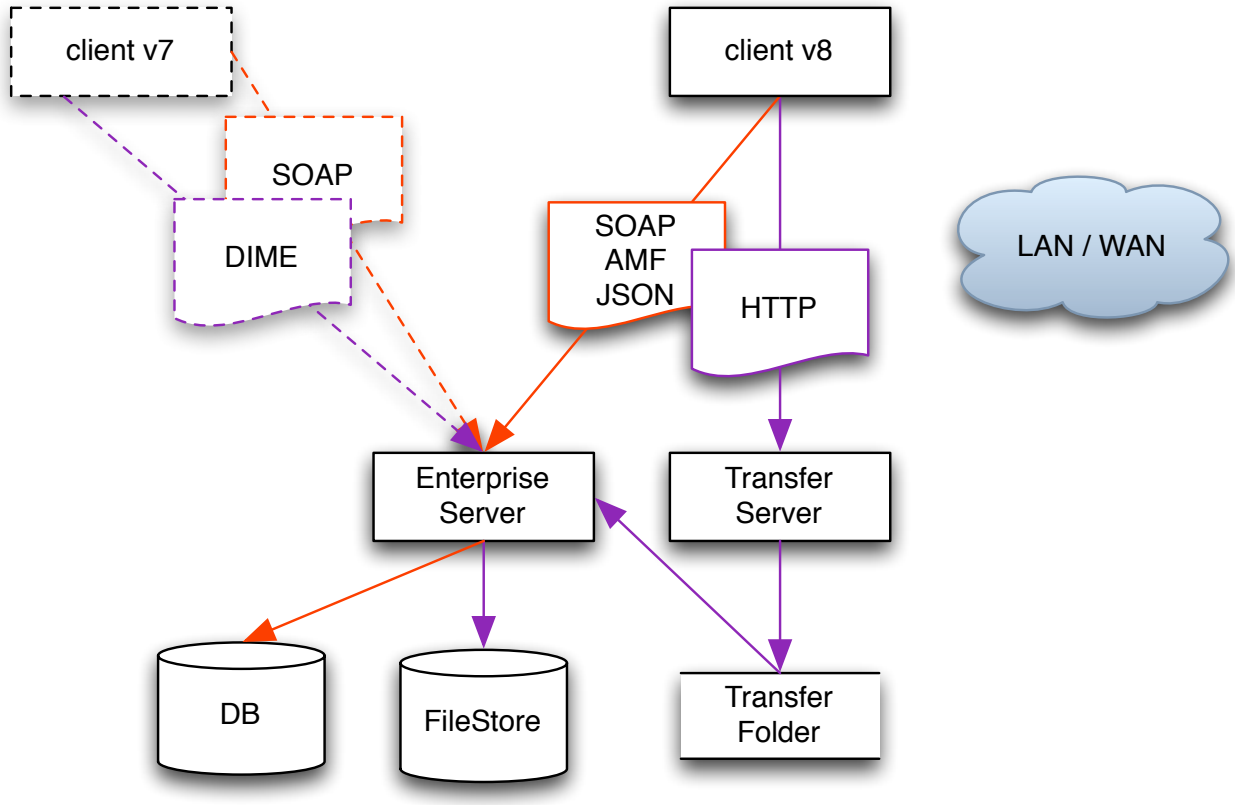
For file downloads, the ‘fileguid’ is already part of the file URL returned by the server through the GetObjectsReponse, but for file uploads clients need to create a new global unique identifier (GUID) in 8-4-4-4-12 format. For example:

```
690aebdf-93d1-ea99-6597-800994575d8c
```

The ‘ticket’ is returned by the server in the LogOnResponse, which needs to be picked up by clients. Only requests that send a valid ticket are handled by the Transfer Server. When the ticket is not valid, a HTTP 403 error is returned. Clients should check if the error message contains the “SCEntError_InvalidTicket” token. If so, the client should try to re-logon, and when successful, it should repeat the last operation with the Transfer Server (such as the file upload or download).

Uploading files

The figure below shows how file attachments travel along services when uploading files (such as the CreateObjects or SaveObjects workflow services). With v7 clients, they both travel through the same request and connection. With v8 clients, there are two connections; one to the Enterprise Server and one to the Transfer Server. In this case, clients can choose between SOAP, AMF or JSON. The red color shows how request data travels through the system. The file attachments are shown in purple.



When clients log in, they catch a new option named 'FileUploadUrl' from the FeatureSet element of the LogOn response. When the HTTP Transfer Server is configured, the value could look like this:

```
http://123.123.123.123/transferindex.php
```

For each file to upload, clients create a new global unique identifier (GUID) in 8-4-4-4-12 format. For example:

```
690aebdf-93d1-ea99-6597-800994575d8c
```

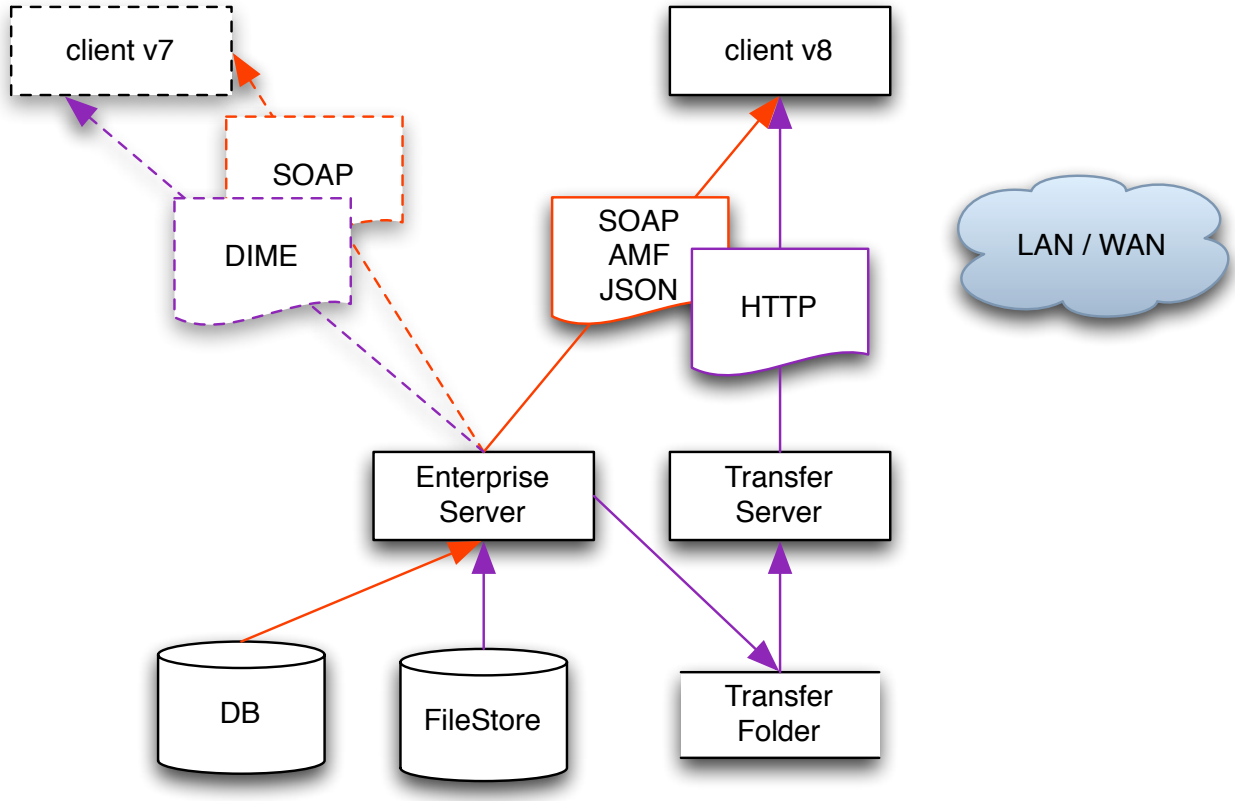
Files are then uploaded through a HTTP PUT command whereby the GUID is passed like this:

```
http://123.123.123.123/transferindex.php?fileguid=<GUID>&ticket=<TICKET>
```

Uploads are done chunk-wise to reduce memory consumption. Clients need to catch unexpected errors (HTTP 4xx / 5xx) and need to follow redirections (HTTP 3xx). When uploaded, the used URL must be filled in at the Attachment->FileUrl element in the service request.

Downloading files

The following figure shows how files are downloaded, such as for the GetObjects workflow service.



The server returns the file location through the service response in Attachment->FileUrl elements. Such URLs looks like this:

http://123.123.123.123/transferindex.php?fileguid=<GUID>&ticket=<TICKET>

When the client is using a 3rd-party widget that shows a preview directly (without saving locally), an optional parameter named “inline=1” can be given. (For example, this is used by the Content Station Web Edition.) Then, the Transfer Server returns through the HTTP response header the disposition attribute set to “inline” value. When the inline parameter is left out, the disposition is set to the “attachment” value, resulting in a Save-As dialog.

Additionally to the URL example given above, there is another parameter, named “format”. This contains the mime-type of the file to download. This parameter is required and provided by the server. It enables the Transfer Server to return the Content-Type field through the HTTP response header.

The client can check its local cache if this version of the file has already been downloaded before. If not, it retrieves the file through a HTTP GET command by following the given URL. Downloads are done chunk-wise to reduce memory consumption. Clients need to catch unexpected errors (HTTP 4xx / 5xx) and needs to follow redirections (HTTP 3xx).

Compression for up-/downloading files [since 9.5]

In order to reduce the upload and download time of files for remote users, the File Transfer Server can compress and de-compress files by using a file compression technique. Whether or not compression is actually used is defined by the client, while the File Transfer Server informs the client which compression techniques are available.

The compression techniques are returned in the LogOnResponse by a feature named ‘AcceptsCompressions’:

```
<ServerInfo>
    ...
    <FeatureSet>
        <Feature>
            <Key>AcceptsCompressions</Key>
            <Value>deflate</Value>
        </Feature>
    </FeatureSet>
    ...
```

In the above example, the client is informed that the 'deflate' compression technique can be used. When multiple techniques are defined, these are comma-separated. These techniques are built-in and therefore the features are provided automatically and they can not be configured. Currently, ‘deflate’ compression is supported only.

Remote workers and remote locations [since 9.5]

File compression should only be used for remote workers. To differentiate such workers from local workers, the options named ‘REMOTE_LOCATIONS_INCLUDE’ and ‘REMOTE_LOCATIONS_EXCLUDE’ of the configserver.php file are used. For a full explanation on how to use these options, see the comments in the configserver.php file.

In the LogOnResponse, Enterprise Server returns this as follows:

```
<ServerInfo>
    ...
    <FeatureSet>
        <Feature>
            <Key>IsRemoteUser</Key>
            <Value>true</Value>
        </Feature>
    </FeatureSet>
    ...
```

If the Value equals ‘true’, the client IP is treated as remote (according to the configuration options), else ‘false’ is returned.

When the ‘IsRemoteUser’ feature is set to ‘true’ and the ‘AcceptsCompressions’ feature lists the ‘deflate’ technology, the client could request the Transfer Server to compress a file (server side) while downloading as follows:

```
http://123.123.123.123/transferindex.php?…&compression=deflate
```

In the same manner, it could request the Transfer Server to uncompress the file (server side) while accepting a file being uploaded.

Regardless of the features returned in the LogOnResponse, it is up to the client to decide whether or not (and when) to apply compression. Please use compression with care. For example: it does not make sense to compress JPEG or ZIP because these files are already compressed. It would lead to taking up CPU processing without an actual reduction in file transfer time.

Content Station 9.5 uses Deflate compression when remote users are saving (uploading) or opening (downloading) WCML articles. This feature becomes affective when in the LogOnResponse the ‘IsRemoteUser’ feature is set to ‘true' and the ‘AcceptsCompressions’ feature lists the ‘deflate’ technology.

Deleting files

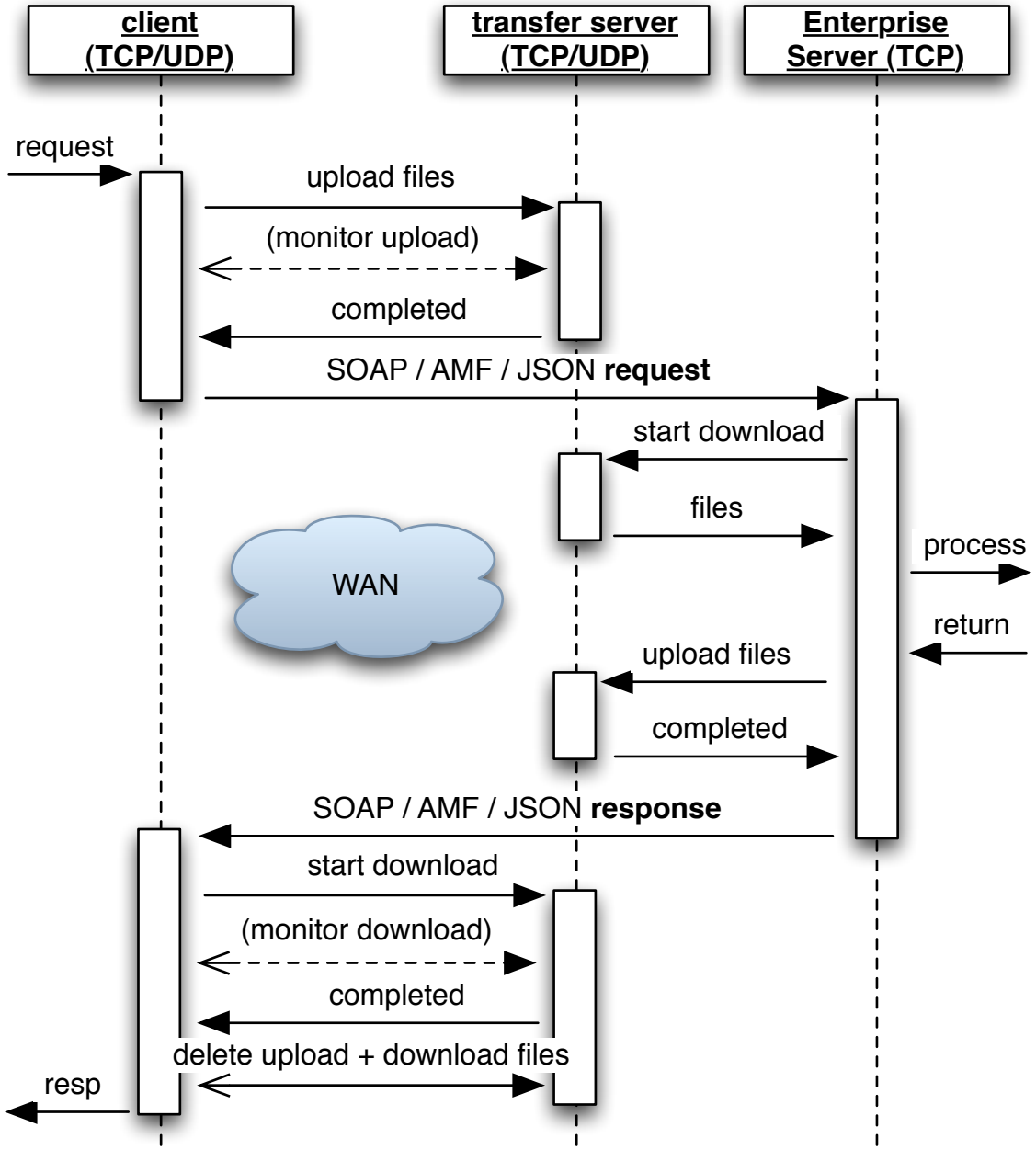
The separation of messages and file transfers can have big performance gains in all kinds of situations. This heavily depends on the smartness of the client application. For example, suppose the client does a CreateObjects request. In case of DIME, one message is sent that includes the whole native file. In case the create action fails, Enterprise Server returns an error (for example ‘Name already exists’). Once the end-user has corrected the error, the client has to send the whole message again, including the native file. Instead, when the client uses the Transfer Server to upload the native file, it can decide to leave the already uploaded file on the Transfer Folder when such an error occurs. This time, it just sends the user changed metadata again (to correct the error), while it leaves the native file untouched. This saves a lot of transfer/waiting time. After the CreateObjects service request is handled successfully, it sends a HTTP DELETE request to the Transfer Server to delete the uploaded file from the temporary Transfer Folder. Doing so, the file GUID is passed like this:

```
http://123.123.123.123/transferindex.php?fileguid=<GUID>&ticket=<TICKET>
```

It is the client application’s responsibility to clean files in the Transfer Folder. When there are files left behind, it is considered to be a client bug. In very exceptional situations, such as client crashes, there could be files left behind though. For those cases an Enterprise Server Job can be used to clean up the Transfer Folder. Nevertheless, clients should not rely on this feature since the folder might then rapidly grow, especially for large systems with many users.

Sequence diagram

Clients sending requests with file uploads first do the uploads and then the request. On download, they wait for the response and then start the download. The diagram below shows the interactions between client and server. Note that at the very end, the client cleans the files on the file transfer server. The reason why the server should not do this for the uploaded files is that when the request fails due to business logics (for example no access rights), the client could leave the files as-is, and try again with different parameters (for example a user choosing another Category). And, it enables clients to build async solutions in future too.



Connector interface classes

To support the new transfer methods changes had to made to the connector interfaces.

- Preview_EnterpriseConnector class:
  - since v8: generatePreview(**Attachment $attachment**, $max, &$previewFormat, & $meta, $bizMetaDataPreview)

before v8: generatePreview(**$format, $buffer**, $max, &$previewFormat, &$meta, $bizMetaDataPreview)

- MetaData_EnterpriseConnector:
  - since v8: readMetaData(**Attachment $attachment**, $bizMetaDataPreview)
    before v8: readMetaData(**$format, $buffer**, $bizMetaDataPreview)

Plugins implementing these connectors have to be changed and must respect the new function parameters.

How to handle attachments within your server plug-ins

As content is no longer directly stored within the attachment object, new functions are made available to read and write content. These functions are provided by the BizTransferServer.class.php module (located in <Enterprise>/server/bizclasses/). To call these functions, a BizTransferServer instance must be created first. As stated before, these functions can be used regardless the file transfer method.

Examples:

- ***read attachment:***

```
require_once BASEDIR . '/server/bizclasses/BizTransferServer.class.php'
$transferServer = new BizTransferServer();
$content = $transferServer->getContent( $attachment );
```

- ***write attachment:***

```
require_once BASEDIR.'/server/bizclasses/BizTransferServer.class.php';
$attachment = new Attachment('native', 'image/jpeg');
$transferServer = new BizTransferServer();
$transferServer->writeContentToFileTransferServer( $content, $attachment );
```

Handshake

“Do you speak English?”. That question is heard at touristic places quite often. But what answer do you expect from people who do not speak English at all? Introducing AMF and JSON brings similar challenges; How can a client start talking to a server without knowing what server it is talking to and what protocols it understands? This challenge is new with v8; Before, only SOAP and DIME were supported. But now, the listed servers (configured in WWSettings.xml or configserver.php) need to be accessed with care, before clients start talking new protocols. Clients have no idea what server version they start talking to since they can not look into the future.

Choosing the best protocol and file transfer

To find out, there is a very lightweight handshake done between client an server. The client fires an empty HTTP request to the entry point with a new handshake parameter. For example this is what a client sends to the workflow entry point:

http://123.123.123.123/transferindex.php?handshake=v1

This tells that the client understands (version 1 of) the handshake. Enterprise Server 7 does not support handshakes and returns the logon page (HTML). This is not XML, so the client can safely assume DIME over SOAP needs to be chosen. When the client does not support it, it should raise an error that it is not compatible with the selected server.

Enterprise Server 8 (and later) will return a home brewed XML structure like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<EnterpriseHandshake version="1">
	<Techniques>
		<Technique protocol="AMF" transfer="HTTP"/>
		<Technique protocol="SOAP" transfer="HTTP"/>
		<Technique protocol="JSON" transfer="HTTP"/>
		<Technique protocol="SOAP" transfer="DIME"/>
	</Techniques>
	<Interfaces minVersion="7" maxVersion="8">
		<Interface name="Administration" type="core"/>
		<Interface name="AdminDataSource" type="core"/>
		<Interface name="DataSource" type="core"/>
		<Interface name="Planning" type="core"/>
		<Interface name="Publishing" type="core"/>
		<Interface name="Workflow" type="core"/>
	</Interfaces>
</EnterpriseHandshake>
```

This tells the client that:

- AMF is preferred above SOAP
- AMF can be combined with HTTP, but not with DIME
- SOAP can be combined with HTTP or DIME
- SOAP over HTTP is preferred above SOAP over DIME

The client knows what protocols and transfers it supports (by itself) and now picks the best match that is most preferred by the server. Now it reconnects to the entry point again to let server know its choice, for example:

http://123.123.123.123/transferindex.php?protocol=AMF&transfer=HTTP

When the client does not connect this way (leaving out the protocol and transfer parameters), the server assumes the client is v7 (or older) and starts using DIME over SOAP for backwards compatibility reasons.

For each request, the client may choose other parameters. For example, it might support AMF, but still does SOAP for some specific services that have not been ported to AMF yet.

Note that the parameters can be applied to all public interfaces: workflow, planning, administration, datasource, datasource admin and publish. Internal application interfaces support SOAP only.

In theory, for each interface used by clients, there should be a handshake. However, there is no reason to support other protocols (for one or the other) from a server point of view. Clients may reuse the returned handshake data assuming it is the same for all interfaces.

Listing the configured servers

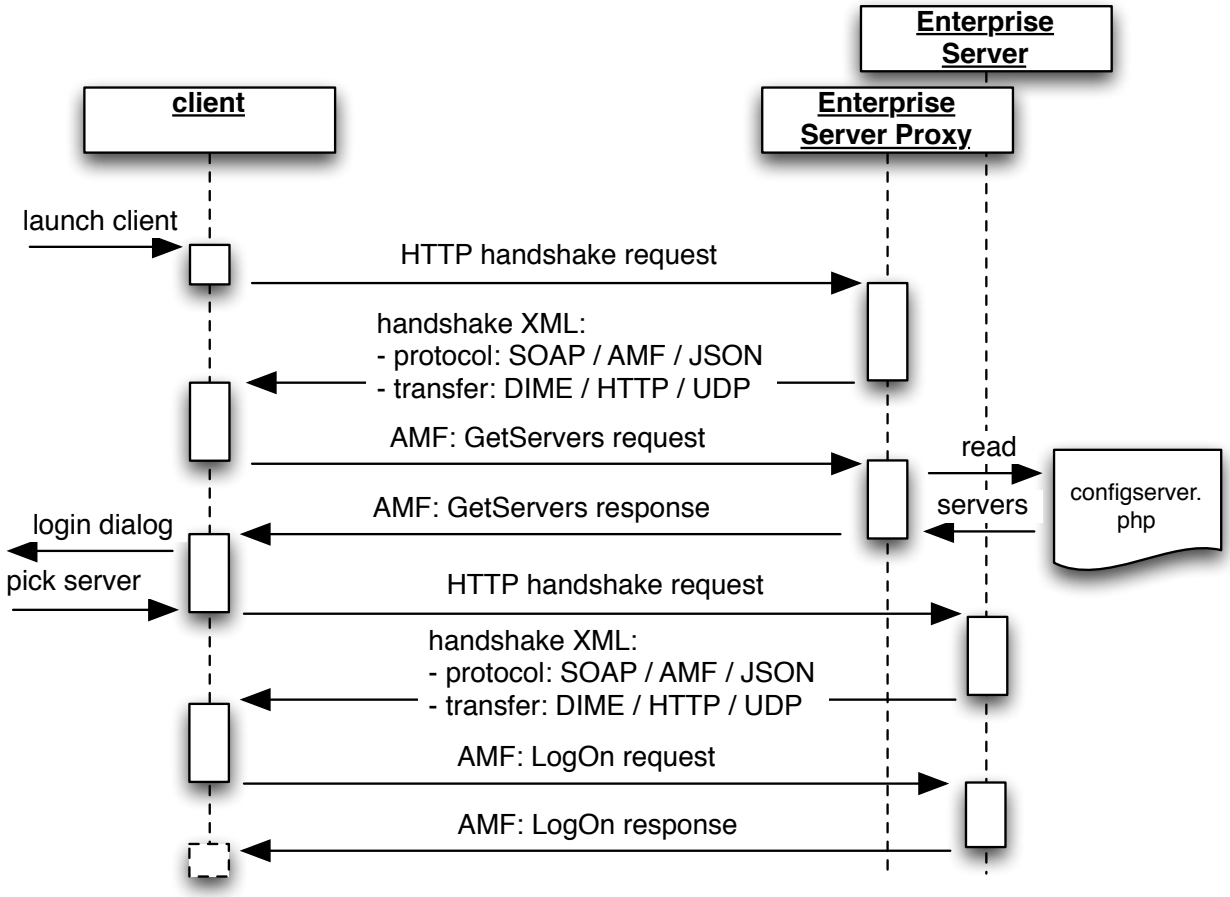
First, the list of servers needs to be determined. When there are one or more <SCEnt:ServerInfo> elements below the <SCEnt:Servers> element, it means that the list of servers is configured client side (WWSettings.xml) and so they are shown at login dialog right away without server interaction.

But, when there is one <SCEnt:ServerInfo> element -directly- below the <Settings> element in the WWSettings.xml file, it means that the list of servers is configured server side (configserver.php). Before talking to the server, clients do a handshake. This determines the protocol and the file transfer technique. When that is known, the client calls the server through a GetServers service to get the list of server to show in the login dialog.

Now the user picks one of the listed servers and attempts to log in. Before talking to the selected server, clients do a handshake. After choosing the best protocol and file transfer, clients set the protocol and transfer parameters to the URL and fire a LogOn request.

Sequence diagram: server-side config

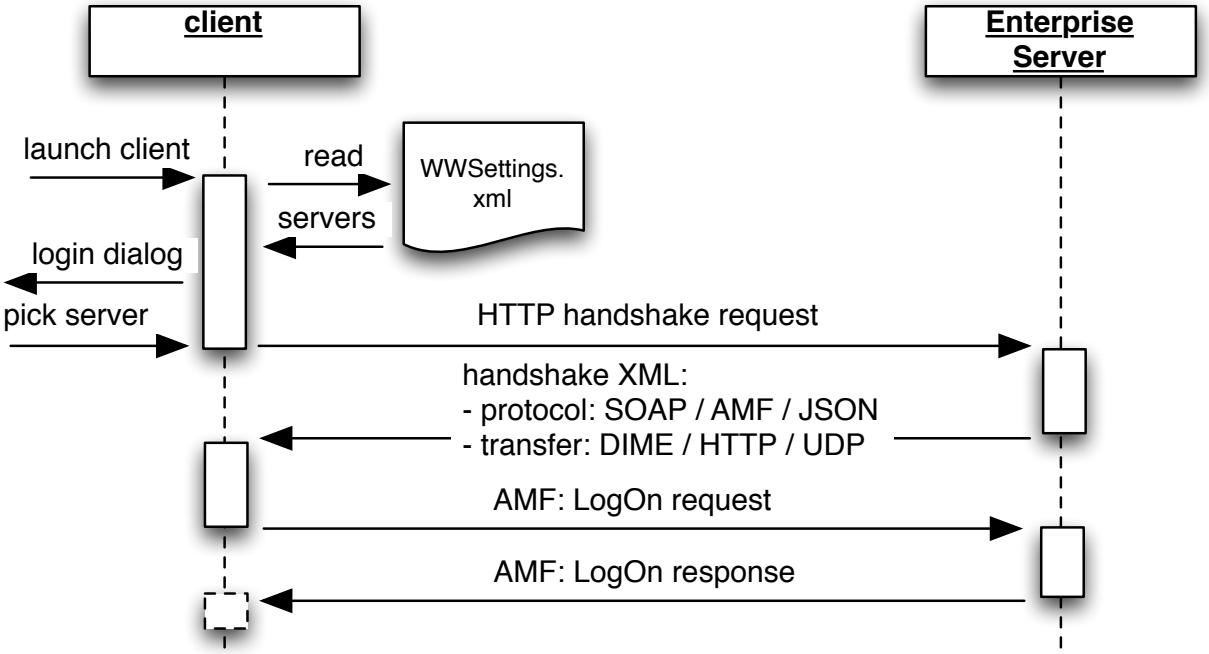
When servers are configured server-side, this is the sequence dialog of the handshake:



Because there can be a mix of server versions, clients needs to do the second handshake too; It could be the case there is a v8 server listing a v7 server or the other way around.

Sequence diagram: client-side config

When servers are configured client-side, this is the sequence dialog of the handshake:



Migration of Enterprise 7 integrations

Changes made to Enterprise Server 8 are done with care; The impact to clients and Server Plug-ins is reduced as much as possible. Nevertheless, SOAP clients are encouraged to migrate and Server Plug-ins (that are dealing with file attachments) are forced to migrate.

SOAP clients

Although DIME is still supported for backwards compatibility reasons, SOAP clients are strongly encouraged to move along with the server and start using the File Transfer Server. Note that DIME might get dropped with v9.

Server Plug-ins

To reduce memory consumption, the way file content is passed on through your Server Plug-ins has been adapted (by Enterprise Server). You need to adjust your plug-ins only when they use the Attachment or SOAP_Attachment data objects. The SOAP_Attachment is no longer used. The Attachment has no longer the Content element set (carrying the whole file in memory). There is a new element named FilePath from where you can read. Best is to leave the file on disk. Or else, try to read chunk-wise to avoid memory consumption. Reading the whole file in memory should be avoided, especially for native file renditions.

Web Service Interfaces

This chapter describes all flavors of the Web Service interfaces exposed by Enterprise Server. For each flavor, its primary purpose is given as well as how it needs to be accessed by client applications.

Flavors and their purposes

Enterprise Server implements and exposes the following Web Service interfaces, each for a different purpose:

Workflow interface

This is the most advanced and heavily used interface of all. It defines all supported editorial workflow operations such as creating, saving and deleting files. Through this interface, editors and layout designers typically manage their articles, Dossiers, layouts, etc. This interface is called by all clients; InDesign, InCopy, Content Station and the Web Editor.

Admin interface

Used for various Brand- and user administration purposes, such as creating Publication Channels for Brands, creating users and assigning them to user groups, etc. These operations are only accessible to Brand- or system administrators. (With Enterprise 7 the interface is not complete yet. For example, you cannot give users access to a Brand and you cannot set up a workflow definition. So, you can set up a basic Brand structure from scratch, but it requires manual completion.) The interface is mainly used by the admin web applications. It could also be very useful for third-party integrators, such as creating many Issues simultaneously for a specific Publication Channel.

System Admin interface [since 9.0]

Used for installation purposes and for technical administration (other than brand- or user administration). Note that the name of this interface does not imply that all its services require system admin access rights. For example, end-users in the workflow system can request which sub-applications are configured for them (when the Elvis Server plug-in is activated, this service tells whether or not the Elvis app should be shown in Content Station for the current user.) Candidate services that could appear in future include Feature Access Profiles, Server Plug-ins, Server Jobs, Server Options.

Planning interface

Through this interface, planning integrations with third-party systems are made, such as Journal Designer (DataPlan) and Timone (Tell). Its main goal is to simplify the workflow interface for plan systems and to offer some planning logics. Through this interface, planned pages and adverts (created in the planning system) are sent to the Enterprise database. InDesign is then able to automatically apply the planned information at the moment layouts are being opened. Planning systems create pages based on layout templates.

Publishing interface

Dossiers can be published to many channels through this interface. This is for instance used for Web CMS and SMS integrations (built with Server Plug-in technology). Previews of the published content can be requested. When content is outdated it can be unpublished to remove it from the channel again. This interface is mainly called by Content Station.

Data Source interface

Any external storage carrying structured catalog data can be integrated with Enterprise. Its

data can be queried through this interface to gain centralized access for client applications. The storage itself can be any database flavor, comma-separated value lists, or any other data source. The data can represent anything, such as a product to appear in catalogs, or programs to appear in TV-listings. Enterprise Server implements the interface and requests Server Plug-ins (containing a DataSource connector) to establish a connection to the external storage and query its structured data. Smart Catalog Enterprise is required which calls the interface to query the structured data and automatically build any catalog on paper within InDesign.

Definitions and entry points

The table below can be used to look up definitions and entry points for all interface flavors. The second column shows the abbreviation (short name) used internally by Enterprise Server. For example, in the .../Enterprise/server/interfaces/services and .../Enterprise/server/service folders you can find subfolders with such abbreviations. The third column shows three items of which their meaning is explained below the table. The last column shows the Enterprise version in which the interface was officially introduced.

interface	short	WSDL(1), URN (2), web service entry point (3)	since
workflow	wfl	SCEnterprise.wsdl urn:SmartConnection .../Enterprise/index.php	v3
admin	adm	SmartConnectionAdmin.wsdl urn:SmartConnectionAdmin .../Enterprise/adminindex.php	v5
system admin	sys	SystemAdmin.wsdl urn:SmartConnectionSysAdmin ../Enterprise/sysadminindex.php	v9
planning	pln	SmartEditorialPlan.wsdl urn:SmartEditorialPlan .../Enterprise/editorialplan.php	v4
publishing	pub	EnterprisePublishing.wsdl urn:EnterprisePublishing .../Enterprise/publishindex.php	v6
data source	dat	PlutusDatasource.wsdl urn:PlutusDatasource .../Enterprise/datasourceindex.php	v6
admin data source	ads	PlutusAdmin.wsdl urn:PlutusAdmin .../Enterprise/datasourceadminindex.php	v6

(1) WSDL

The Web Service Definition Language (WSDL) is a document which defines an interface. This is the communication contract between client and server. It lists the server operations (Web Services) exposed to the clients and defines the SOAP data structure sent through

client requests and received through server responses. The Enterprise WSDL files are located in the .../Enterprise/server/interfaces folder. This folder contains more WSDL files than listed in the table. Those are for internal use and should NOT be used for integrations.

(2) URN

The Uniform Resource Name is used to separate the interfaces from each other. For example, a User entity is defined for the workflow- and the admin interface. The one in the admin interface is more detailed. To avoid class name clashes, the URN is used as prefix, for example AdmUser. Only for the workflow interface, those prefixes are not applied (this is considered the 'main' flow which is mostly used).

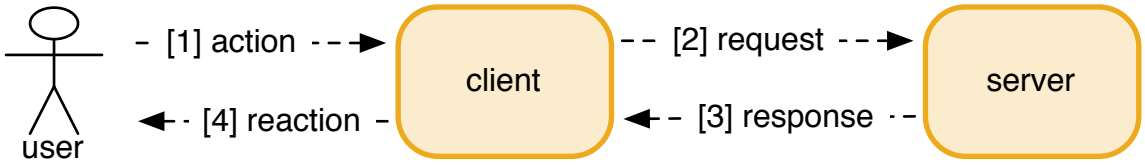
(3) web service entry point

A PHP index file used by client applications to connect to. This can also be used to request the WSDL file, which is done as follows: index.php?wsdl. It is recommended to use this method and to parse the received WSDL file client-side. WSDL files should never be read from the .../Enterprise/server/interfaces folder because HTTP read access could be denied. But also, the server sets the soap:address element on-the-fly before sending the WSDL back to client.

Inspecting client-server traffic

Once you have read this guide in its entirety, you might wonder how operation sequences (should) take place, or what data structures are sent back and forth between clients and server. Playing with existing clients while server DEBUG logging is enabled could give you a quick starting point and allows you to examine the process that occurred. This could be helpful when you want to create a new client application from scratch, or let your client support new features that are exposed by a freshly released Enterpriser Server (or hook into running services by using Server plug-ins, which is out-of-scope for this guide). This section tells how this can be done.

In the figure below a scenario is shown in which a user is working with any of the client applications, such as InDesign or Content Station. Whenever an action [1] is done that requires the help of the server, the client fires a Web Service request [2] to the server. After processing, the server returns the results through a Web Service response [3] back to the client. The client mostly gives some kind of reaction [4] informing the user that the operation was successful (or not) such as a dialog, a document that gets opened, or simply a spinning wheel that has stopped.



Playing around is obviously not recommended to do on a production server. There can be cases though where you simply want to examine what happens on a very rich data set, typically a production or demo server. When you are not the only person working on that server, enabling system wide server logging logs all operations of all users. This cacophony of operations will blur the logging you are looking for, as fired by your client only. Therefore, improvements have been made in this area since Enterprise Server 8.

Improved logging [since 8.0]

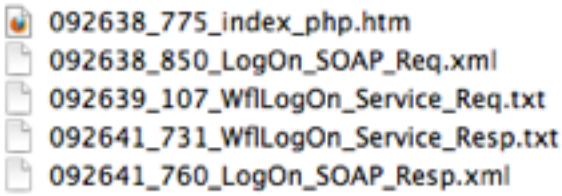
Since Enterprise 8, logging can be enabled per client IP, and for each client IP a separate logging folder is created. (See DEBUGLEVELS option described in the Admin Guide how

to configure.) You can add your client IP with a DEBUG value so that only for your client application all details are logged.

The following steps show how to inspect the Web Services traffic:

1. Enable DEBUG logging on the server. (See DEBUGLEVELS option in the Admin Guide.)
2. Set the LOG_INTERNAL_SERVICES option to *true* in the configserver.php file.
3. Remove any existing logging from previous sessions.
4. Perform some actions, such as login, create document, check-in document, run a query, etc.
5. Have a look in the log folder. The Web Service requests and responses are logged, all in separate files. Request files have a "_Req" postfix and responses a "_Resp" postfix.

For example, logging of a LogOn service call could look like this:



The name of each HTML log file refers to the PHP file that was used to handle a Web Service or web application. In this case, the index_php.htm file refers to the index.php file. This PHP file can be looked up in Definitions and entry points and when found, it means that a Web Service was handled. In this case it is found that the workflow interface is used. When not found, it could be any of the web applications. The index_php.htm file contains all logging of the Web Service call handling, in this case the LogOn.

The LogOn_SOAP_Req.xml file tells that the client has used the SOAP protocol for the LogOn request. The server returns the corresponding response (also in SOAP format) as logged in LogOn_SOAP_Resp.xml. Drag & drop them into a web browser (or XML editor) to view the structured data in readable format.

Because the LOG_INTERNAL_SERVICES option is enabled, the WflLogOn_Service_Req.txt and WflLogOn_Service_Resp.txt files are logged as well. This contains the data model of PHP that is built from the request or response. Note that clients could use old structures that are still supported by the new server (for backwards compatibility). The SOAP log contains the structure exactly how it was communicated between client and server (in either the old or the new structure). The TXT files contains only the new structure (old structure is mapped to new structure by the interface layer of the server). This is done to let the core server and the server plug-ins work with new structures only, and not to worry them with both structures (old and new).

Notes

- Attached documents (DIME attachments) sent along with SOAP traffic are not logged server side. This is to avoid outrageous disk space consumption. InDesign/InCopy clients support DIME logging though.

For SOAP and JSON the original request is logged. Because AMF is a binary format, this is not done for AMF. In such cases you have to rely on the PHP objects logged in TXT format; this can be enabled with the LOG_INTERNAL_SERVICES option.

Service validation [since 8.0]

Since Enterprise Server 8 it is possible to validate Web Service requests and responses. This is implemented for all supported protocols and interfaces. This feature can be enabled with the SERVICE_VALIDATION option in the configserver.php file. It is recommended to enable it when you are developing a client that integrates with Enterprise. When a request is found to be invalid, this is a client bug and so please check the SDK documentation to fix the problem. When a response is not valid, it must be a server bug and so please report it to WoodWing Support. To continue your developments in the meantime, please add the path reported to be invalid at the SERVICE_VALIDATION_IGNORE_PATHS option in the configserver.php file to suppress the validation error. This allows you to keep the validation enabled, which is most important during development and tests.

Understanding SOAP using a WSDL

There are lots of published documents and books explaining all the ins- and outs of WSDLs in great detail. Much of the WSDL technology (features and options) is simply not used by Enterprise. As a starting point, this paragraph takes a login response (from the server SOAP log folder) as an example and explains how you can look up its data structure. This method can be applied to all client requests and server responses.

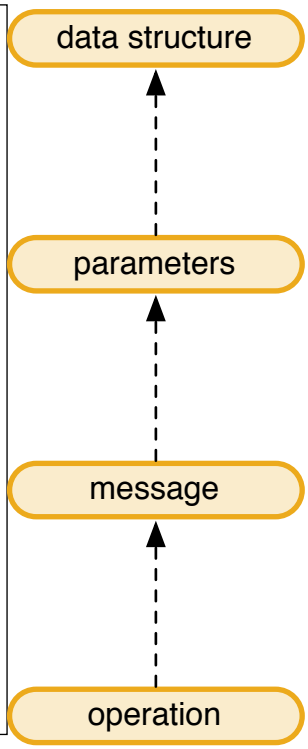
```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
      xmlns:SOAP-ENV="http://schemas.../"
      xmlns:xsi="http://www.w3.org/..."
      xmlns:xsi="urn:SmartConnection">
<SOAP-ENV:Body>
      <LogOnResponse>
            <Ticket>...</Ticket>
            <Publications>
                  <PublicationInfo>
                        <Id>1</Id>
                        <Name>WW News</Name>
                        ...
                  </PublicationInfo>
            </Publications>
            ...
      </LogOnResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In the following steps, used colors refer to the figure shown to the right:

1. Enable DEBUG logging.
   (See the Enterprise Admin Guide about how to enable this.)
2. Login with client, which then fires a LogOn request, as logged on the server.
3. Open the SOAP log file, that can be found in the server log folder.
4. Determine the used **URN** and look up the WSDL file (by using the table discussed in the previous chapter). In this example, you will find the SCEnterprise.wsdl file.
5. Determine the **operation** (fired by the client) and use this as a starting point to look up the **parameters** and **data structure** in the WSDL (as looked up). This is explained below in more detail.

Open the found SCEnterprise.wsdl file in a web browser and start reading from the bottom up. The colors correspond with the above. Underlined items are type definitions that should be looked up (also from the bottom up). All data structure items mentioned can be found this way:

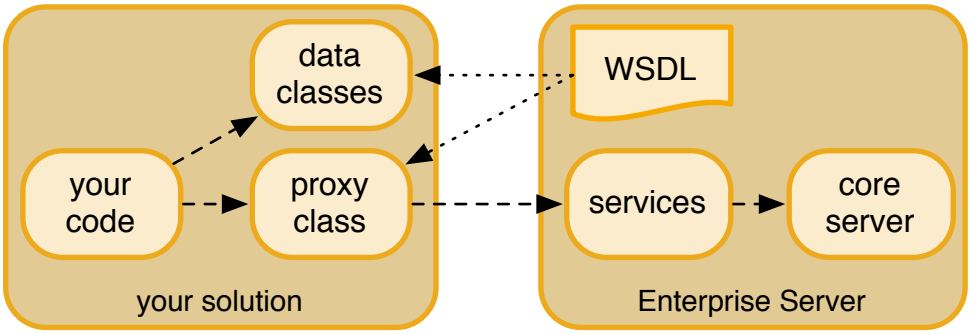
```
<complexType name="PublicationInfo">
      <all>
            <element name="Id"    type="xsd:string"/>
            <element name="Name" type="xsd:string"/>
            ...
      </all>
</complexType>
<complexType name="ArrayOfPublicationInfo">
      <sequence>
            <element name="PublicationInfo" ...
                              type="tns:PublicationInfo"/>
      </sequence>
</complexType>
<element name="LogOnResponse">
  <complexType>
      <sequence>
            <element name="Ticket"  type="xsd:string"/>
            <element name="Publications"
                        type="tns:ArrayOfPublicationInfo" .../>
            ...
      </sequence>
  </complexType>
</element>
<message name="LogOnResponse">
      <part name="parameters" element="tns:LogOnResponse"/>
</message>
<operation name="LogOn">
      <input message="tns:LogOn"/>
      <output message="tns:LogOnResponse"/>
</operation>
```



Client Applications

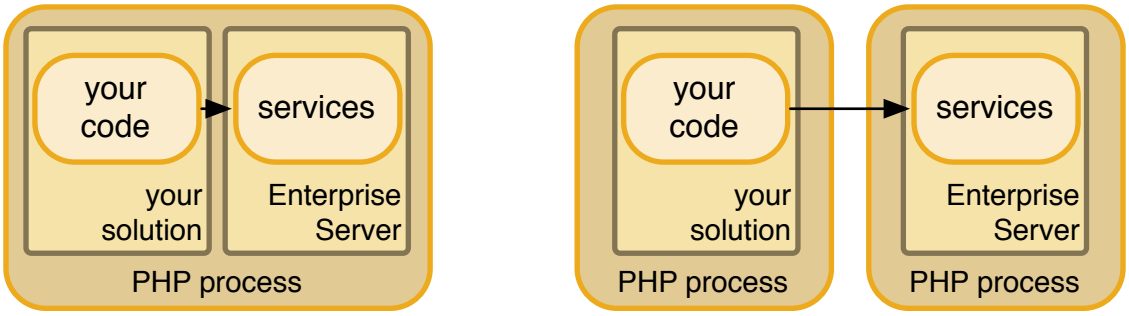
The Web Service interfaces are accessible in different ways. The way to choose depends on the programming language used to develop your application and the way it needs to be deployed. Most common programming languages today support Web Services, including PHP, Java, RealBasic, VisualBasic, C++, C#, Flex and many more. Enterprise Server implements its interfaces with PHP. When your solution is written in a programming language other than PHP, you need to develop a client application that talks Web Services as specified by the WSDLs of Enterprise Server.

Typically the WSDL file is read by your SOAP library/tool to generate a proxy class (from its operations) and a bunch of data classes (from its type definitions). This way, you can build a request data structure based on the generated data classes, and fire the request through the proxy class by simply calling one of its methods (operations). The proxy takes care of building the SOAP request itself and parsing the returned SOAP response. In other terms, all SOAP challenges are encapsulated and hidden from your solution.



Enterprise WSDLs refer to the DIME attachment specification to send documents along with SOAP traffic. DIME is required up to and including Enterprise Server 7. Since Enterprise Server 8 it is possible to move away from DIME and use the Transfer Server instead to send documents directly over HTTP. This is faster in execution and easier to integrate than DIME and therefore clients are encouraged to use the Transfer Server.

When your solution is written in PHP, you can use helper classes shipped with Enterprise Server. When your solution -always- runs on the -same- server machine (as the Enterprise Server) you can use the PHP service helper classes. This way, your solution is executed by the very same process as the application server, as shown on the left side of the figure below. When it needs to be installed on other server machines though, you need the PHP SOAP helper classes. In this situation, whether it runs on the same(!) machine or not, your solution runs in a separate process, as shown on the right side of the figure below. In this case, there is a little overhead of SOAP traffic.



.NET clients and Java clients using SOAP

Enterprise 7 has changed the standard notation used for arrays in the WSDL files. (See also Arrays.) As a result, .NET and Java integrations could no longer use the WSDL files shipped with the server, and the array definitions needed to be manually changed back to the 'old' notation.

For example, this was the 'old' notation used for Enterprise 6 (and older):

```
<complexType name="ArrayOfUser">
	<sequence>
		<element name="User" minOccurs="0" maxOccurs="unbounded" type="tns:User"/>
	</sequence>
</complexType>
```

while this is the 'new' notation used for Enterprise 7 (and later):

```
<complexType name="ArrayOfUser">
	<complexContent>
		<restriction base="soap-enc:Array">
			<attribute ref="soap-enc:arrayType" wsdl:arrayType="tns:User[]"/>
		</restriction>
	</complexContent>
</complexType>
```

To solve this incompatibility, Enterprise 8 has introduced an option that automatically converts any requested WSDL file from the new into the old array notation on-the-fly.

.NET clients

When importing WSDL files in your .NET project, please add the ws-i value to the wsdl parameter to trigger that conversion, for example:

```
http://localhost/Enterprise/index.php?wsdl=ws-i
```

At the time of updating this document with Enterprise 8 info, a demo application is being developed in .NET showing how to integrate with Enterprise (which uses this conversion method). Once completed, the application will appear on Labs.

Java clients

Enterprise Server 8 makes it easier to integrate with Java clients. Java classes are generated from the WSDL and shipped in the SDK folder at the following location:

```
Enterprise/sdk/java/src
```

This is done for all data structures (simpleType), requests and response classes. Whenever changes are made to the WSDL, regenerated classes are shipped with the server. Therefore make sure to reimport and compile the classes in your Java project to use the latest features.

From Enterprise 8.3.2. / 9.0.2 onwards, essential fixes have been made to the Java classes shipped in the SDK folder. These fixes solve fatal errors in your Java client. For example

```
org.xml.sax.SAXException: Invalid element in
com.woodwing.enterprise.interfaces.services.wfl.PublicationInfo -
PublicationInfo
```

The shipped Java classes with Enterprise Server 8.0.x, 8.1.x, 8.2.x, 8.3.0, 8.3.1, 9.0.0 and 9.0.1 cannot be used. For those versions it is needed to regenerate the classes yourself. That can be done with the WSDL2Java tool from Apache Axis:

```
java org.apache.axis.wsdl.WSDL2Java -u -W -p
com.woodwing.enterprise.interfaces.services.wfl http://localhost/
Enterprise/index.php?wsdl=ws-i -o out
```

Note that the special "wsdl=ws-i" parameter triggers Enterprise Server to change the array definitions into WS-I notation for the requested WSDL (as explained earlier in this chapter).

Support for this parameter is introduced in Enterprise 8. Earlier versions do not ship the Java classes, but you could generate them yourself as written above. For Enterprise 7 you could copy the WSDLs and manually adjust all the array definitions (as explained earlier in this chapter) before you generate the Java classes.

A “HelloWorld” sample application made in Java is available from WoodWing Labs, showing how to logon to Enterprise Server through the SOAP workflow interface by using the generated Java classes. Use this sample as a starting point.

Flex clients using AMF [since 8.0]

To ease Flex integrations, Enterprise Server 8 ships request-, response- and data classes for all interfaces. For each class it provides an Action Script, which can be found here:

```
.../Enterprise/sdk/flex/src/com
```

When developing an AMF client that integrates with Enterprise Server, please import these classes into your Flex project as a starting point. This can be done for one, many or all interfaces, depending on which interface is most suitable for your integration. When integrating a newer (updated) version of Enterprise Server, reimport those classes again, since the classes shipped with Enterprise are always respecting the latest WSDL changes.

Let’s use the LogOn request as an example. The LogOn request class (Action Script) for the workflow interface can be found here:

```
.../Enterprise/sdk/flex/src/com/woodwing/enterprise/interfaces/
      services/wfl/WflLogOnRequest.as
```

The login dialog could be made like this:

```
<mx:Panel title="Login">
      <mx:TextInput id="userName"/>
      <mx:TextInput id="password"/>
      <mx:Button label="Login" click="doWflLogOn()"/>
</mx:Panel>
```

A proxy object to the workflow interface could be made like this:

```
<mx:RemoteObject id="wflProxy" destination="wflDestination"
            showBusyCursor="true" source="WflServices">
      <mx:method name="LogOn" result="onWflLogOnResult(event)"
            fault="handleException(event)"/>
</mx:RemoteObject>
```

The login script (behind the login dialog) could look as simple as this:

```
private function doWflLogOn():void {
      var req:WflLogOnRequest = new WflLogOnRequest();
      req.User = userName.text;
      req.Password = password.text;
      wflProxy.LogOn( req );
}
```

Once logged in, Flex gets called back, from where you can pickup data, like the ticket:

```
private function onWflLogOnResult(evt:ResultEvent):void {
      Alert.show(evt.result.Ticket.toString(),
            "Got a workflow ticket!");
}
```

At the time of updating this document with Enterprise 8 info, a demo application is being developed in Flex showing how to integrate with Enterprise with AMF. Once completed, the application will appear on Labs.

JavaScript clients using JSON [since 8.0]

When developing a JavaScript client that integrates with Enterprise Server, JSON is the most suitable and recommended protocol to use. The basic concept is as follows: Some JavaScript functions can be built into an HTML web page. Instead of refreshing / reloading the entire page on user submits, the web page keeps loaded and the JavaScript functions

communicate with the server in the background. Ajax technology is used to fire JSON RPC requests over HTTP to Enterprise Server. Technically this is done asynchronously to keep the web browser responsive while waiting for the server's response (but logically it could feel synchronous with the end-user waiting for a response too). When the response arrives, a registered callback function in JavaScript is triggered, which allows the client to continue its procedure. With all this in place, we speak of a JavaScript client application.

All operations specified in any of the Web Service Interfaces can be used with JSON. But unlike SOAP and AMF, for JSON there are no request or data classes shipped. This is with a reason: It would lead into downloading all class definitions from server to client before the client can start talking. This significant overhead is something to avoid for the sake of performance (and there is no technical need to have such classes in place). The communication using JSON is very lightweighted and data properties that are not known by client or server are simply not sent over HTTP. Basically, everything that is defined as nil / nullable can be simply left out (unlike SOAP and AMF that have to specify the property being set to nil) which reduces communication traffic.

For example, this is how a LogOn request looks like:

```
{
	"jsonrpc":"2.0",
	"method":"LogOn",
	"params":[{
		"User":"woodwing",
		"Password":"ww",
		"ClientName":"localhost",
		"ClientAppName":"my demo web app",
		"ClientAppVersion":"8.0",
		"RequestTicket":true
	}],
	"id":2
}
```

On the server, arrived JSON classes are mapped onto PHP classes like done for SOAP and AMF. Therefore the client application should specify the class names so that Enterprise Server can do the mapping. (This is not needed for the request class itself since it is already specified in the "method" parameter the JSON RPC structure.) Enterprise Server expects the first property of an object to be named "__classname__" that caries the name of the data class (which is the same as the complexType as specified in the WSDL file). For example, the GetPagesInfo request takes an Issue and an Edition data classes, which therefore must be specified, as done by the red marked fragments:

```
{
	"jsonrpc":"2.0",
	"method":"GetPagesInfo",
	"params":[{
		"Ticket":"c1a0844ep03hRxZAvy5SE7mGmBEsHHCTVBzVmAhC",
		"Issue":{
			"__classname__":"Issue",
			"Id":"2"
		},
		"IDs":{},
		"Edition":{
			"__classname__":"Edition",
			"Id":"1"
		}
	}],
	"id":4
}
```

At the time of updating this document with Enterprise 8 info, a demo application is being developed in JavaScript showing how to integrate with Enterprise with JSON (using jQuery). Once completed, the application will appear on Labs.

Note that the demo client application includes some JavaScript modules that takes care of the low level communication details written above, except for the "__classname__" property that needs to be set by the client. Basically those classes can be used for any integration using the JSON RPC 2.0 standard. However, they are enriched with some error handling to simplify clients working with exception handling. Therefore, when developing your own client application, please copy the JavaScript classes included by the demo.

You client application (HTML page) should have the very same web root location as Enterprise Server. For example, when the client application accesses the server through the "http://localhost/enterprise/index.php" entry point, then the client application should reside under the "http://localhost/" web root as well. Doing so, Make sure the URL in your web browser address bar is -exactly- the same as the one used in jQuery.Zend.jsonrpc(...) in the example scripts. Note that even "localhost" and "127.0.0.1" do NOT match.

PHP clients using SOAP

Especially for backend integrations PHP could be used to integrate Enterprise Server. It is possible to integrate with Enterprise Server 'internally' or 'externally'. It is important to categorize your client in one of the two since it has a big impact on the whole solution.

Internal PHP integrations are custom PHP modules triggered by a custom web application (or crontab) that include Enterprise Server directly, or Server Plug-ins that are included by Enterprise Server. They run within the very same PHP process as Enterprise Server. Therefore the integration does not need to run over an HTTP connection, which makes the integration easier to develop and faster in execution.

External PHP integrations are custom stand-alone PHP client applications. Their PHP modules run at a server machine and they communicate to Enterprise Server over an HTTP connection. Therefore they are called *client* applications, seen from Enterprise Server point of view. The SOAP is the most obvious protocol to chose for such integrations. When it is needed to upload or download files to Enterprise Server, DIME can still be used but is no longer recommended. It is better to integrate the Transfer Server instead, which runs over another HTTP connection.

Internal PHP integrations

PHP data classes as well as service request- and response classes are generated from the WSDLs and shipped with Enterprise Server located in this folder:

```
.../Enterprise/server/services/<interface>
```

For example, this is how you can create a new user:

```
require_once( '.../Enterprise/config/config.php' ); // TODO: Replace ... with web root folder!
require_once BASEDIR.'/server/secure.php';
require_once BASEDIR.'/server/interfaces/services/adm/DataClasses.php'; // AdmUser
require_once BASEDIR.'/server/services/adm/AdmCreateUsersService.class.php';
try {
	// Build new user in memory.
	$userObj = new AdmUser();
	$userObj->Name = 'woodwing';
	$userObj->FullName = 'WoodWing Software'
	$userObj->Deactivated = false;
	$userObj->Password = 'ww';

	// Build service request to create a new user.
	$request = new AdmCreateUsersRequest();
	$request->Ticket = '...'; // TODO: Fill-in your ticket here! (take it from LogOn)
	$request->RequestModes = array();
```

```
        $request->Users = array( $userObj );

        // Request Enterprise to create user in the database.
        $service = new AdmCreateUsersService();
        $response = $service->execute( $request );
        // $response is an AdmCreateUsersResponse object

        // Just display the user's DB id of the created user.
        echo $response->Users[0]->Id;
} catch( BizException $e ) {
        echo $e->getMessage();
}
```

Basically, the whole pattern of calling services is always the same, no matter what service is called. First build a request object in memory and then execute the service by passing in the request. This returns a response object which can be used to validate or analyze the results. This is all shown in the example.

The DataClasses.php file is included to define all PHP classes derived (generated) from the WSDL file, in this case SmartConnectionAdmin.wsdl. In the WSDL data classes are defined by so called complexType elements. In this case the complexType "User" is of our interest. For each complexType a PHP data class is generated, which is listed in the DataClasses.php file. The PHP classes have a prefix to indicate the interface they originate from. (By exception, no prefixes are used for the workflow interface.) For this case Adm prefix is used, and so for the "User" complexType a PHP class named "AdmUser" is generated. See Definitions and entry points to look up interfaces and prefixes. The $user variable represents the AdmUser object instance for which properties can be set.

It is a good habit not to list parameter values in the constructor, but to explicitly assign values to the created object's properties individually, as done in the example for Name, FullName, etc. The reason is that parameters might appear or disappear in the future, which might break your code when using constructors. When using properties, your code won't break.

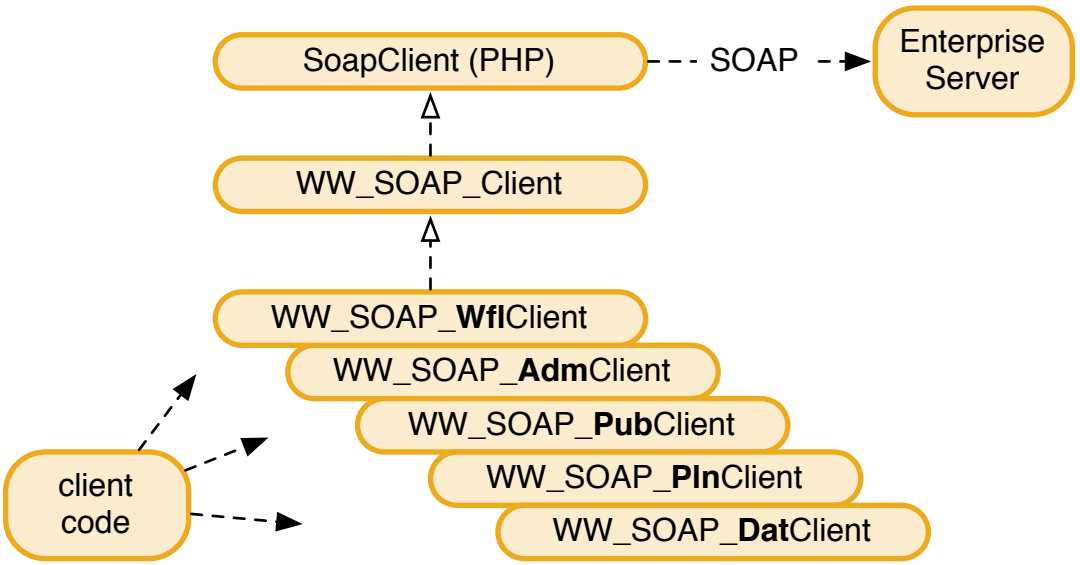
To create users, the WSDL specifies CreateUsersRequest and CreateUsersResponse elements. For those elements PHP classes are generated as well, and again prefixed, for this interface with Adm. By including the AdmCreateUsersService.class.php three classes are included at once:
AdmCreateUsersService, AdmCreateUsersRequest and AdmCreateUsersResponse. Those objects are used to execute the service.

External PHP integrations

For automatic data class mapping and DIME attachment support, Enterprise has extended PHP's SoapClient class with its WW_SOAP_Client class. For each interface flavor, another class is provided, such as WW_SOAP_AdmClient which communicates with the Administration interface. (See Definitions and entry points for the list of available interfaces.) These classes parameterize the WW_SOAP_Client class with the corresponding entry point and WSDL information, and map WSDL elements to PHP classes for programming convenience. All these classes can be found in this folder:

```
.../Enterprise/server/protocols/soap
```

The figure below shows how the classes inherit from each other, as displayed in the middle. On the left side, your PHP client code is calling any of the helper classes. On the right, the Enterprise Server is invoked through SOAP to execute a service.



Some test modules in the .../Enterprise/server/wwtest folder use the SOAP helper classes from where you might want to take some useful code fragments:

- workflow interface: speedtest.php
- admin interface: adminservicetest.php
- planning interface: clientplan.php
- data source interface: datasourceservicetest.php

Note that even for an external integration, you can still include classes from Enterprise Server without using Enterprise Server as a server. A good reason for that is to profit from the helper- and data classes to speed up your client developments.

An example of a SOAP client application calling the admin interface (over HTTP), which is requested to create a new user and retrieve its database id:

```
require_once( dirname(__FILE__).'/../../config/config.php' );
try {
	$ticket = '...'; // TODO: Fill-in your ticket here! (take it from LogOn)

	// Build new user in memory.
	require_once BASEDIR.'/server/interfaces/services/adm/DataClasses.php'; // AdmUser
	$userObj = new AdmUser();
	$userObj->Name = 'woodwing';
	$userObj->FullName = 'WoodWing Software';
	$userObj->Deactivated = false;
	$userObj->Password = 'ww';

	// Build service request to create a new user.
	require_once BASEDIR.'/server/interfaces/services/adm/AdmCreateUsersRequest.class.php';
	$request = new AdmCreateUsersRequest();
	$request->Ticket = $ticket;
	$request->RequestModes = array();
	$request->Users = array( $userObj );

	// Request Enterprise to create user in the database.
	require_once BASEDIR.'/server/protocols/soap/AdmClient.php';
	$soapClient = new WW_SOAP_AdmClient();
	$response = $soapClient->CreateUsers( $request );

	// Just display the user's DB id of the created user.
	echo 'Created user with DB id: '.$response->Users[0]->Id.'<br/>';
} catch( SoapFault $e ) {
	echo $e->getMessage() . '<br/>';
} catch( BizException $e ) {
```

```
      echo $e->getMessage() . '<br/>';
}
```

External PHP integrations - stand alone

The disadvantage of the helper- and data classes is, that you need to copy them from the Enterprise Server machine to your own server and keep them in sync. Alternatively you can use the PHP SoapClient class directly, but keep in mind that:

- There is no support for file transfers.
- There is no data class mapping done on the client side. This means you need to work with stdClass objects only.

The same example as above, but now without any helper classes:

```
try {
      $ticket = '...'; // TODO: Fill-in your ticket here! (take it from LogOn)

      // Build new user in memory.
      $userObj = new stdClass();
      $userObj->Name = 'woodwing';
      $userObj->FullName = 'WoodWing Software';
      $userObj->Password = 'ww';

      // Build service request to create a new user.
      $request = new stdClass();
      $request->Ticket = $ticket;
      $request->RequestModes = array();
      $request->Users = array( $userObj );

      // Request Enterprise to create user in the database.
      $soapClient = new SoapClient(
            'http://localhost/Enterprise/adminindex.php?wsdl',
            array( 'location' => 'http://localhost/Enterprise/adminindex.php',
                  'uri' => 'urn:SmartConnectionAdmin',
                  'soap_version' => SOAP_1_1, 'trace' => 1 ));
      $response = $soapClient->CreateUsers( $request );

      // Just display the user's DB id of the created user.
      echo 'Created user with DB id: '.$response->Users[0]->Id.'<br/>';
} catch( SoapFault $e ) {
      echo $e->getMessage() . '<br/>';
}
```

External PHP integrations - with Transfer Server [since 8.0]

For integrations with the Transfer Server a helper class in the utils folder is available named WW_Utils_TransferClient. This class can upload or download files to/from the Transfer Server over HTTP.

The example below illustrates how this helper class can be used. First an object is retrieved through the workflow interface with the GetObjects service call. The native file rendition is thereby requested. The 'transfer' => 'HTTP' option tells the server to use the Transfer Server (instead of DIME). As a result, the GetObjects response contains a URL to the Transfer Server to download the file. The WW_Utils_TransferClient utils class is used to download that native file.

```
require_once( dirname(__FILE__).'/../../config/config.php' );
require_once BASEDIR.'/server/secure.php';
try {
      $ticket   = '...'; // TODO: Fill-in your ticket here! (take it from LogOn)
      $objectId = '...'; // TODO: Fill-in the object id your want to download native file for!
```

```
	// Request for native file for an object.
	require_once BASEDIR.'/server/services/wfl/WflGetObjectsService.class.php';
	$request = new WflGetObjectsRequest();
	$request->Ticket = $ticket;
	$request->IDs = array( $objectId );
	$request->Lock = false;
	$request->Rendition = 'native';

	// Request Enterprise to create user in the database.
	require_once BASEDIR.'/server/protocols/soap/WflClient.php';
	$options = array( 'transfer' => 'HTTP', 'protocol' => 'SOAP' );
	$soapClient = new WW_SOAP_WflClient( $options );
	$response = $soapClient->GetObjects( $request );
	$attachment = $response->Objects[0]->Files[0];

	// Download the native file from Transfer Server.
	require_once BASEDIR.'/server/utils/TransferClient.class.php';
	$transferClient = new WW_Utils_TransferClient( $ticket );
	if( $transferClient->downloadFile( $attachment ) ) {
		echo 'Downloaded file successfully.<br/>';
	} else {
		echo 'ERROR: Failed downloading file.<br/>';
	}
} catch( SoapFault $e ) {
	echo $e->getMessage() . '<br/>';
} catch( BizException $e ) {
	echo $e->getMessage() . '<br/>';
}
```

The example below shows roughly how to upload files. After doing so, the attachment can be passed to a CreateObjects or SaveObjects request.

```
try {
	$ticket = '...'; // TODO: Fill-in your ticket here! (take it from LogOn)

	// Build a native image attachment in memory.
	$attachment = new Attachment();
	$attachment->FilePath = '/full/path/to/my/native/image/file.jpg'; // TODO: adjust
	$attachment->Rendition = 'native';
	$attachment->Type = 'image/jpeg';

	// Upload the image file to the Transfer Server folder.
	require_once BASEDIR.'/server/utils/TransferClient.class.php';
	$transferClient = new WW_Utils_TransferClient( $ticket );
	$transferClient->uploadFile( $attachment );

	// TODO: Call CreateObjects or SaveObjects

} catch( BizException $e ) {
	echo $e->getMessage() . '<br/>';
}
```

Shared Concepts

This chapter describes concepts used in all Web Service interfaces.

WSDL

Each interface is described with a separate WSDL. The WSDL v1.1 specification distinguishes between two message styles: document and RPC. Furthermore there are different serialization formats, with SOAP encoding and literal being the two popular serialization formats today. The WSDLs shipped with Enterprise Server all use the document message style with literal encoding, also known as document/literal.

Attachments

The Web Service interfaces use XML messages through an HTTP connection. Any files transferred are sent through SOAP attachments with DIME being used to encapsulate this into a single data stream. The SOAP message itself and the attachments are all encapsulated by DIME. To handle a DIME request or response, the DIME must be parsed to be able to access the SOAP message itself. There are only a few Web Services that transfer files, which is limited to the workflow- and planning interfaces. Those can be recognized by the *dime:message* element shown for the few operations in the WSDL, for example:

```
<operation name="CreateObjects">
      <soap:operation soapAction="urn:SmartConnection#CreateObjects"/>
      <input>
            <dime:message
                  layout="http://schemas.xmlsoap.org/ws/2002/04/dime/closed-layout"
                  wsdl:required="false"/>
            <soap:body use="literal"/>
      </input>
      <output>
            <soap:body use="literal"/>
      </output>
</operation>
```

Note that the *input* element stands for the SOAP request fired by the client and the *output* element for the SOAP response returned by the server. The fragment above tells us that the CreateObjects operation (Web Service to upload documents) accepts DIME attachments, but that none are returned, which makes sense.

For more information on SOAP attachments, see http://www.w3.org/TR/soap12-af/.

For more information on DIME encapsulation, see http://www.watersprings.org/pub/id/draft-nielsen-dime-soap-01.txt.

Since Enterprise 8, it is recommended to use the Transfer Server instead of DIME. For both technologies, the DIME definitions in the WSDLs specify what service requests or responses can deal with file attachments.

Errors

A SOAP *Fault* is returned to clients when an error has occurred server-side during any Web Service execution. For example:

```
<SOAP-ENV:Envelope>
      <SOAP-ENV:Body>
            <SOAP-ENV:Fault>
                  <faultcode>SOAP-ENV:Client</faultcode>
                  <faultstring>Access denied (S1002)</faultstring>
                  <faultactor/>
                  <detail>1661(C)</detail>
            </SOAP-ENV:Fault>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

When the *faultcode* is set to Client it means that the caller has somehow passed wrong parameters or tries to do something that is against the rules. When set to Server it means that the server is not 'willing' to process the operation or has an internal error. It is a rough indication who might have caused the error: Client or Server. Typically used for debugging.

The *faultstring* is a localized string that can be directly shown to the user. The server error code is attached to this message in "(Sxxxx)" format. Clients that need to know what error occurred, should parse this error code, for example by taking the code 1002 in the given example. Clients sometimes know better what happened than the server, especially when firing many requests to establish one logical heavy operation. They are free to interpret the error code and display other messages (other than the ones returned to the client) to help the user understand what happened. The 1000-1999 range is reserved for operational errors; the 2000-2999 range is reserved for license errors.

Depending on the fault, a *detail* message can also be provided by the server to pass on extra information. The detail message is not localized and mostly cryptical, so it should never be shown to end users. In the given example, the server informs that the requested operation was performed for an object (ID=1661), for which the user has has no Change Status (C) access rights. Clients could try to parse this, but its format is free and subject to change. So, it is not recommended and should be used only in exceptional cases. For example, the server can add tokens as well to the *detail* message; for example in case an invalid ticket is passed (perhaps because it is expired) a fault is returned with *detail* message set to "SCEntError_InvalidTicket". This should be checked by clients to determine if there is a need to raise the re-logon dialog.

Tickets

Most Web Services require a ticket as input parameter. This ticket is obtained from the LogOn operation. The LogOn operation validates the user name and password and returns a ticket when the user account is valid. The ticket is used by clients for subsequent requests. Tickets are interchangeable between all interfaces.

By default, a logon/ticket session expires within 24 hours (or 1 hour for web clients). Whenever a request with a valid ticket arrives, the server resets its expiration timer for that session. As long as requests are fired before the expiration ends, clients are able to continue working, thereby keeping one license seat occupied. When a ticket has expired, an error is returned and the client needs to re-logon obtaining a new/different ticket. At this point, the clients risks that the last seat was taken in the meantime.

When the very same user does logon with the very same client application, but from a different location (IP address), the server assumes the user has moved. And so, the ticket from the first application is made invalid. This is to avoid accidentally keeping two license seats occupied for the whole day.

Services

Unlike the stored information in the database, Enterprise Server itself is stateless. Every requested service runs on its own. When one service depends on another service, typically resulting data from one is passed on to the other. For example, an object id returned through a search query can be used to open the object file for editing.

Each service intends to represent one user operation. In other terms, when a user performs one action, there should be just one service executed only. This is needed for two reasons:

- **Performance** - Think of a slow connection with one second network overhead. Performing five requests will require five seconds extra wait time, which is unacceptable.
- **Architecture** - One service representing one logical action implies the need of passing all information to execute the service. This makes it possible to interpret the intention of the operation, which is why it is a good thing to develop custom solutions using Server Plug-ins hooking into Web Services.

However, in some cases multiple services are called by client applications, trigged by one logical action. For example, when a user logs in, Content Station typically requests to execute the inbox query immediately after, to show objects assigned to that user. Such scenarios are valid since running a query is a totally different logical operation than the login.

Client applications can wait for each service to complete and then fire a next request. This synchronous communication method works well but is far from optimal. Calling services asynchronously is allowed by Enterprise Server and can significantly speed up end user wait times. Content Station does this (by running multiple threads). Obviously, this can be done when there are no specific dependencies between services (as mentioned above).

Arrays

One of the most often used structures in SOAP is the array. It allows transferring many of the same entities (objects, relations, pages, etc.) in a list through requests and responses. The way arrays must be formatted is specified in the WSDL. For Enterprise, those definitions are always prefixed with *ArrayOf* followed by the entity name. For example:

```
<complexType name="ArrayOfPlacement">
      <sequence>
            <element name="Placement" minOccurs="0"
                  maxOccurs="unbounded" type="tns:Placement"/>
      </sequence>
</complexType>
```

It tells SOAP messages using this type that zero to many *Placement* elements are allowed. The elements must be structured as follows:

```
<Placements>
      <Placement>
            ...
      </Placement>
      <Placement>
            ...
      </Placement>
</Placements>
```

Since Enterprise 7.0 this notation has been changed to meet new standards with better support of SOAP tools. The definition has been changed in such a way that the old notation (usage) is still supported though:

```
<complexType name="ArrayOfPlacement">
      <complexContent>
            <restriction base="soap-enc:Array">
                  <attribute ref="soap-enc:arrayType"
                        wsdl:arrayType="tns:Placement[]"/>
            </restriction>
      </complexContent>
</complexType>
```

This tells the same as before, but now a new notation can be used too:

```
<Placements SOAP-ENC:arrayType="ns1:Placement[]" xsi:type="SOAP-ENC:Array">
      <item>
            ...
      </item>
      <item>
            ...
      </item>
</Placements>
```

Basically, the parent *Placements* attributes says that its child *item* elements are actually *Placement* elements. For Enterprise 7, Content Station uses this technique, while InDesign / InCopy still use the ‘old’ method.

User access rights

One of the keystones of the Enterprise workflow system is support for user access rights. Whenever an end user tries to do something for which no access is granted, and “Access denied (S1002)” error is raised. This should be determined by the server only and not by client applications. In such a case, the server returns a SOAP fault (see Errors). Such an error is shown in a dialog that is raised by client applications.

Enabling vs disabling access

Access rights are defined through Access Profiles. (See the Enterprise Admin Guide for information about how to configure these.) Access feature options can be selected, which means they are enabled. However, when not selected, it does not(!) disable the feature; it only means that the feature is not enabled in this particular access profile, so you are not blocking access or something like that. A user has no access for a certain feature if there is no profile found with the corresponding option selected. For this reason, it is recommended to split options through many profiles and enable just a few. (This instead of having all enabled and disabling a few.) Else you might find yourself wondering why an end user has gained access unexpectedly.

Interpreting access definitions

When the end user performs a login action to the system, the server returns the configured profile definitions (through the LogOnResponse -> FeatureProfiles element) made by the admin user that are relevant for the end user. Only access options that have no default value are returned. You can determine the defaults by creating a new Access Profile through the admin pages and see which ones are selected. (At least up to v7.0 all options are selected by default, with the exception of the “Force Track Changes” option.)

Let’s say that the system admin creates one Access Profile, clears the “Create Dossiers” option and names it “no Dossier creation”. The definition in the login response will look as follows:

```
<LogOnResponse>
      ...
      <FeatureProfiles>
            <FeatureProfile>
                  <Name>no Dossier creation</Name>
                  <Features>
                        <AppFeature>
                              <Name>CreateDossier</Name>
                              <Value>No</Value>
                        </AppFeature>
                  </Features>
            </FeatureProfile>
            ...
      </FeatureProfiles>
      ...
</LogOnResponse>
```

Note that the “Create Dossiers” is a localized term. To uniquely identify this access feature, internal keys are used (in this case *CreateDossier)*. Those keys can be used by client applications to look up and interpret. For example, a client application implementing a Create Dossier operation could check for this specific key.

With the definition available, the Brand admin can use it to give the end user access to the Brand in the Authorization Maintenance admin page. Let’s say that the Brand admin has given the end user access to the WW News Brand through the “no Dossier creation” profile (for all statuses and all Categories).

When the end user performs a login action, this configuration is reflected in the login response too (through Publications -> PublicationInfo -> FeatureAccessList element). See the figure below. The profile name (the text marked in red in the figure below) refers to the profile definition (the text marked in red in the figure above). This way clients can look up definitions. Because the Brand admin user has configured for all statuses and all Categories, respectively *State* and *Section* elements are not provided (*xsi:nil* attribute set to “true”). Nil typically means no specific item configured, which implies all options. (Note that Category was formerly named Section.) When configured, an id is filled in for those elements. When access rights are configured for an Issue with the “Overrule Brand” option enabled, the *Issue* element is used to pass its id.

```
<LogOnResponse>
      ...
      <Publications>
            <PublicationInfo>
                  <Id>1</Id>
                  <Name>WW News</Name>
                  ...
                  <FeatureAccessList>
                        <FeatureAccess>
                              <Profile>no Dossier creation</Profile>
                              <Issue xsi:nil="true"/>
                              <Section xsi:nil="true"/>
                              <State xsi:nil="true"/>
                        </FeatureAccess>
                  </FeatureAccessList>
            </Publications>
      </PublicationInfo>
      ...
</LogOnResponse>
```

Disabling GUI items

For user convenience, the client application GUI can be enhanced by hiding/disabling operations that are expected to always fail for a specific context. For example, when the user is not allowed to create Dossiers at all, no matter what, the client application could disable the “Create Dossier” operation in its GUI (such as context menus). Nevertheless, this must be done with great care. When disabling is done too rigidly it could accidentally withhold users from doing operations that they are allowed to. This can be implemented by interpreting the login response, specifically the two areas mentioned in the examples above. When the user is working in the context of the WW News Brand (for example selected in the GUI) the client needs to check all the profiles for that Brand (by looking up the profile definitions as explained above). If none of them gain access to the *CreateDossier* feature, it is safe to disable the “Create Dossier” operation in the GUI. If the context of the Brand is undetermined (for example global view on the system) all profiles (of all Brands) need to be checked. When the currently selected Issue (if any) has the Overrule Brand option set, only the configurations made for the Issue should be checked (and the configurations for its Brand should be ignored!).

Overruling access rights

When normal configurations made through admin pages do not fulfill the customer's needs, Server Plug-ins could be the answer. Its connectors allow you to hook into any workflow service. See the Server Plug-ins documentation for more info. When any operation is not allowed, the connector could throw a BizException as follows:

```
throw new BizException( 'ERR_AUTHORIZATION', 'Server', '' );
```

For example, it could check if the end user is member of a specific user group, it could check the status of the object involved, it could check the deadline and system time, etc, and then decide to block access.

Enterprise Data Entities

Web Services are all about operations through which data structures are passed and returned. Operations are performed by the server on demand of client applications. Before diving into operations, you first need to understand what the data structures represent. Those are defined in documents (WSDLs) and named after Enterprise's data entities, such as a 'Page' and an 'Edition'. This section explains the most important entities used within the Enterprise world.

Workflow Entities

Objects

Enterprise manages workflow objects. The way objects are treated strongly depends on the object type. For example, an article is an Object for which the type is set to Article. By knowing the type, specific business rules are triggered, such as whether an object is placeable, if it can be placed multiple times, on which other object type it can be placed, etc. Once an object is created, its type can never change.

The complete list of supported types can be found in the WSDL in the ObjectType definition. One of them is the Other type, which is especially useful for system integrators introducing foreign object types that do not look like any of the standard supported types and need no special treatment by Enterprise itself. For example, custom Server Plug-ins (or SOAP clients) could check for this type and do something fancy with the object, such as unzipping a ZIP file and dynamically create objects from the files found inside.

Formats

Note that different formats (technical types) are supported per object type. For example for Article objects, the following formats are commonly used: InCopy, plain text, RTF, Word and Excel. For formats, MIME content types are used (see http://www.w3.org/Protocols/rfc1341/4_Content-Type.html). Common format types used in Enterprise are:

- application/incopy
- application/indesign
- image/jpeg
- image/tiff
- application/pdf
- application/postscript
- application/msword
- application/rtf
- text/richtext
- text/plain

Object IDs

Each object has an ID to uniquely identify the object within the Enterprise system. The object ID is typically numeric, but it can also be any string (which is sometimes done by (Content Source) integrations such as Fotoware). Strings are allowed, as long they can be used as a folder name on Mac OSX and Windows, which is required by Enterprise clients.

Status

An object is always in exactly one workflow status at the same time. This tells which workflow definition the object is following. The workflow is set up through the Brand Maintenance admin pages (see the Admin Guide for details.) A status depends on the object type; when there is no status configured for an article object, articles cannot be created. A status has an ID which is unique within the whole system. Personal statuses have the ID set to -1 (regardless of the user or object type). Those special statuses are recognized by the system and treated differently; they are not stored in the database (unlike normal statuses) and are created on-the-fly which makes users think they truly

exist. A status has a localized name and a color configured, which both have no meaning for the system at all. A sequence number is used to order statuses shown in lists.

Object properties

Along with each object, metadata is stored. It consists of a pre-defined set of properties to which custom properties can be added. Metadata is structured differently depending on the context of usage, as explained in the following paragraphs.

Metadata services

When end users are retrieving, creating or saving objects (or setting properties), metadata properties are put in a hierarchical tree. This tree structure can also hold entities, such as a status which contains the id, name, etc. (see fragment below). The entity data is assumed to be read-only when passed along with objects. Typically it is the server returning such data for client convenience. All of this gets ignored when it is passed back by the clients to the server, except for the id. Ids are referring to entity instances stored in the database, so when the ids are changed during an object save operation (SaveObjects), the server will update references made from the object to the entity instances stored in the database.

```
<Object>
      <MetaData>
            ...
            <WorkflowMetaData>
                  ...
                  <State>
                        <Id>257</Id>
                        <Name>For Review</Name>
                        <Type>Article</Type>
                        <Produce xsi:nil="true"/>
                        <Color>#666633</Color>
                        <DefaultRouteTo xsi:nil="true"/>
                  </State>
                  ...
            </WorkflowMetaData>
            ...
      </MetaData>
</Object>
```

The format of properties and entities (such as the *State*) can be looked up in the WSDL (see the fragment below). For example, the status name is a *string*.

```
      <complexType name="State">
            <all>
                  <element name="Id" type="xsd:string"/>
                  <element name="Name" type="xsd:string"/>
                  <element name="Type" type="tns:ObjectType"/>
                  <element name="Produce" nillable="true" type="xsd:boolean">
                  <element name="Color" nillable="true" type="tns:Color"/>
                  <element name="DefaultRouteTo" nillable="true" type="xsd:string"/>
            </all>
      </complexType>
```

Custom properties are sent in list format (through MetaData -> ExtraMetaData) as shown in the fragment below. Their names are prefixed with “C_”. Regardless of the configured data type, the current value(s) is always a list of strings.

```
<Object>
	<MetaData>
		...
		<ExtraMetaData>
			<ExtraMetaData>
				<Property>C_IS_MY_HAIR_OK</Property>
				<Values>
					<String>No</String>
				</Values>
			</ExtraMetaData>
			...
		</ExtraMetaData>
	</MetaData>
</Object>
```

Query services

When users perform a search query, the properties are returned in rows and columns. Such properties (sent through QueryObjects and NamedQuery services) are formatted using the very same structure elements (such as *Property* and *Row*). For entity elements the element name itself represents the name. Its elements are concatenated. For example, the status name is *State* and the status id is *StateId*, as shown in the following fragment:

```
<QueryObjectsResponse>
	<Columns>
		...
		<Property>
			<Name>State</Name>
			<DisplayName>Status</DisplayName>
			<Type>list</Type>
		</Property>
		<Property>
			<Name>StateId</Name>
			<DisplayName>Status ID</DisplayName>
			<Type>string</Type>
		</Property>
		...
	</Columns>
	<Rows>
		<Row>
			...
			<String>For Review</String>
			<String>257</String>
			...
		</Row>
		...
	</Rows>
</QueryObjectsResponse>
```

It shows that (at least) two columns are returned; the status name and the status id. There is also (at least) one row returned, which contains the actual data. Which columns are included depends on the configuration made in the “Query Result Columns” of the Dialog Setup admin page (for QueryObjects service) or the definition of the Named Query (for NamedQuery service). See the Admin Guide for configuration details.

For a full list of property names, look in the server/bizclasses/BizProperty.class.php file.

Dialog services

Whenever a client application raises a workflow dialog, it requests the dialog definition (through GetDialog service). The actual object properties are included as well (in GetDialog -> MetaData) to be displayed in the dialog. The properties are flattened to the very same structure elements (MetaDataValue). This is done for client application convenience. For example, when the Status property is configured to show in the dialog,

and the object is currently in the “For Review” status (id=257), the returned structure looks like the following fragment:

```
<GetDialogResponse>
      <Dialog>
            ...
            <MetaData>
                  ...
                  <MetaDataValue>
                        <Property>State</Property>
                        <Values>
                              <String>257</String>
                        </Values>
                  </MetaDataValue>
                  ...
            </MetaData>
            ...
      </Dialog>
</GetDialogResponse>
```

You might wonder why the name of the status id property is not named *StateId*, but just *State*. This is because from an admin user point of view (configuring dialogs) the status name should be shown. But, in the dialog, the end user is selecting a status from the list, which results in an id. To make references correctly within this service response, the *State* property is used to represent the status id.

See Workflow services for a detailed service explanation.

Object metadata

As mentioned in the previous paragraph, object properties can be structured in a hierarchical tree. The advantage is that the structure is organized, humanly readable and the data types are pre-defined for all built-in properties. This is done by using the *MetaData* definition in the workflow WSDL. It consists of the following main structure parts with different purposes:

- *BasicMetaData* - Identifies the object and tells how it is bound to the Brand and workflow. For new objects, there is no id, and so name, type and Brand (publication) are needed.
- *TargetMetaData* - Obsoleted since Enterprise 6. Use Object -> Targets instead. See Structural change for multiple channels for more information.
- *RightsMetaData* - Copyright information.
- *SourceMetaData* - Credit information.
- *ContentMetaData* - Properties closely related to the native content file. Which properties are filled differs per object type or file format. These properties are versioned.
- *WorkflowMetaData* - Tells how the object is currently doing in the workflow. Who created it, who is working on it, what the progress status is, what deadlines should be met, etc.
- *ExtraMetaData* - Used to carry custom properties along with the object. Since these properties are configurable, there is no type checking done through the WSDL.

Object renditions

Per individual object, Enterprise can store multiple files, called renditions. Typically objects have a native rendition and possibly preview, thumbnail and output renditions. The WSDL identifies the following renditions:

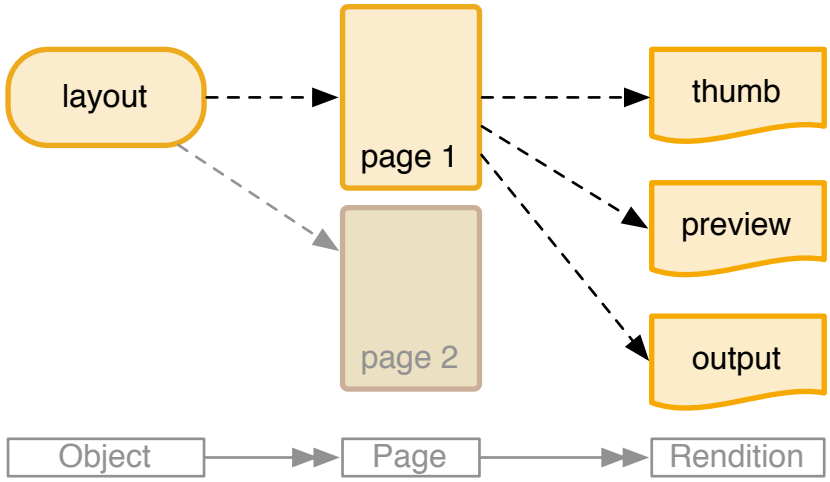
- none - no rendition, used for example to get an object's properties without any of its files.
- thumb - optimized for thumbnail overviews, typically JPEG of approximately 100 pixels wide.
- preview - optimized for preview purposes, typically JPEG of approximately 600 pixels wide.
- placement - abstract rendition, requested by InDesign when an image is placed. The Enterprise Server returns the rendition with the highest quality, preferably native.
- native - primary rendition used for editing, for example a PSD file for an image.
- output - export rendition, typically PDF or EPS for the pages of a layout object.
- trailer - preview rendition for audio/video.

Layout pages

Enterprise stores InDesign files (layouts). With a layout, layout template, Layout Module or Layout Module template, pages are stored. These layout flavors are objects, but note that the pages themselves are not treated as objects. Whenever the term *layout* is used in the paragraph, it refers to any of the layout flavors.

Page renditions

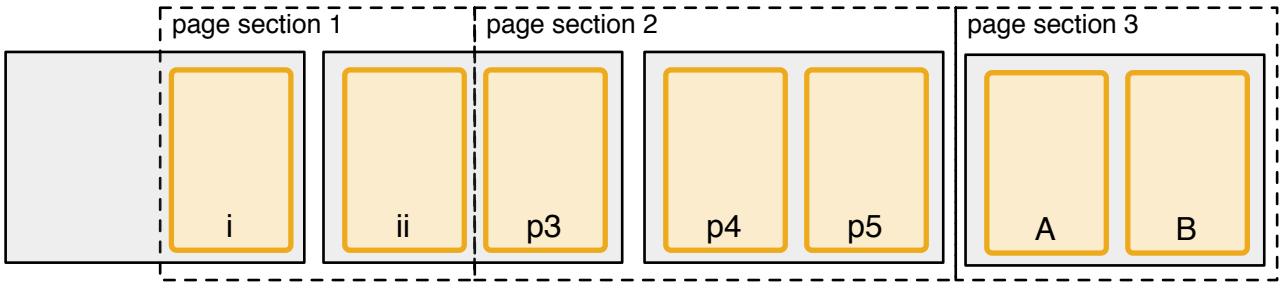
Pages have renditions, just like objects have. And so, a thumb, preview and output rendition file can be stored per page. The output rendition is typically PDF or EPS.



Note: Page rendition files are not versioned; only the last version of each rendition is tracked. When restoring an old version of a layout without InDesign intervention, the page renditions remain untouched. As a result, the version that is 'too new' is still shown in the Publication Overview.

Page numbers

Per page, the following is tracked: order, number and sequence. The *order* stands for the logical internal numeric position, as used by InDesign internally. Per page section, you can restart this numbering. Although you choose a numbering system (alphanumeric, roman, arabic, etc), the order is always numeric. The page *number* is the human readable representation. This reflects the page numbering system and optionally can be prefixed with the page section prefix. The number is typically used to print on the page. The page *sequence* represents the page position within the layout. This is used to uniquely identify pages since the page order does not tell, as shown in the following figure.

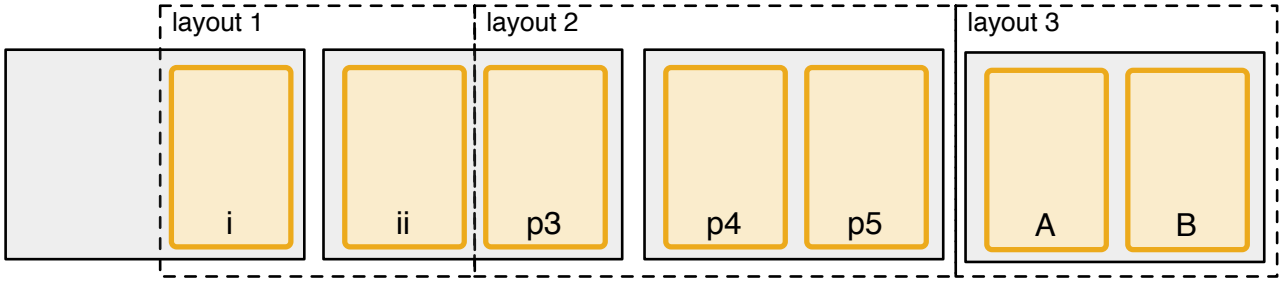


number	i	ii	p3	p4	p5	A	B
order	1	2	3	4	5	1	2
sequence	1	2	3	4	5	6	7

It shows a layout with 7 pages divided into 3 page sections. Each section uses a different numbering system, respectively roman, arabic and alphanumeric. The second section has continued numbering and defines prefixes. The third section has restarted numbering.

Splitting up pages

The example above could work for small documents such as brochures. For large documents, such as magazines and books, splitting up pages into multiple layouts is recommended. Let's split-up the pages in three layout objects, exactly the way it is split-up by page sections. You will end up with 3 layouts, each having one page section, as shown in the following figure.



number	i	ii	p3	p4	p5	A	B
order	1	2	3	4	5	1	2
sequence	1	2	1	2	3	1	2

Notice that the page *sequence* is now 'restarted' per layout object.

With pages split-up into layouts, they are no longer strictly bound to each other. This enables the server to put them in the Publication Overview with more intelligence. When creating a new layout (for example containing pages p6 and p7) this automatically gets inserted into the overview, without the need to open and edit "layout 2".

Object relations

Objects can be related to each other in several ways. For example, an article can be placed onto a layout. In this case we speak of a 'Placed' relation type of which the layout is the parent and the article is the child. Another example is an image that is put into a Dossier. In this case, we speak of a 'Contained' relation type with Dossier as parent and image as child. Most relations are parent-child relations which implies that they can be represented hierarchically in the search results. The complete list of supported relation types can be found in the *RelationType* element of the workflow WSDL file.

Since Enterprise 6 the 'Related' type is introduced which is quite exceptional. It is used for Dossiers that are related to each other. Unlike the hierarchical relations, this is a bother-sister relation which are therefore not shown in the search results.

Client applications typically inform the server explicitly which relations to make (or remove). The following workflow services can be used to manage object relations:

- CreateObjectRelations
- UpdateObjectRelations
- DeletetObjectRelations
- GetObjectRelations

Since Enterprise 7 users can create an object (let's say a new article), whereby he/she is offered to select the "Create Dossier" item of the Dossier property in the workflow dialog. Therefore, the object creation service (CreateObjects) supports implicit Dossier creation. This is simply done by sending a fictitious relation to the

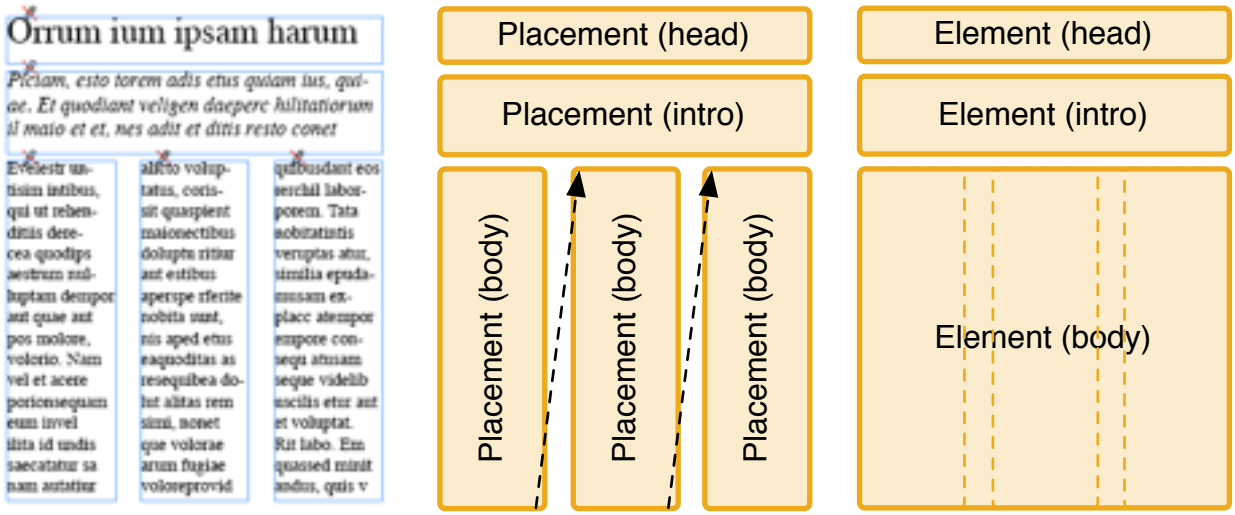
Dossier through this service. This triggers the server to create a new Dossier before creating the article. With both objects available, the server then creates a 'Contained' relation between them. Since the parent Dossier object is not created at the time when the service is called, there is no id and so -1 should be specified. The fragment below shows how this is done.

```
<CreateObjects>
      <Objects>
            <Object>
                  ...
                  <Relations>
                        <Relation>
                              <Parent>-1</Parent>
                              <Child xsi:nil="true"/>
                              <Type>Contained</Type>
                              ...
```

Placements and Elements

With InDesign, articles and images are placed on layouts by using frames. Frames can either be textual or graphical. From a multiple frame selection, an article can be created. Images are created from a single frame. Each frame has geometrical information and is stored into Enterprise as a *Placement* (whenever an object is made from it).

Consider a placed article shown on the left in the figure below. It consist of five frames; one *head* frame, one *intro* frame and three *body* frames. The three *body* frames are linked for continuous reading. The figure in the middle shows how InDesign frames become Enterprise placements. Multiple frames that are linked are seen as just one story, just like single frames. A story is called *Element* in Enterprise (or Component for the end user). In the example, the three *body* frames become one *Element*. So the article consists of three elements/components; *head*, *intro* and *body*, as shown on the right.



Assume that the article is placed on the layout. The create and save requests (CreateObjects and SaveObjects) will carry out the elements and placements to the server.

For **layout** create/save operations in Objects -> Object -> Placements element, the structure is shown in the fragment below. The *ElementID* for the body placements is the same. This way, the placements are bundled per element. The *FrameOrder* tells the reading sequence of the placements.

```
...
<Placements>
	<Placement>
		<Page>1</Page>
		<Element>head</Element>
		<ElementID>CAA9DCF6-4F67-43AE-B9E0-1E01DBA40CC4</ElementID>
		<FrameOrder>0</FrameOrder>
		...
	<Placement>
		<Page>1</Page>
		<Element>intro</Element>
		<ElementID>D65C0A5F-8E0B-4456-985A-ECC03BDAE6C4</ElementID>
		<FrameOrder>0</FrameOrder>
		...
	<Placement>
		<Page>1</Page>
		<Element>body</Element>
		<ElementID>23383EBF-4740-4481-AF76-C7410744A094</ElementID>
		<FrameOrder>0</FrameOrder>
		...
	<Placement>
		<Page>1</Page>
		<Element>body</Element>
		<ElementID>23383EBF-4740-4481-AF76-C7410744A094</ElementID>
		<FrameOrder>1</FrameOrder>
		...
	<Placement>
		<Page>1</Page>
		<Element>body</Element>
		<ElementID>23383EBF-4740-4481-AF76-C7410744A094</ElementID>
		<FrameOrder>2</FrameOrder>
		...
</Placements>
...
```

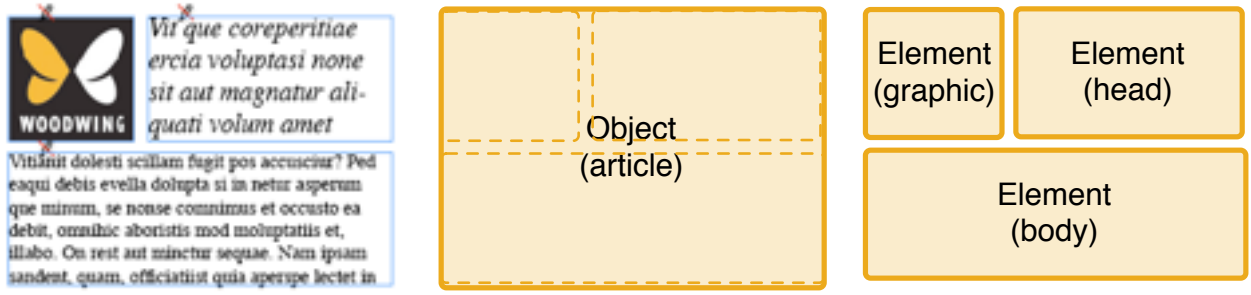
For **article** create/save operations in Objects -> Object -> Placements element, the structure is shown in the fragment below. It lists the three elements.

```
...
<Elements>
	<Element>
		<ID>CAA9DCF6-4F67-43AE-B9E0-1E01DBA40CC4</ID>
		<Name>head</Name>
		...
	<Element>
		<ID>D65C0A5F-8E0B-4456-985A-ECC03BDAE6C4</ID>
		<Name>intro</Name>
	...
	<Element>
		<ID>23383EBF-4740-4481-AF76-C7410744A094</ID>
		<Name>body</Name>
		...
</Elements>
...
```

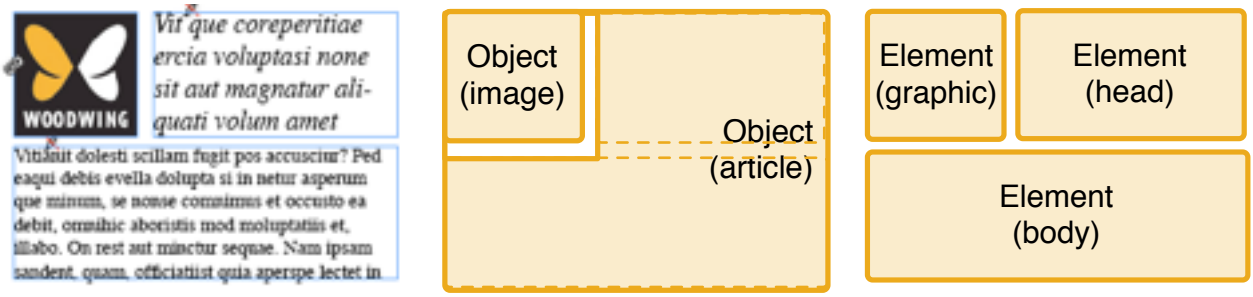
[Since 9.4] When two articles would share the same Element IDs (GUIDs), placing these articles together on the same layout could lead in content loss. Therefore, if a client other than InDesign, InCopy or InDesignServer calls the CreateObjects service (with Lock=*false*) for a WCML article, Enterprise Server generates new Element IDs (GUIDs) and updates the Elements and the article WCML before saving it in the database and filestore.

Articles and graphics

Articles can consist of a mix of text frames and graphic frames. The figure below shows an article with one graphic frame and two text frames, as shown on the left. In the middle, it shows there is just one object involved, which is the article. On the right, it shows that there are three elements.



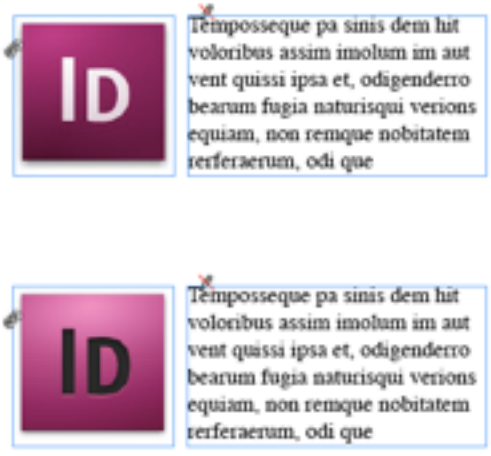
The very same example could be made differently. The figure below shows a graphics frame that holds a placed image object. (Note the little chain icon on the left of the butterfly, which is shown instead of the little pencil icon on top.) The two text frames belong to the article object, as shown in the middle. Also here, there are three elements involved, as shown on the right.



Placements and Editions

Placed objects (such as articles and images) can be assigned an Edition on the layout. InDesign allows users to set Editions per story element (changing one frame affects all frames that belong to the same element). Nevertheless, Editions are tracked by Enterprise at a more granular level, which is per placement.

An example. Imagine you have written a manual about your InDesign CS3 plug-in. Then CS4 comes out, but you are still doing heavy maintenance to the document and update your readers on a regular basis. Actually, the CS4 update doesn't have much impact to your document. It affects just some articles and the InDesign logo that is used in several places. Obviously, you would like to share as much as possible between the two manuals and publish both simultaneously. This is where editioning could help. By simply placing the CS3 and CS4 logo in the same InDesign layout, and tagging CS3 Edition for one and CS4 Edition for the other. The figure on the right shows how the result could look like in InDesign.



When saving the layout, the request from InDesign (SaveObjects) looks like the fragment shown below. because there are three objects related, there are three *Related* elements. They are all on the same page and share the same parent. For the images, a specific Edition is specified. The article is published for both Editions, and so nil is given, which means all Editions.

Note that adding CS5 and CS6 Editions later (in the Maintenance pages), this article will still get published for those future Editions. But there won't be any logos available yet, which must be created and placed on the layout at that time, just as it was done in this example for CS4.

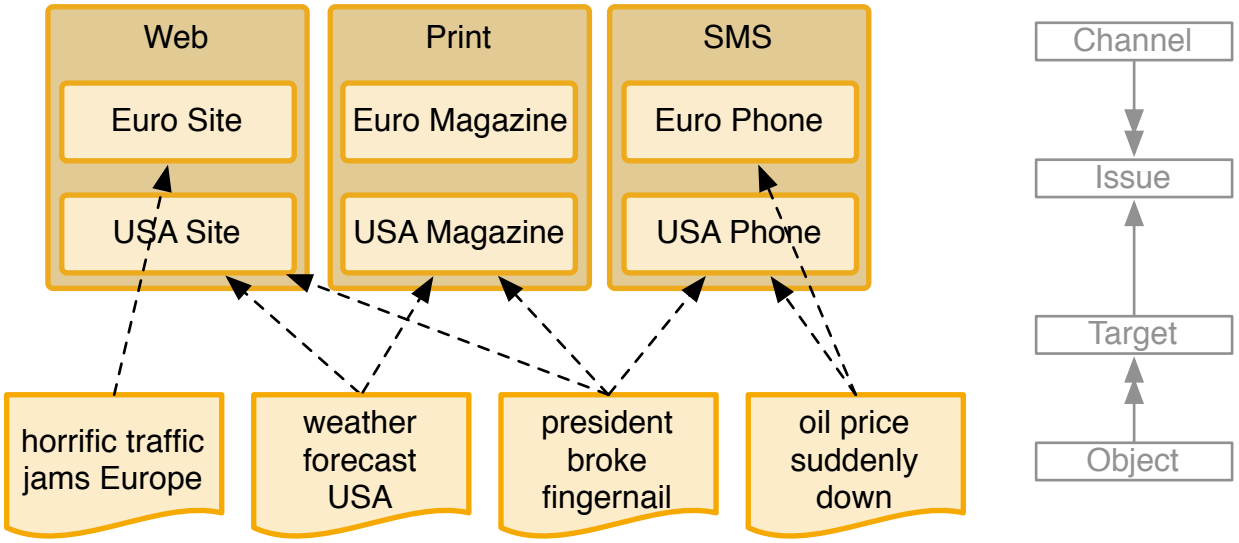
```
...
<Relations>
      <Relation>                    <!-- CS3 logo -->
            <Parent>1710</Parent>
            <Child>1711</Child>
            <Type>Placed</Type>
            <Placements>
                  <Placement>
                        <Page>1</Page>
                        <Element>graphic</Element>
                        ...
                        <Edition>
                              <Id>1</Id>
                              <Name>CS3</Name>
                        </Edition>
                        ...
      <Relation>                    <!-- CS4 logo -->
            <Parent>1710</Parent>
            <Child>1712</Child>
            <Type>Placed</Type>
            <Placements>
                  <Placement>
                        <Page>1</Page>
                        <Element>graphic</Element>
                        ...
                        <Edition>
                              <Id>2</Id>
                              <Name>CS4</Name>
                        </Edition>
                        ...
      <Relation>                    <!-- article -->
            <Parent>1710</Parent>
            <Child>1713</Child>
            <Type>Placed</Type>
            <Placements>
                  <Placement>
                        <Page>1</Page>
                        <Element>body</Element>
                        ...
                        <Edition xsi:nil="true"/>
                        ...
```

Targets

Objects can be targeted for an Issue. This means it is intended to get published for that Issue. During creation, or at the end of the selection & gathering phase, objects typically get targeted. During production, users could add (or remove) targets to send them through additional publication channels. After production, admin users might postpone some objects (that could not get published) to the next Issue by changing targets. With Enterprise 5 (or earlier), objects were always targeted for just one Issue though.

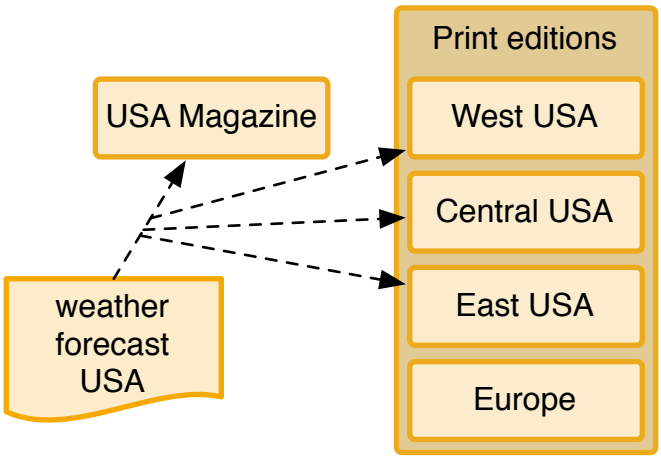
Since Enterprise 6, objects can have zero, one or many targets. Zero targets are especially needed to support selection & gathering. Many targets are needed to support multiple channeling.

The left the figure below shows some example objects targeted for several Issues. Targets are represented by dashed arrows. Each Issue belongs to a certain channel, such as “Euro Site” which belongs to the “Web” channel. The figure on the right shows how the abstract Enterprise entities are related to each other.



Target Editions

For print channels, admin users can setup Editions per channel. (This is done by means of the Maintenance pages, see the Admin Guide for details.) When an object is targeted for an Issue that belongs to a print channel, end users can select any of the configured Editions in the workflow dialogs. Selected Editions are tracked by Enterprise per target. The figure to the right illustrates how Editions relate to targets. (It zooms into a fragment of the target example shown above.) As shown, the user has selected three Editions for the target.



When the object should be targeted for all Editions, client applications should pass xsi:nil=”true” attribute for the *Editions* element. This is an exceptional meaning of the nil attribute; for other entities (other than Editions) nil means that the data is not provided, which implies existing data needs to remain untouched in the database.

Object targets

During create and save operations (CreateObjects and SaveObjects workflow services) the object targets can be set. These operations only work for objects that are locked by the current user.

Important: When an object is locked for editing by the current user, its targets can be set by other users in the meantime. Therefore, the targets sent through the create and save operations are assumed to be complete.

Since Enterprise 6, the *TargetMetaData* element are obsolete. (For more details, see Structural change for multiple channels.) Instead, the *Targets* element should be used and the *TargetMetaData* element should be nullified (by setting the xsi:nil=”true” attribute).

Example of *Target* usage:

```
<Object>
      <MetaData>
            ...
            <TargetMetaData xsi:nil="true" />
            ...
      <Targets>
            <Target>
                  <Issue>
                        <Id>421</Id>
                        <Name>USA Magazine</Name>
                  </Issue>
                  <Editions>
                        <Edition>
                              <Id>801</Id>
                              <Name>West USA</Name>
                        </Edition>
                        <Edition>
                              <Id>802</Id>
                              <Name>Central USA</Name>
                        </Edition>
                        ...
```

When objects are locked by someone else, client applications can still change the object targets. This can be done by calling the following workflow services:

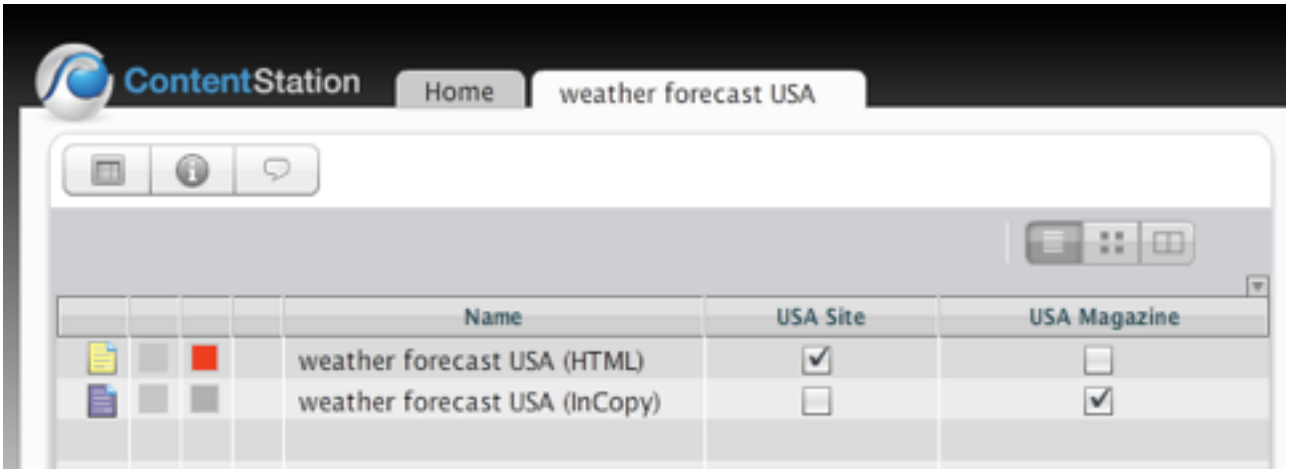
- CreateObjectTargets
- UpdateObjectTargets
- DeleteObjectTargets

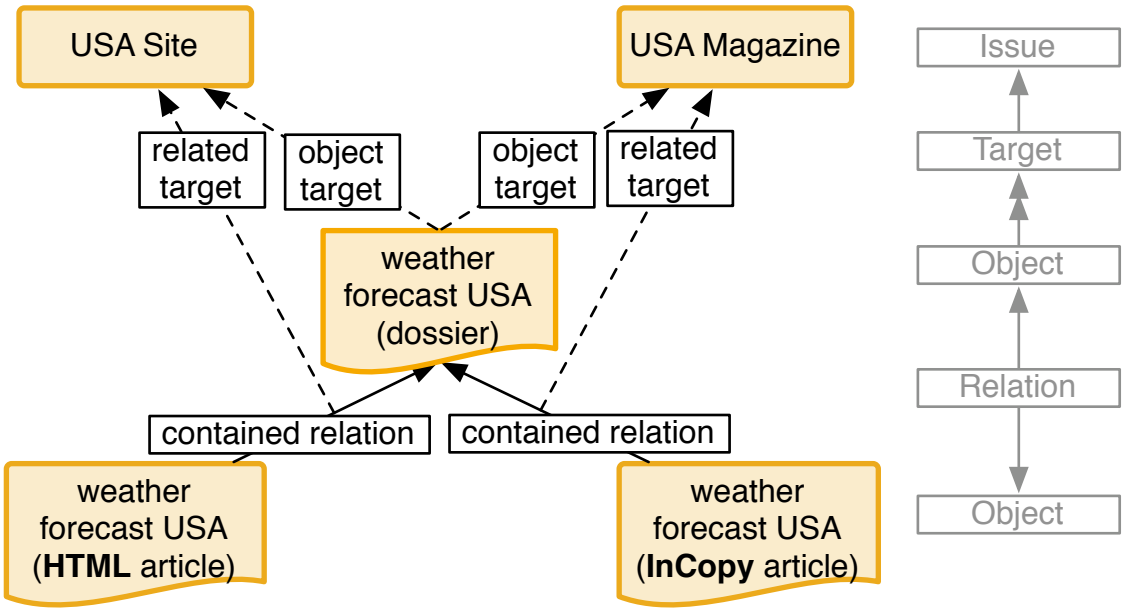
Important: Similar to the create/save operations, the *UpdateObjectTargets* service assumes that the passed object targets are complete.

Related targets

Objects can inherit targets from other objects. This typically happens when objects are related to each other. Let’s take the “weather forecast USA” example and assume it consist of two articles and a Dossier. One article is in HTML format, and the other in InCopy format. The Dossier is given two object targets: “USA Site” and “USA Magazine”. In Content Station the Dossier looks as follows:

As we can see, the user has targeted each article to a different issue. Now, the Dossier has object targets, but the articles each have a related target. The figure below shows how that is tracked in Enterprise’s data model.



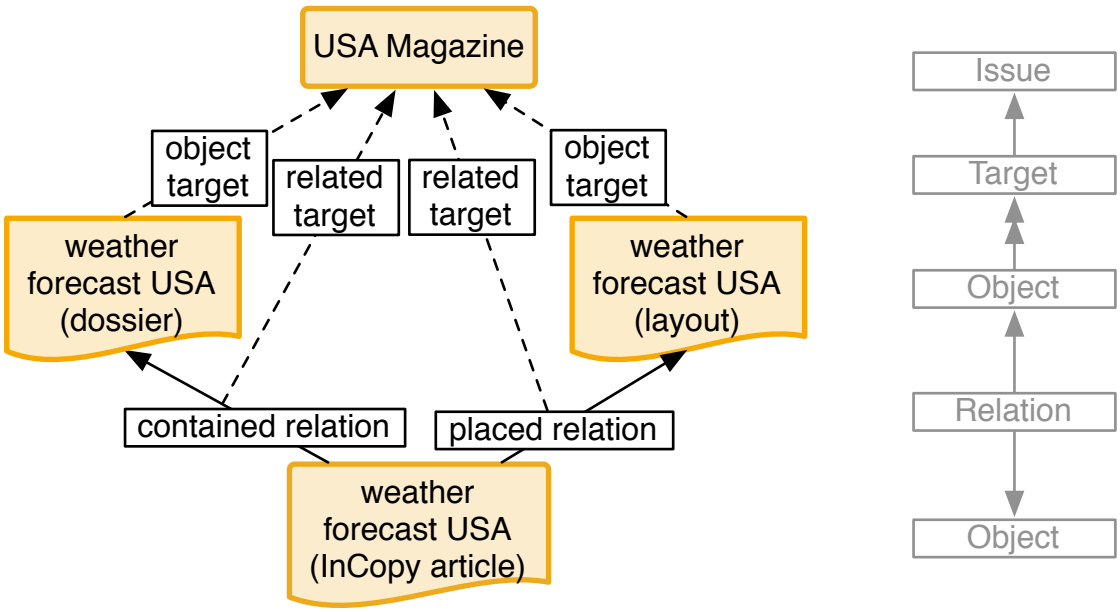


When the user tags the related targets for the articles, Content Station calls the *UpdateObjectRelation* service. For the Dossier-article relation, it passes the targets as shown in the following fragment:

```
<UpdateObjectRelations>
      ...
      <Relations>
            <Relation>
                  <Parent>1461</Parent>
                  <Child>1464</Child>
                  <Type>Contained</Type>
                  ...
                  <Targets>
                        <Target>
                              <Issue>
                                    <Id>89</Id>
                                    <Name>USA Magazine</Name>
                              </Issue>
                              <Editions xsi:nil="true"/>
                        </Target>
                        ...
```

Let's involve a layout in this example. The layout is just targeted to the print issue "USA magazine". When the article is placed on the layout, an object relation is created between

both. And again, upon the relation, a new relation target build. The following figure shows how this is tracked in the Enterprise model:



Structural change for multiple channels

Since Enterprise 6, various structural changes have been implemented which are mainly about the need to support multiple publication channels. This had quite some impact on the workflow WSDL as described below. This paragraph is written for those who you are migrating from Enterprise 3/4/5 (to 6 or higher).

- New insights told that objects are no longer targeted to a Brand (formerly called Publication). The object is owned by the Brand, and therefore the *Publication* element needed to move from *TargetMetaData* to *BasicMetaData*.
- At Brand level, Issues and Editions could be configured. There was a need to introduce another concept, namely Publication Channels. Publication Channels are configured as part of the Brand, and Issues and Editions are configured as part of the Publication Channel.
- Objects could be assigned to just one Issue at the same time. Also here, another concept was introduced, called a target. A target is one Issue with optionally a set of Editions. An object can be ‘targeted’ to many Issues, which is established by having multiple targets.
- The *TargetMetaData* element could hold one ‘target’ only (which was one Issue with many Editions). To support many, a new element named *Targets* was introduced directly under the *Object* element. This move was also done because targets can change while objects are locked (while other properties can’t). To visualize this, the targets do no longer belong under the *MetaData* element.
- The term 'section' is too print oriented and does not cover its intended use. Also, for other channels than print, the term 'section' is not appropriate. Therefore it was changed to 'category', not only for the GUI, but for the WSDL as well. For channels other than print, the term section did not have much meaning. It is therefore changed to Category, not only for the GUI, but for the WSDL as well.
- Just like the *Publication* element, the *Section* is nothing like a target anymore. Therefore it is moved to *BasicMetaData* element too, and renamed to *Category*.

Although the WSDL structure has been changed, Enterprise 6 and 7 still support requests that are structured the old way for backwards compatibility reasons. Nevertheless, this will no longer be done for future versions. Server Plug-in developers don't have to deal with this since the core server does restructure on-the-fly before calling the plug-ins for incoming requests, and after outgoing responses. This includes all changes mentioned above. So Server Plug-ins should deal with the new structure only, while there could be still old clients talking.

Enterprise 5 (or lower)

```
<Object>
   <MetaData>
      <BasicMetaData>
         ...
      </BasicMetaData>
      <TargetMetaData>
         <Publication>
            <Id>1</Id>
            <Name>WW News</Name>
         </Publication>
         <Issue>
            <Id>1</Id>
            <Name>2nd issue</Name>
         </Issue>
         <Section>
            <Id>1</Id>
            <Name>News</Name>
         </Section>
         <Editions>
            <Edition>
               <Id>1</Id>
               <Name>North</Name>
            </Edition>
            <Edition>
               <Id>2</Id>
               <Name>South</Name>
            </Edition>
         </Editions>
      </TargetMetaData>
      ...
```

Enterprise 6 (or higher)

```
<Object>
   <MetaData>
      <BasicMetaData>
         ...
         <Publication>
            <Id>1</Id>
            <Name>WW News</Name>
         </Publication>
         <Category>
            <Id>1</Id>
            <Name>News</Name>
         </Category>
         ...
      </BasicMetaData>
      <TargetMetaData xsi:nil="true" />
      ...
   <Targets>
      <Target>
         <Issue>
            <Id>1</Id>
            <Name>2nd issue</Name>
         </Issue>
         <Editions>
            <Edition>
               <Id>1</Id>
               <Name>North</Name>
            </Edition>
            <Edition>
               <Id>2</Id>
               <Name>South</Name>
            </Edition>
         </Editions>
...
```

Enterprise in Action

When client applications are used by end users, the Enterprise Server is continuously requested by those clients to execute various operations. For common scenarios, this section gives a bird's-eye view of such interactions. This is especially done to help integrators understand the main flow. With this consensus, it is assumed that less frequently used scenarios and variations of the main flow become much easier to examine and understand. Those are out-of-scope for this guide.

Planning in Action

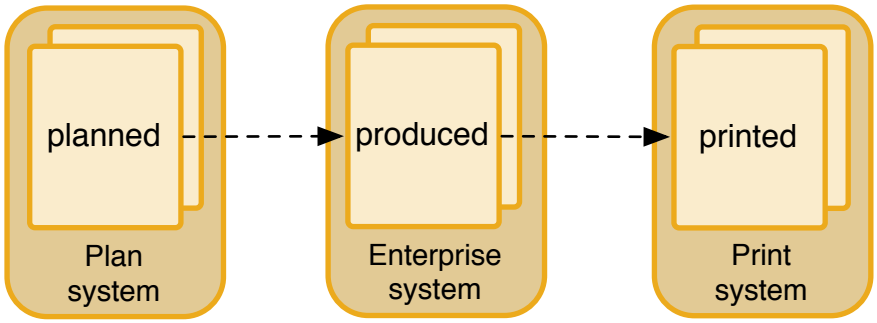
Planning is an optional feature which is taken care of by a third-party plan system integrated in Enterprise. Which plan system to integrate is the customer's choice. Examples are Journal Designer (DataPlan) and Timone (Tell). Planning integrations are typically established through the planning interface, but in some cases with extra help of the workflow interface.

Planning integrations allow planners to synchronize their issue plans to the production system. Customers can work in two different way, both of which are supported:

- **Throw the plan over the fence** - Some customers like to create a full plan, synchronize it, continue in production, and no longer care about the plan. At this point, the planning system could shut down and progress is monitored in production (using the Publication Overview). This is the most simple form, wherein production takes over the lead.
- **Continuous synchronization** - Some customers have a dedicated planner who takes the lead from creation till the planned issue gets finally printed. Plans are continuously synchronized with production while many editors and layout designers are producing pages. The idea is that the planner is the boss and so plans should be respected in production. For example, layout designers in production are bound to the planned pages and their numbering, but they have full control over the placed objects. Placed adverts are the only exception in this and are created by the planner. Advert positions on pages are important since ads have different prices depending on their positions. Layout designers have to respect that, but still have some freedom to make them fit onto the page nicely.

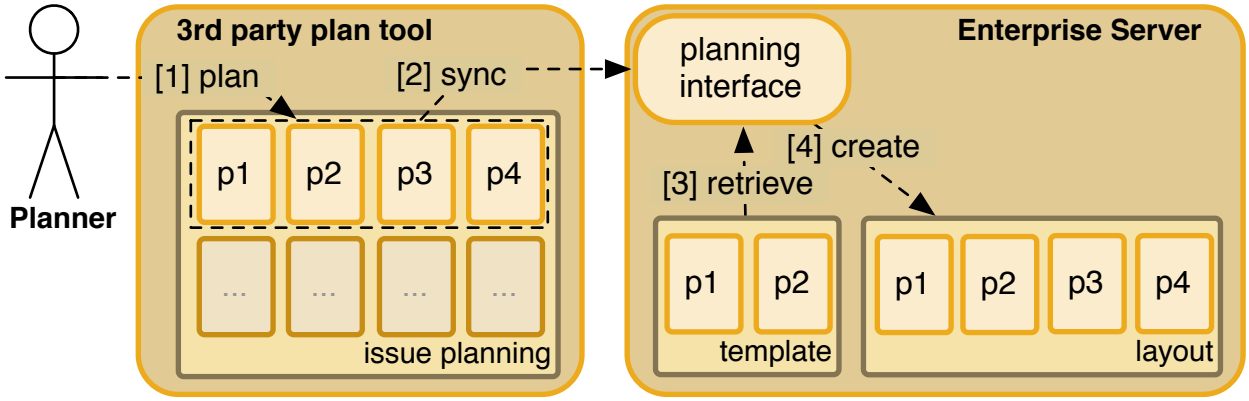
Page planning

Pages are used within a print-oriented workflow. Pages can be planned, produced and printed. In this context, Enterprise does the production, and third-party systems do the planning and printing.



While the planner is creating pages in the plan system, planned pages also get created within the Enterprise database through the planning interface. This is done by creating layout objects from layout templates, which reside in the Enterprise database. The template pages are replicated for the layout for as many pages that are planned (by the planner).

The following figure shows a planner with four pages planned [1]. The third-party planning tool picks up those pages and synchronizes [2] the four pages into the Enterprise system through the planning interface. The Enterprise Server retrieves [3] the requested layout template and creates a new layout [4] from it.

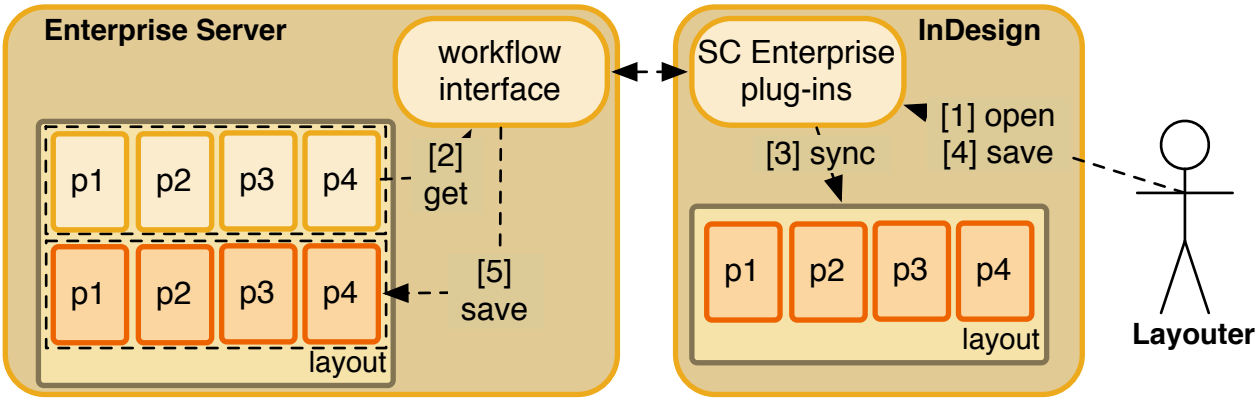


At this point, a layout is created in Enterprise database with four planned pages. The InstanceType (as defined at workflow WSDL) is set to ‘Planning’ for these pages. Also, a message is sent to the layout and a flag is set. The flag indicates that the attention of the layout designer is needed; the message informs the layout designer about the creations (or changes) made to the planned layout.

Once pages are planned, the production can pick them up. The layout designer finds the flagged layouts that were planned at his/her query results. (To see the flags, special query columns must be configured; for more information, see the Admin Guide.) At this point, the binary layout file itself is a rough copy (made by the server) of the layout template. Once the layout designer opens it, this file is sent from the server to the InDesign client, along with the planned(!) pages. Once opened in InDesign, the Smart Catalog plug-ins are able to create concrete pages (for production) based on the arrived planned pages and update the binary file. Before doing so, a dialog is raised to ask the layout designer whether or not to do this automatically. The layout designer can cancel this and do it manually. When the message is confirmed though, it is done automatically by the plug-ins. This includes the following operations:

- Pages are added or removed, because the template could have less or more pages than the layout requires. New pages are based on the master page, as indicated by the planner.
- Pages are renumbered, to reflect the Issue planning. Template pages typically start with page 1, but layout pages could start with any number, depending on their position in the planning.
- Adverts are placed on pages. For more details, see advert planning.

Let’s recap the step wherein planned pages are taken into production (see the figure below). First, the layout designer opens a layout [1] and the plug-ins request it to get [2] it from the database. The planned pages (orange) are synchronized [3] into production by the plug-ins. The produced pages (red) are saved [4] on the layout designer’s command and stored [5] in the database.



Now there are two sets of pages: planned and produced. The produced pages have the InstanceType (as defined at workflow WSDL) set to ‘Production’.

Workflow in Action

Common sequences

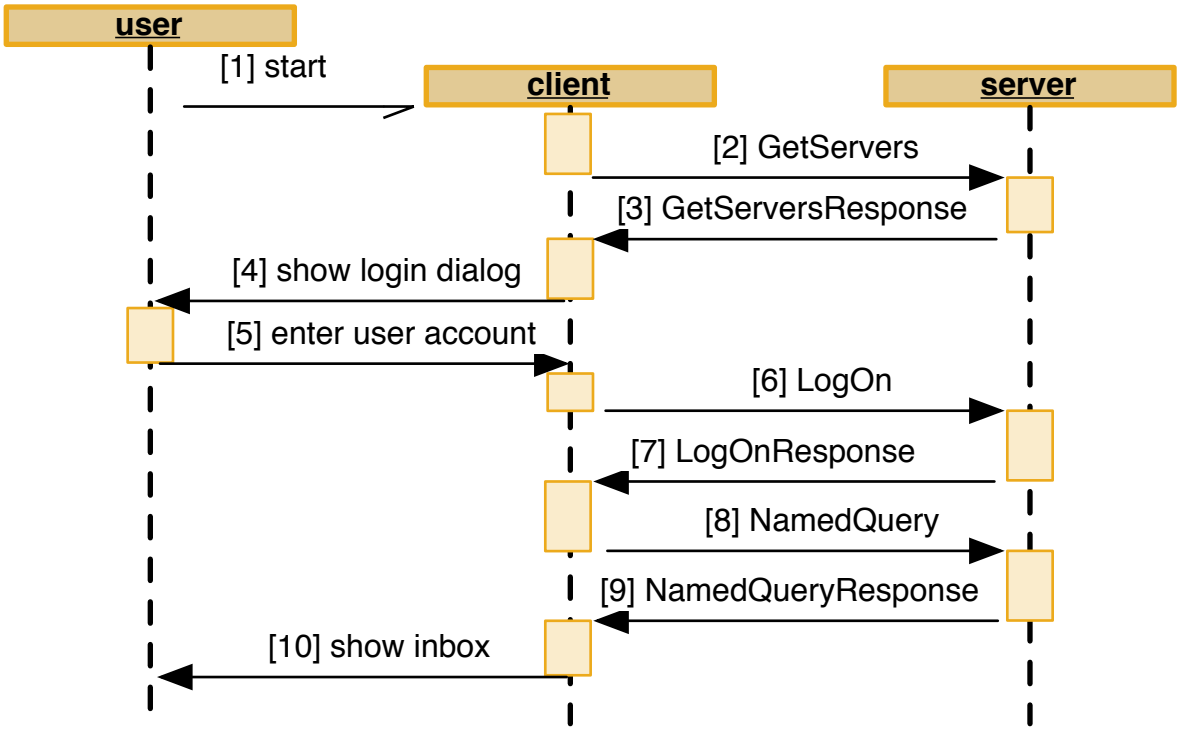
Client applications implement all kinds of features by requesting services in the server. Many services stand on their own, but there are some logical sequences that are commonly used. This paragraph takes you by the hand running through the most important sequences while explaining the system at high level.

Startup and login

The figure below shows an end user (on the lleft) working with a client application (center) which talks through SOAP with the server (right). When the user starts [1] the client application, it does some initialization and requests [2] the server to return [3] a list of the configured* application servers.

* Note: These servers are configured by admin users on the client machines (using the SCEnt:Servers setting in the WWSettings.xml file) or at a centralized server machine (using APPLICATION_SERVERS setting in the configserver.php file). (For configuration details, see theAdmin Guide.) The example assumes that the application servers are configured server side. If not, steps [2] and [3] are skipped.

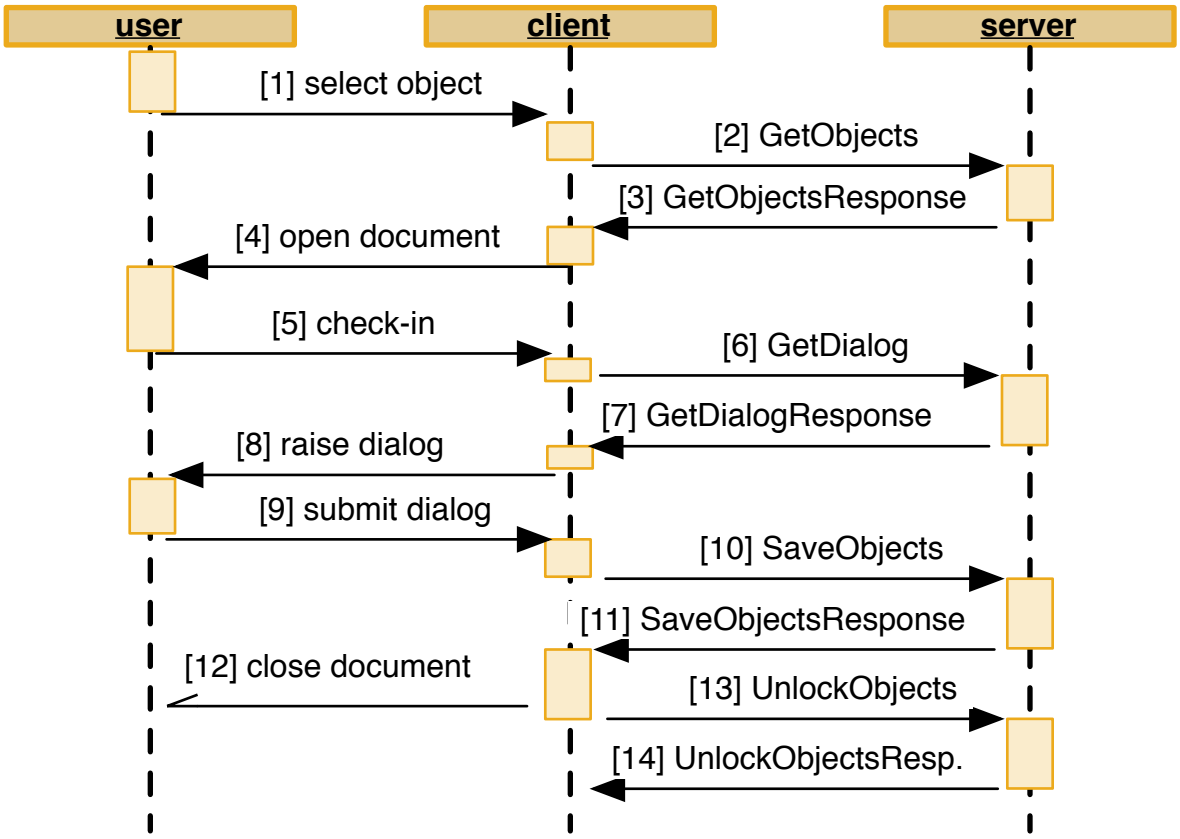
This list is shown [4] in the login dialog. The user enters his / her name and password, picks one of the listed application servers and submits [5] the dialog. The client requests the picked server to login [6] and passes the name and password entered by the user. The server checks the user account and returns [7] a newly created ticket to start a session. This ticket is used by the client for all next following requests.



Implicit to the login operation, clients fire a search request [8]. This can be the inbox (such as Content Station does) or the last-used query (such as InDesign does). The server runs the query in the database and returns [9] a list of objects. The client lists [10] the objects as rows in a table view.

Editing a document

Once the client is started and the user is logged in, the user can start working. Let's say that the user want to edit an article, layout or image. The figure below shows how that is done. The user picks [1] one of the listed objects from the query results. The client takes the object id of the selected row and requests [2] to lock that object id and retrieve the native file of the object. When not already locked, the server locks the object, retrieves the native file from the file store and sends [3] it back (as DIME attachment). The client saves the file locally and opens it [4] in its own application (such as InDesign) or starts another application (such as Photoshop). Now the user starts editing the file and does a check-in [5] once completed. The client requests [6] for the dialog definition. The server queries the definition (as defined on the admin pages) from the database and returns it [7]. The client builds the dialog based on the definition and shows [8] it to the user. The user adjusts some properties in the workflow dialog [9] such as the status. The client uploads [10] the file (as a DIME attachment) back to the server, which creates a new version and stores it in the file store. The updated object properties are returned [11] when successful. When the client did not unlock the object while saving [10], it unlocks afterwards [13] for which the server does return [14] no info.



Object locking

Object files can be opened for editing by any user (with “Open for Edit” access rights). Doing so, objects are locked to prevent two users accidentally working on the same content at the same time. Once an object is locked, other users can read it, but cannot update the object files nor the object properties. Other aspects to objects are not locked though, such as targets, messages, and relations. Those could be changed by other users while the object is locked.

Creating and opening

When creating new objects, the lock can be preserved to let the user continue editing the content. This is mostly done for a “Save Version” action of an article or layout for which a *CreateObjects* service is called, as shown in the fragment below. When the user does a “Check In” action to a new file, the same request is fired but then the *Lock* parameter is set to *false* instead. This is mostly done for new image file uploads.

```
<CreateObjects>
      <Ticket>5863c2527QG5KBoTnmSEX8yPrX6P9S1IF7Wxbnfj</Ticket>
      <Lock>true</Lock>
      <Objects>
            <Object>
                  <MetaData>
                        <BasicMetaData>
                              <ID xsi:nil="true"/>
                              ...
```

For existing objects, a lock can be set through the *GetObjects* service by setting the *Lock* parameter to *true* as shown in fragment below.

```
<GetObjects>
      <Ticket>5863c2527QG5KBoTnmSEX8yPrX6P9S1IF7Wxbnfj</Ticket>
      <IDs>
            <String>1720</String>
      </IDs>
      <Lock xsi:type="xsd:boolean">true</Lock>
      ...
</GetObjects>
```

Storing and closing

When the user has changed the object properties and the object file content, a new version can be created, and the lock can be released simultaneously. For new objects, the *CreateObjects* service is called, and *SaveObjects* service for existing objects. This is mostly done for “Check In” actions, as shown here:

```
<SaveObjects>
      <Ticket>5863c2527QG5KBoTnmSEX8yPrX6P9S1IF7Wxbnfj</Ticket>
      <CreateVersion xsi:type="xsd:boolean">true</CreateVersion>
      <ForceCheckIn xsi:type="xsd:boolean">false</ForceCheckIn>
      <Unlock xsi:type="xsd:boolean">false</Unlock>
      <Objects>
            <Object>
                  <MetaData>
                        <BasicMetaData>
                              <ID>1720</ID>
                              ...
```

Alternatively, when all object properties and file content is already saved before, just the lock can be released through the more cheaper *UnlockObjects* service as shown in the next fragment.

```
<UnlockObjects>
      <Ticket>5863c2527QG5KBoTnmSEX8yPrX6P9S1IF7Wxbnfj</Ticket>
      <IDs>
            <String>1720</String>
      </IDs>
      ...
</UnlockObjects>
```

Setting properties

Properties can be changed for a single object by using the *SetObjectProperties* service and for multiple objects (since 9.2) by using the *MultiSetObjectProperties* service. Note that most object properties can be changed, but the contents of the object files remains untouched and so properties that are strongly content related can also not be changed either. Because the Version property is content related, it won't get increased.

There are two ways of locking, each with their own advantages and disadvantages:

- **Optimistic locking.** This method is used when the file was not previously locked by the client and the service is called directly. In this scenario, the file is not actually locked during the time the user updates the properties. It is assumed that the changes can be submitted without the file being locked by another user in the mean time. This method was implemented in Content Station 7.
- **Pessimistic locking.** In this method, the client application first locks the object (through *GetObjects* or *CreateObjects* service) and then calls the *SetObjectProperties* service. It then needs to release the lock through the *UnlockObjects* service afterwards. This way, before the dialog is shown to the user, it could already fail (when someone else has locked it in the meantime) and an error raises. This method was implemented in Smart Connection 7.

[Since 9.2] The *MultiSetObjectProperties* service is designed to simultaneously modify a few properties for many objects. For a selection of multiple objects, it is significantly faster to use this service than repeatedly calling the *SetObjectProperties* service for each object.

Important notes for Server Plug-ins:

- Instead of "Set Properties", users can use the "Send To" or the "Send To Next" actions as well. Doing so, client applications fire the *SendTo* workflow request to the server. Internally, the server redirects this service to the *SetObjectProperties* service as soon as it arrives, even before any Server Plug-in connector is called. The *SendTo* service is actually a simplified version of the *SetObjectProperties* service. During redirection, the server enriches some object information such as the object targets. This is all done to simplify property interception using custom Server Plugins. In other terms, the *WflSendTo* service connector is never(!) called and *WflSetObjectProperties* is called instead.
- [Since 9.2] When implementing the *WlfSetObjectProperties* connector it is mandatory to also implement the *WflMultiSetObjectProperties* connector for the plug-in, and vice versa. Failure to do so will result in validation errors on the Server Plug-ins or Health Check pages.
- [Since 9.2] When implementing the *WlfSendTo* connector it is mandatory to also implement the *WflSendToNext* connector for the plug-in, and vice versa. Failure to do so will result in validation errors on the Server Plug-ins or Health Check pages.

Automatic workflow status assignment [since 8.2]

Since Enterprise 8.2 it is allowed to create objects without specifying a status. Enterprise Server will then automatically take the first status configured in the workflow (for the given object type and Brand). When there are no statuses defined, but the Personal Status feature is enabled, a Personal Status will be assigned to the newly created object. SOAP clients calling the CreateObjects service should set the State to nil as follows:

```
<CreateObjects>
      <Objects>
            <Object>
                  <MetaData>
                        ...
                        <WorkflowMetaData>
                              ...
                              <State xsi:nil="true">
                              ...
                        </WorkflowMetaData>
                        ...
                  </MetaData>
            </Object>
      </Objects>
</CreateObjects>
```

Search

Object searching is the heart of Enterprise; it is a fundamental and complex centralized feature of the workflow system pumping objects through its veins. To breakdown complexity and to differentiate between all kind of search features, Enterprise has introduced several functional areas, as explained in this paragraph. The search results are explained in a following paragraph.

Freestyle search

Enterprise supports freestyle search: users can search for objects their own way using various parameters. This feature is supported by the *QueryObjects* service. This subparagraph describes several ways of using this service.

Full text search

The user can type just a piece of text to search for, without indicating to which object property the filtering should be applied. In fact, the user is grabbing for text through any property. The figure below shows how this search looks in Content Station.

When the user clicks the search button, the client application fires the *QueryObjects* request to the server. The fragment below shows how this looks. Since there is actually no specific property involved, it passes the reserved property name *Search* to trigger this feature in the server.

```
<QueryObjects>
      <Ticket>...</Ticket>
      <Params>
            <QueryParam>
                  <Property>Search</Property>
                  <Operation>=</Operation>
                  <Value>hello world</Value>
            </QueryParam>
      </Params>
      ...
```

Browse search

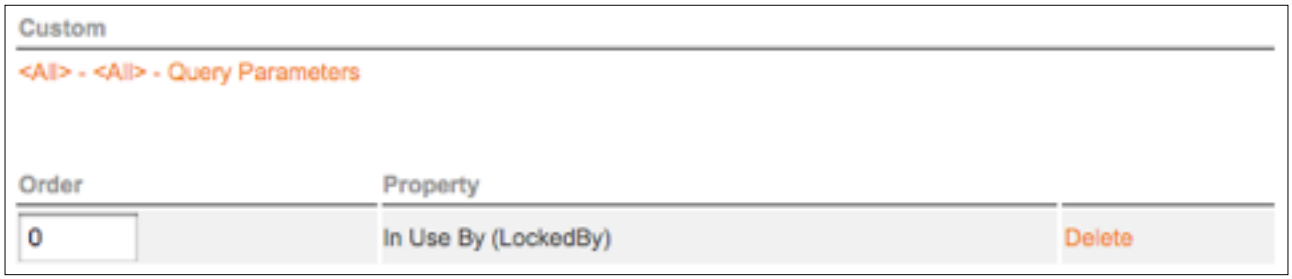
The most direct and simple way of searching is by selecting one of the predefined filters (also known as Search Modes). The figure below shows that the user has chosen the “WW News” Brand and the “sport” Category using Content Station.

After clicking the search button, Content Station fires a search query, as shown in the fragment below. The ids of the Brand (formerly publication) and Category are passed.

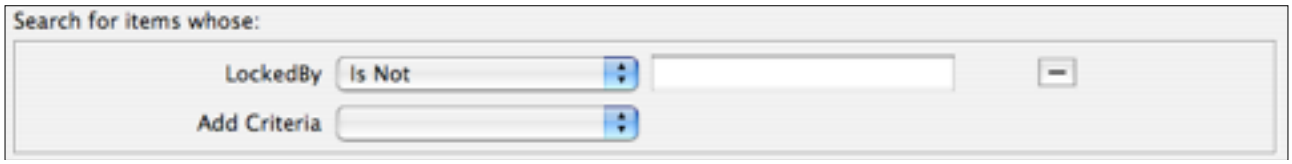
```
<QueryObjects>
      <Ticket>...</Ticket>
      <Params>
            <QueryParam>
                  <Property>PublicationId</Property>
                  <Operation>=</Operation>
                  <Value>1</Value>
            </QueryParam>
            <QueryParam>
                  <Property>CategpryId</Property>
                  <Operation>=</Operation>
                  <Value>3</Value>
            </QueryParam>
      </Params>
      ...
```

User Query

In the “Dialog Setup” Maintenance page, system admin users can define what parameters are allowed for filtering within User Queries (the Search Modes “Search” or “Custom Search” in the client GUI). (See the Admin Guide for configuration details.) In the example below, the *In Use By* property is configured for the query parameters. Internally this property is named *LockedBy*.



A user logged in to a client application can add the pre-configured query parameters to any custom search query. The figure below shows a fraction of the “Search Criteria” dialog in InDesign. In this example, the user has just added the *LockedBy* property to his/her query:



InDesign then fires the query request as show in the fragment below. Now it uses the unequal operation (!=) and leaves the value empty, just like the user did. The server then returns all objects that are currently in use. See the last subparagraph for an example of how the results usually look.

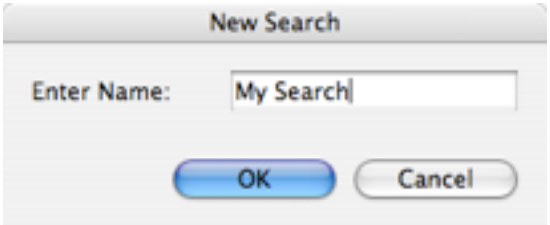
```
<QueryObjects>
      <Ticket>...</Ticket>
      <Params>
            <QueryParam>
                  <Property>LockedBy</Property>
                  <Operation>!=</Operation>
                  <Value/>
            </QueryParam>
      </Params>
      ..
```

The full set of operators can be found in the workflow WSDL. The fragment below shows the definition (made for Enterprise 7) in two-fold. On the left, the WSDL is opened in a plain-text editor and on the right it is opened in an XML editor (or Web browser). The plain-text version shows that some operators use “<”. This is the XML escaped representation of “<” which means “less than”. In an XML editor (or Web browser) it shows the escaped version. The SOAP requests contain the escaped “<” character because “<” is a reserved symbol in XML. Nevertheless, this is hidden from client applications (using SOAP/XML tools) and from Server Plug-ins; they simply use the unescaped “<” character.

```
<simpleType name="OperationType">
      <restriction base="string">
            <enumeration value="&lt;"/>
            <enumeration value=">"/>
            <enumeration value="&lt;="/>
            <enumeration value=">="/>
            <!-- 4.2 - Greater Than or Equal -->
            <enumeration value="="/>
            <enumeration value="!="/>
            <enumeration value="contains"/>
            <enumeration value="starts"/>
            <enumeration value="ends"/>
            <enumeration value="within"/>
            <!-- 5.0 Time interval -->
      </restriction>
</simpleType>
```

```
- <simpleType name="OperationType">
  - <restriction base="string">
      <enumeration value="<"/>
      <enumeration value=">"/>
      <enumeration value="<="/>
      <enumeration value=">="/>
      <!-- 4.2 - Greater Than or Equal -->
      <enumeration value="="/>
      <enumeration value="!="/>
      <enumeration value="contains"/>
      <enumeration value="starts"/>
      <enumeration value="ends"/>
      <enumeration value="within"/>
      <!-- 5.0 Time interval -->
    </restriction>
  </simpleType>
```

Saving a User Query



Enterprise client applications allow users to save their queries. The user can fill in a name as shown in the figure to the right. At this stage, the query is saved in the memory of the client application. When the user logs out, the queries are stored in the server through the *LogOff* request, as shown in the fragment below. The WSDL tells that the *Value* element must be a string. Adding XML elements as a string would make the SOAP request invalid. Therefore the string is escaped. Escaped characters are hard to read for humans, so in the example below they are left out and green colored italic formatting is used instead.

```
<LogOff>
      <Ticket>...</Ticket>
      <SaveSettings xsi:type="xsd:boolean">true</SaveSettings>
      <Settings>
            <Setting>
                  <Setting>UserQuery3_My Search</Setting>
                        <Value>
                              <?xml version="1.0" encoding="UTF-8" standalone="no" ?>
                              <UserQuery>
                                    <QueryParam>
                                          <Type>string</Type>
                                          <Property>LockedBy</Property>
                                          <Operation>!=</Operation>
                                          <Value></Value>
                                    </QueryParam>
                              </UserQuery>
                        </Value>
                  </Setting>
            </Setting>
...
```

The next time a user logs in again, the saved query is passed back (using the same structure) through the login response in the LogOnResponse -> Settings element. The query is then made available to the user to run again (or to make justifications).

Search by date

In a QueryObjects request, parameters can be added by specifying a Property, Operator and a Value. For any of the date object properties such as Created, Modified and Deleted, the operators ‘starts’ and ‘within’ can be used. A specification of the Value format can be found on w3.org: XML Schema Part 2: Datatypes Second Edition. This format is partially implemented by Enterprise Server as explained in the following paragraphs.

The ‘starts’ operator

Enterprise validates the following syntax for the Value attribute:

[-]? [P] [017] [D]

With the following meaning:

- [-]? The - (minus) is optional. If it is present, it requests to search backward in time, else forward in time. Note that for the currently supported date properties, looking forward is rather exceptional and therefore not described here.
- [P] The P is mandatory and denotes a duration field.
- [017] The number of days (mandatory), which can be 0, 1 or 7. The handling of the number of days is as follows:
  - 0: Everything where the date matches today (from 00:00:00 up to 23:59:59).
  - 1: Everything where the date matches yesterday (from 00:00:00 till 23:59:59, so excluding today). This requires the leading minus.
  - 7: Looks at the last full week, starting from the first day of last week up to the last day of last week (so excluding this week). This requires the leading minus.
- [D] The D is mandatory and indicates that the duration is in Days.

Valid examples:

- -P7D (= last full week, excluding this week)
- P0D (= today)

Example request for objects that were modified yesterday:

```
<QueryObjects>
      <Ticket>...</Ticket>
      <Params>
            <QueryParam>
                  <Property>Modified</Property>
                  <Operation>starts</Operation>
                  <Value>-P1D</Value>
            </QueryParam>
      </Params>
      ..
```

The 'within' operator

Enterprise validates the following syntax for the Value attribute:

[-]? [P] [T]? [0-9]+ [DMH]

With the following meaning:

[-]?	The - (minus) is optional. If it is present it requests to look an x number of duration units before now. If it is absent it requests to look for a number of duration units from now.
[P]	The P is mandatory and denotes a duration field.
[T]?	The T is optional. If specified, the duration is a Time, else a Date.
[0-9]+	The number of duration units (mandatory) represented by one or more digits.
[DMH]	The duration unit, which must be one of these characters: D, M or H. The meaning depends if T is specified: - When T is not specified: D = days, M = months. (H is not valid.) - When T is specified: H = hours, M = minutes. (D is not valid.)

Note: Unlike the 'starts' operation, there is no special handling for the number: if you request 7 days in the past it will look from 7 days in the past, and not the full week before.

Valid examples:

-PT30M (= last 30 minutes)

-P1M (= last month)

Example request for objects that were created within the last half year:

```
<QueryObjects>
      <Ticket>...</Ticket>
      <Params>
            <QueryParam>
                  <Property>Created</Property>
                  <Operation>within</Operation>
                  <Value>-P6M</Value>
            </QueryParam>
      </Params>
      ..
```

Predefined search

Enterprise supports predefined searches;: users can pick a predefined query (also called Named Query or Custom Search) and optionally use one of its parameters (if any). This feature is supported by the *NamedQuery* service. This paragraph describes several ways of using this service.

Inbox

Maybe the most frequently used query is the *Inbox* query. It shows the objects that are assigned to the user or one of the groups the user is in. From the *Inbox*, the user opens files that must be worked on. Up to Enterprise 6, the *Inbox* query was shipped as a true

Named Query. This implies that any system admin user can adjust it in the “Named Queries” Maintenance page.

Since Enterprise 7, the *Inbox* is built-in to the server. Nevertheless, it still acts as a true Named Query, as if it still exists. It is hidden from the “Named Queries” Maintenance page though, so changes cannot be made anymore.

When a user selects and runs the *Inbox* query, client applications fire the *NamedQuery* request as shown in the fragment below. Content Station does this also automatically right after the user has logged in.

```
<NamedQuery>
      <Ticket>...</Ticket>
      <Query>Inbox</Query>
      <Params xsi:nil="true"/>
      ...
```

Named Query

The “Named Queries” Maintenance page allows admin users to configure any Named Query (see the Admin Guide for details.) Let’s say he/she has made a query called “Name Search” and specified some parameters in the “Interface” field as shown in the fragment below. There are two parameters; *ObjectName* and *ObjectType*. The type has a default value *Layout* of a list with three options: *Article*, *Image* and *Layout*.

```
ObjectName,string
ObjectType,list,Layout,Article/Image/Layout
```

Enterprise 6 (and before) ships the “Name Search” query. With Enterprise 7 a full text search is directly available. Therefore the “Name Search” is no longer shipped.

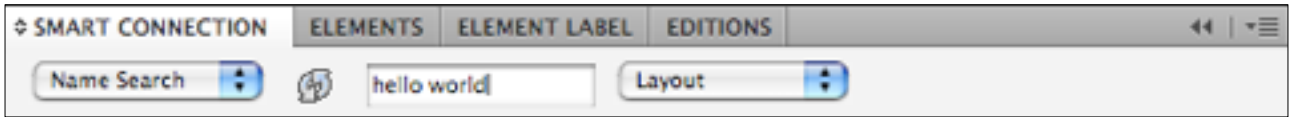
When the user logs in to Enterprise, the configured Named Queries are returned by the server through the *LogOnResponse*. The fragment below shows how that is done for our “Name Search” example query. Per query, it specifies which parameters can be used. For the example query you can find the two configured parameters.

```
<LogOnResponse>
      ...
      <NamedQueries>
            <NamedQuery>
                  <Name>Name Search</Name>
                  <Params>
                        <PropertyInfo>
                              <Name>ObjectName</Name>
                              ...
                        </PropertyInfo>
                        <PropertyInfo>
                              <Name>ObjectType</Name>
                              ...
                        </PropertyInfo>
                  </Params>
            </NamedQuery>
            ...
```

Zooming into the details of ObjectType, the default value and its option are included:

```
<PropertyInfo>
      <Name>ObjectType</Name>
      <DefaultValue>Layout</DefaultValue>
      <ValueList>
            <String>Article</String>
            <String>Image</String>
            <String>Layout</String>
      </ValueList>
      ...
</PropertyInfo>
```

After login, the client applications list the queries in their search panes, allowing users to pick one. In the figure below, the user logged in to InDesign and has picked our *Name Search* query and filled in a search string “hello world” for the *ObjectName* parameter. The *Layout* option of the *ObjectType* parameter is preselected (and remained untouched by user).



By clicking the Search button, InDesign fires the *NamedQuery* request as shown in the fragment below. Both parameters as filled in by the end user are sent along the request.

```
<NamedQuery>
      <Ticket>...</Ticket>
      <Query>Name Search</Query>
      <Params>
            <QueryParam>
                  <Property>ObjectName</Property>
                  <Operation>=</Operation>
                  <Value>hello world</Value>
            </QueryParam>
            <QueryParam>
                  <Property>ObjectType</Property>
                  <Operation>=</Operation>
                  <Value>Layout</Value>
            </QueryParam>
      </Params>
      ...
```

Search results

Regardless of the search technique used, the granular structure of the search results is always the same. This simplifies client applications and Server Plug-ins parsing results. Although there are two different responses specified in the workflow WSDL (*QueryObjectsResponse* and *NamedQueryResponse*), both have the same structure.

A search result is basically a table of columns and rows. Which columns are returned depends on the configurations made in the Maintenance pages:

- For the *QueryObjects* service, the returned columns are configured on the “Dialog Setup” page through “Query Result Columns” actions. (See the Admin Guide for configuration details.)
- For the *NamedQuery* service, the returned columns are determined by the SQL query as configured on the “Named Queries” page. (See the Admin Guide for configuration details.)
- Every found object in the search results is represented by a *Row* element, and its properties by *String* elements. An important assumption is that the n^{th} *String* element

corresponds with the nth *Property* element under *Columns*. The fragment below shows a search result listing two objects and three columns. The second column represents the object type. So the second *String* value of each row (object) contains the object type. The different colors show the mapping of rows to columns.

```
<QueryObjectsResponse>
      <Columns>
            <Property>
                  <Name>ID</Name>
                  <DisplayName>ID</DisplayName>
                  <Type>string</Type>
            </Property>
            <Property>
                  <Name>Type</Name>
                  <DisplayName>Type</DisplayName>
                  <Type>string</Type>
            </Property>
            <Property>
                  <Name>Name</Name>
                  <DisplayName>Name</DisplayName>
                  <Type>string</Type>
            </Property>
            ...
      </Columns>
      <Rows>
            <Row>
                  <String>1708</String>
                  <String>Article</String>
                  <String>my article</String>
                  ...
            </Row>
            <Row>
                  <String>1721</String>
                  <String>Layout</String>
                  <String>my pages</String>
                  ...
            </Row>
            ...
      </Rows>
      ...
```

Note that the *ID*, *Type* and *Name* properties are always returned, regardless of the configuration made. This is considered to be the bare minimum to provide client applications with. In other terms, clients may assume these properties are always present.

Regardless of the property, client applications and Server Plug-ins should never hard-code column positions. For example, the *Type* column is in the second position, but it can be located at any other position.

The principle shown in the following fragment may be used in your Server Plug-in (*WflQueryObjects* connector) to determine the column positions of those properties that you are looking for. When you are developing a client application, you could use this fragment as pseudo code. This custom example is only interested in *Type* and *Name* object properties. With those two column indexes available, it runs through the rows (objects) and checks the object type. For articles only, it adds “(article)” to the object name, to meet the customer’s requirements. This customization affects the search results of all clients. Obviously you can take out any other columns and perform any other custom operation to the rows.

```
final public function runAfter( WflQueryObjectsRequest $req, WflQueryObjectsResponse &$resp )
{
	// Determine column indexes to work with
	$indexes = array( 'Type' => -1, 'Name' => -1 );
	foreach( array_keys($indexes) as $colName ) {
		foreach( $resp->Columns as $index => $column ) {
			if( $column->Name == $colName ) {
				$indexes[$colName] = $index;
				break; // found
			}
		}
	}
	// Run through rows to be returned and mark articles
	foreach( $resp->Rows as &$row ) { // for all rows to return soap client
		$type = $indexes['Type'] > -1 ? $row[ $indexes['Type']] : '';
		if( $type == 'Article' ) {
			if( $indexes['Name'] > -1 ) {
				$row[$indexes['Name']] .= ' (article)';
			}
		}
	}
}
```

Custom properties are set up on the “Metadata” Maintenance page. (See the Admin Guide for configuration details). These properties are internally prefixed with a “C_”. For example, a custom property named “HELLO” is internally called “C_HELLO”.

Spelling [since 7.4]

The Multi-Channel Text Editor of Content Station 7.4 includes support for spelling checking. This feature runs through new web services exposed by Enterprise Server 7.4. This feature is about having a consistent spelling checking throughout the whole workflow. For example, once an article is spell-checked by an editor, he/she puts the article in the next status, indicating the article is ready for publishing. When at that stage the article shows five errors, no matter on which machine or on which editor the article gets opened, it should still show these (and only these) same five errors.

Customers are free to choose their favorite spelling engine. Most commonly used engines are integrated and provided as demo server plug-ins. Integrators are able to build their own integration with other engines. For that, a new business connector interface is added to integrate spelling engines:

`Enterprise/server/interfaces/plugins/connectors/Spelling_EnterpriseConnector.class.php`

The supported spelling languages depend on the availability of the dictionaries for the chosen spelling engine. Enterprise should not stand in the way at this point and let system admins freely configure any (or many) of those languages.

Since the new configuration is much more advanced, the article's text might be split into words differently (as a pre-processing step for spelling). As a result, there could be unneeded spelling errors for those two editors. This is a known limitation.

Note that also InCopy and InDesign do not use the Spelling and Suggestions features as offered by Enterprise Server. Instead, they use their own spelling features as built-in by Adobe. This could lead to spelling differences between the editors.

Configurations applied to the ENTERPRISE_SPELLING option are returned through the LogOnResponse to Content Station. See the Admin Guide for installation and configuration of spelling integrations.

New services are added to the workflow WSDL: CheckSpelling, GetSuggestions and CheckSpellingAndSuggest.

Preview & Copy-Fit [since 7.4]

Content Station ships an editor that supports multi-channel editing of articles in WCML format. Enterprise Server offers web services Content Station can talk to. A workspace folder can be created at server side to temporary store the article and the layouts it is placed into. Having that in place, previews, PDFs and copy-fit information can be requested. At the back-end, the server integrates Smart Connection for InDesign Server to do the actual previewing, PDF generation and copy-fit calculation for the article being edit.

The following features are in place:

- Write-To-Fit, Preview and PDF of placed articles.
- Write-To-Fit, Preview and PDF of articles derived from article templates*.
  * The template must be created from a layout to inherit geometrical information.
- Save article at workspace.
  - Does not create a version in the database.
  - Also needed for Auto Save feature.
- New article vs existing article.
  - New articles are not stored in database yet, but can be edited (but not previewed).
- Creating article from template.
- Word/Char/Para counts (done client side only).

Preview optimizations [since 7.4]:

- Only preview pages on which article is placed.
- Only retrieve the current page.

Preview optimizations [since 9.5]:

- Save and keep layout in workspace. Reuse layout for succeeding requests.
- Save changed article components separately in workspace for fast loading by SC.
- Generate previews and copy-fit for changed text components only.
- SC for IDS no longer retrieves the brand setup through logon response.
- CS editor updates internal versions (GUIDs) of changed stories only.
- CS editor and File Transfer Server use the Deflate compression technique to compress WCML articles while transferring them to or from remote locations.

The following services are added to the workflow interface (SCEnterprise.wsdl):

```
CreateArticleWorkspace
      Ticket
      ID (nil for new articles)
      Format ('application/incopyicml', used for file extension, and for text conversion?)
      Content (embedded XML, taken from article template)
CreateArticleWorkspaceResponse
      WorkspaceId
```

Since Enterprise 9.5, when the `ID` parameter is provided, the `Content` parameter can be set to nil. In that case, the latest version of the article is retrieved by Enterprise Server and put into the workspace folder. This saves time for remote workers and/or large articles.

```
ListArticleWorkspaces (used for recovery only)
      Ticket
ListArticleWorkspacesResponse
      Workspaces
            WorkspaceId
            WorkspaceId
```

```
GetArticleFromWorkspace (used for recovery only)
      Ticket
      WorkspaceId
GetArticleFromWorkspaceResponse
      ID (nil for new articles)
      Format
      Content (embedded XML)
```

```
SaveArticleInWorkspace (used for auto-save only)
      Ticket
      WorkspaceId
      ID (nil for new articles)
      Format ('application/incopyicml', used for file extension, and for text conversion?)
      Elements (dirty frames only, nil when Content is used instead)
            Element
                  Content (embedded XML, of one frame only)
      Content (embedded XML, used after adding/removing text components)
SaveArticleInWorkspaceResponse
      [empty]
```

```
PreviewArticleAtWorkspace (does implicit save at workspace)
      Ticket
      WorkspaceId
      ID (nil for new articles)
      Format ('application/incopyicml', used for file extension, and for text conversion?)
      Elements (dirty frames only, nil when Content is used instead)
            Element
                  Content (embedded XML, of one frame only)
      Content (embedded XML, used after adding/removing text components)
      Action (Compose/Preview/PDF)
      LayoutId
      EditionId
PreviewArticleAtWorkspaceResponse (Preview/PDF do Compose implicitly)
      Placements
            Placement (copyfit info)
      Elements
            Element (word/char/para/line counts)
      Pages (nil on Compose)
            Page
                  Files
                        File
                              FileUrl (taken from v8)
                              L> http://.../previewindex.php?ticket=12&
                              workspaceid=34&action=preview&pageseq=1&
                              layoutid=56&editionid=78&articleid=90
      LayoutVersion (as used, can be auto updated server side)
```

```
DeleteArticleWorkspace
      WorkspaceId
DeleteArticleWorkspaceResponse
      [empty]
```

Spread Preview [since 7.6]

In the Multi-Channel Text Editor of Content Station, the user can toggle between single page and Spread Preview mode. In single page preview mode, the preview pane of the editor displays all pages that contain (a part of) the article only. In Spread Preview mode, when an article is placed on one page of a spread (but not on the sibling page) the sibling page is included too, just to 'complete' the spread view.

Related to this feature, when an article text frame runs over two pages of a spread, it should be visualized on -both- pages with a gray box in the preview pane. (Before, only one of the two pages showed the gray box.)

In this chapter, the following terms are used:

Term	Meaning
spread	Left and right page together (as shown in InDesign).
sibling page	The other page of the spread. Talking about the left page, the right page is the sibling page, or vice versa.
placement tile	Part of a placement. When placement (text frame) runs over two pages of a spread, it gets 'divided' into two tiles; One tile that fits on the left page and one tile that fits on the right page. Both tiles glued together has the same shape/dimensions as the placement itself.

When an article text frame (from now called 'placement') is placed on one page of the spread, but also partly covers the sibling page, the placement is enriched with two 'tiles'. As long as a placement fits on its page, there are no tiles. When one placement covers two pages, there are two tiles, each with coordinates and dimensions relative to the corresponding page. For a tile, Enterprise keeps track of placement db id, page sequence nr, left, top, width and height.

A PlacementTile is new workflow data type added to the SCEnterprise.wsdl. The Placement element has a new property named Tiles, which is a list of PlacementTile elements.

When a layout is stored (through CreateObjects / SaveObjects services), the Smart Connection plug-ins for InDesign send tile info along with the page placements. This happens only when necessarily; when the placement fits on the page, no tiles are provided. Enterprise Server stores the tiles in the database in a new table, named smart_placementtiles.

When the layout and article are stored at the server, the Multi-Channel Text Editor of Content Station can request a preview through the PreviewArticleAtWorkspace service. This service has a new parameter named PreviewType, which can be either 'page' or 'spread'. In single page preview mode, Content Station sends 'page' and in Spread Preview mode, it sends 'spread'.

When it comes to preview generation, Enterprise Server calls InDesign Server through SOAP, and feeds its JavaScript module named IDPreview.js (which can be found in the server/apps folder). That module talks through scripting API to InDesign Server and Smart Connection plug-ins. This integration is responsible for returning previews of pages

containing article content. Results are written in the composedata.xml file in the user's workspace folder (server side) along with the generated preview files. So far, nothing new.

If an article placement is placed on a spread (for example: one text frame on pages 2 and 3) and the new PreviewType parameter is set to 'spread', IDPreview.js returns previews of -both- pages (one that contains the article, and one that does not).

When a placement has tiles, Smart Connection plug-ins provide Tiles information through the frameData JSON. This info is read by IDPreview.js and is written to the composedata.xml. Therefore, the XML structure is expanded with a <tiles> element that may occur inside a <textframe> element. Inside, a list of <tile> elements provides info about the dimension of each placement tile.

Enterprise Server (more precisely: the previewArticleAtWorkspace function of BizWebEditWorkspace class) parses the composedata.xml file and composes the Tiles structure returned through the PreviewArticleAtWorkspace service response.

Content Station now displays the Spread Preview. The logics behind placing pages left or right are implemented in Content Station. This depends on the reading order. Therefore, the ReversedRead property is introduced in the PublicationInfo and IssueInfo elements in the workflow WSDL (SCEnterprise.wsdl). This reading order info is returned through the LogOn response.

For the Publication Overview in Content Station, the placement tiles are also important. Without tiles, Content Station won't know if a text frame of an article runs over the other page of the spread (or not). Therefore, Enterprise Server returns extra PlacementInfo elements through the GetPages service (as called by Content Station for the Publication Overview). For each placement, but now also for each placement tile, a PlacementInfo exists.

When a layout is sent to the Trash Can, the tiles remain in the database (in the smart_placementtiles table) so that the tiles are available again when the layout is restored. When a layout is removed permanently (purged), the tiles are removed from the database.

Annotations [since 8.0]

This chapter describes how the workflow interface/web services definition (SCEnterprise.wsdl) is extended to support the Annotations feature.

Unlike other workflow services, the SendMessages service ‘does it all’. Where other services are very explicit (create, update, delete), this service looks at the given Message elements and determines what operation to perform (create, update, delete) one by one. When a message exists, it does an update, else a create. When the UserID and ObjectID (of a Message element) are set to zero, a delete operation is performed instead. In fact, the list of passed in Message elements can be seen as a list of commands. This have been the case from the early days of Enterprise, and remains unchanged. Nevertheless, this principle is good to understand before reading this chapter. Since 8.0 the delete operations are made more explicit to distinguish between users deleting messages, and messages marked as ‘read’ (which is explained in the next chapters).

It is by design that placing Sticky Notes on layouts in InDesign are not directly reflected to other users (such as in the Publication Overview in Content Station) until the layout is saved. The other way around, when messaging (n-casting) is enabled, and users edit Sticky Notes (or replies) in the Publication Overview, those changes -are- reflected directly to the layout opened in InDesign. InDesign is responsible for handing conflicts related to Sticky Notes and replies.

Configuration

Access rights

On the Profile Maintenance page, the “Edit Sticky Notes” option in the Workflow section has been renamed in Enterprise 8.0 to “Create and Reply Sticky Notes” and moved to a new section named “Annotations”. Two more access rights are added to that section as well: “View Sticky Notes” and “Delete Sticky Notes”.

When the “Edit Sticky Notes” was enabled (or disabled) for older Enterprise Server versions, once migrated to Enterprise 8.0, “Create and Reply Sticky Notes” and “Delete Sticky Notes” are automatically set accordingly. The “View Sticky Notes” is always enabled.

For newly created profiles, all three access right options under the “Annotations” section are enabled by default.

Important: Access rights are checked client side, not server side. Whether certain operations are allowed or not is made clear to end-users immediately instead of for example after an expensive save operation of a layout. In other terms, the web services allow all message operations, as requested by clients. As a result, users working with Smart Connection 7.x or Content Station 7.x are allowed to delete message, while Smart Connection 8.0 / Content Station 8.0 could disallow (depending on the configured access rights). This behavior is by design.

Unread messages in Search results

On the Dialog Setup admin page a new column is added named Unread Messages (UnreadMessageCount). This can be added to any of the Query Result Columns. Once configured, it shows the number of Sticky Notes (including replies) that are placed on a layout which aren’t read by anyone yet. (This also includes system messages sent to objects.)

Similar to “PlacedOn” and “Issues”, the new property “UnreadMessageCount” is a calculated property. This means that its value can only be determined after the object is retrieved from the database. The consequence is that you cannot search/sort on this

property without use of Solr. In case of Solr, searching and sorting is possible as the value of “UnreadMessageCount” is indexed by Solr.

The user creating Sticky Notes or replies does see them counted as unread messages in the search results too, like anyone else. This is expected behavior: it should be seen as a mark for new messages added to the workflow that needs someones attention when working on those objects.

For the Inbox search, users can simply add the Unread Messages column through the GUI of Smart Connection or Content Station: by right-clicking the column header and selecting the column.

Messages in email

See the Admin Guide for how to set up and configure email for Enterprise. The enterprise/config/templates folder contains email HTML files. In any of the email template files, you can add the %Messages% tag, which gets replaced by the messages placed on the layout. For example, when a layout is routed to a user, that user will receive an email where the Sticky Notes (and its replies) are already listed in the email.

Message restructuring (since 8.0)

Since 8.0 a new type is introduced, named ‘MessageList’:

```
<complexType name="MessageList">
    <all>
        <element name="Messages" type="tns:ArrayOfMessage"
                 nillable="true" />
        <element name="ReadMessageIDs" type="tns:ArrayOfString"
                 nillable="true" />
        <element name="DeleteMessageIDs" type="tns:ArrayOfString"
                 nillable="true" />
    </all>
</complexType>
```

Basically, all Messages- and ReadMessageIDs properties are replaced with MessageList properties. This affects LogOnResponse, LogOff request, UnlockObjects request, Object, CreateObjects request, SaveObjects request, ObjectPageInfo request, SendMessages request,

And, the meaning of ReadMessageIDs has been changed; It no longer means to delete the messages. Instead, there is a new property introduced, named DeleteMessageIDs that does actual deletions. When messages are read by a user, the ReadMessageIDs is used. This is for v8 clients talking to a v8 server. When v7 clients are talking, the v8 will detect and change the request on-the-fly in its service layer.

Server Plug-ins: Migration to 8.0 (or newer)

Because the service layer in Enterprise Server takes care of restructuring messages on-the-fly, Server Plug-ins (as well as the core server) do not have to deal with old structures: there is only the new structure to deal with. Nevertheless, this means that old plug-ins needs to be migrated when they intercept the messages.

User messages

Unlike other message types, messages sent to users are delete once read.

SC/CS 7.x (or older)

A user logs in to the system:

```
<LogOnResponse>
      ...
```

```
      <Messages>
            <Message>
                  <ObjectID xsi:nil="true" />
                  <UserID>woodwing</UserID>
                  <MessageID>8547...DD310</MessageID>
                  <MessageType>user</MessageType>
                  ...
            </Message>
      </Messages>
      ...
</LogOnResponse>
```

A user has read the message and logs off:

```
<LogOff>
      ...
      <ReadMessageIDs>
            <String>8547...DD310</String>
      </ReadMessageIDs>
      ...
</LogOff>
```

Note that the ReadMessageIDs indicate that the messages needs to be deleted. Since 8.0 this request is made more explicit, as explained in the next paragraph.

SC/CS 8.0 (or newer)

A user logs in to the system:

```
<LogOnResponse>
      ...
      <MessageList>
            ...
            <Messages>
                  <Message>
                        <ObjectID xsi:nil="true" />
                        <UserID>woodwing</UserID>
                        <MessageID>8547...DD310</MessageID>
                        <MessageType>user</MessageType>
                        ...
                  </Message>
            </Messages>
            ...
      </MessageList>
      ...
</LogOnResponse>
```

A user has read the message and logs off:

```
<LogOff>
      ...
      <MessageList>
            ...
            <DeleteMessageIDs>
                  <String>8547...DD310</String>
            </DeleteMessageIDs>
            ...
      </MessageList>
      ...
</LogOff>
```

Note that for the LogOnResponse, the Server detects the client major version in the ClientAppVersion parameter of the LogOn request. When this version is 8 (or newer), the MessageList structure is used. When the version is 7 (or older), the obsoleted Messages structure is used instead.

Sticky Notes (since 4.2)

Since Enterprise 4.2 a new message type is introduced, named 'sticky':

```
<simpleType name="MessageType">
    <restriction base="string">
        <enumeration value="system" />
        <enumeration value="client" />
        <enumeration value="user" />
        <enumeration value="sticky" />
        <enumeration value="reply" />
    </restriction>
</simpleType>
```

And, the Message is extended with a property named 'StickyInfo':

```
<complexType name="Message">
    <all>
        <element name="ObjectID" nillable="true" type="xsd:string" />
        <element name="UserID" nillable="true" type="xsd:string" />
        <element name="MessageID" nillable="true" type="xsd:string" />
        ...
        <element name="StickyInfo" nillable="true" type="tns:StickyInfo" />
        ...
    </all>
</complexType>
```

Also, the StickyInfo type definition is introduced:

```
<complexType name="StickyInfo">
    <all>
        <element name="AnchorX" type="xsd:double" />
        <element name="AnchorY" type="xsd:double" />
        <element name="Left" type="xsd:double" />
        <element name="Top" type="xsd:double" />
        <element name="Width" type="xsd:double" />
        <element name="Height" type="xsd:double" />
        <element name="Page" type="xsd:unsignedInt"
                 nillable="true" minOccurs="0" maxOccurs="1" />
        <element name="Version" type="xsd:string" />
        <element name="Color" type="tns:Color" />
        <element name="PageSequence" type="xsd:unsignedInt" />
    </all>
</complexType>
```

Smart Connection: Place new Sticky Note on a page

Smart Connection 7.x (or older)

In InDesign (or InCopy), the user places a Sticky Note on a page and saves the layout:

```
<SaveObjects>
      ...
      <Messages>
            <Message>
                  <ObjectID>27</ObjectID>
                  <UserID xsi:nil="true" />
                  <MessageID>8547...DD310</MessageID>
                  <MessageType>sticky</MessageType>
                  <MessageTypeDetail></MessageTypeDetail>
```

```
                  <Message>The text does not fit here.</Message>
                  <TimeStamp>2012-03-13T17:40:14</TimeStamp>
                  <Expiration xsi:nil="true" />
                  <MessageLevel xsi:nil="true" />
                  <FromUser>woodwing</FromUser>
                  <StickyInfo>
                        <AnchorX>43.022093</AnchorX>
                        <AnchorY>177.920463</AnchorY>
                        <Left>61.046512</Left>
                        <Top>155.247207</Top>
                        <Width>181.000000</Width>
                        <Height>78.000000</Height>
                        <Page>1</Page>
                        <Version>0</Version>
                        <Color>FF0000</Color>
                        <PageSequence>1</PageSequence>
                  </StickyInfo>
            </Message>
      </Messages>
      ...
</SaveObjects>
```

Smart Connection 8.0 (or newer)

In InDesign (or InCopy), the user places a Sticky Note on a page and saves the layout:

```
<SaveObjects>
      ...
      <Objects>
      <Object>
            ...
            <MessageList>
                  ...
                  <Messages>
                  <Message>
                        <ObjectID>27</ObjectID>
                        <UserID xsi:nil="true" />
                        <MessageID>8547...DD310</MessageID>
                        <MessageType>sticky</MessageType>
                        <MessageTypeDetail></MessageTypeDetail>
                        <Message>The text does not fit here.</Message>
                        <TimeStamp>2012-03-13T17:40:14</TimeStamp>
                        <Expiration xsi:nil="true" />
                        <MessageLevel xsi:nil="true" />
                        <FromUser>woodwing</FromUser>
                        <StickyInfo>
                              <AnchorX>43.022093</AnchorX>
                              <AnchorY>177.920463</AnchorY>
                              <Left>61.046512</Left>
                              <Top>155.247207</Top>
                              <Width>181.000000</Width>
                              <Height>78.000000</Height>
                              <Page>1</Page>
                              <Version>0</Version>
                              <Color>FF0000</Color>
                              <PageSequence>1</PageSequence>
                        </StickyInfo>
                        <ThreadMessageID xsi:nil="true" />
                        <ReplyToMessageID xsi:nil="true" />
                        <MessageStatus>None</MessageStatus>
                  </Message>
                  </Messages>
```

```
                ...
            </MessageList>
            ...
        </Object>
        </Objects>
        ...
</SaveObjects>
```

Same for CreateObjects request.

Content Station: Place a new Sticky Note on a page

Content Station 7.x (or older)

On the Publication Overview, the user places a Sticky Note on a page:

```
<SendMessages>
        ...
        <Messages>
            <Message>
                <ObjectID>10</ObjectID>
                <UserID xsi:nil="true" />
                <MessageID></MessageID>
                <MessageType>sticky</MessageType>
                <MessageTypeDetail></MessageTypeDetail>
                <Message>The text does not fit here.</Message>
                <TimeStamp>2012-03-16T12:48:46</TimeStamp>
                <Expiration xsi:nil="true" />
                <MessageLevel xsi:nil="true" />
                <FromUser>woodwing</FromUser>
                <StickyInfo>
                    <AnchorX>0</AnchorX>
                    <AnchorY>0</AnchorY>
                    <Left>163</Left>
                    <Top>216</Top>
                    <Width>300</Width>
                    <Height>100</Height>
                    <Page>0</Page>
                    <Version>0</Version>
                    <Color>ff0000</Color>
                    <PageSequence>1</PageSequence>
                </StickyInfo>
            </Message>
        </Messages>
</SendMessages>
```

Content Station 8.0 (or newer)

On the Publication Overview, the user places a Sticky Note on a page:

```
<SendMessages>
        ...
        <MessageList>
            ...
            <Messages>
                <Message>
                    <ObjectID>10</ObjectID>
                    <UserID xsi:nil="true" />
                    <MessageID></MessageID>
                    <MessageType>sticky</MessageType>
                    <MessageTypeDetail></MessageTypeDetail>
                    <Message>The text does not fit here.</Message>
                    <TimeStamp>2012-03-16T12:48:46</TimeStamp>
                    <Expiration xsi:nil="true" />
```

```
                        <MessageLevel xsi:nil="true" />
                        <FromUser>woodwing</FromUser>
                        <StickyInfo>
                              <AnchorX>0</AnchorX>
                              <AnchorY>0</AnchorY>
                              <Left>163</Left>
                              <Top>216</Top>
                              <Width>300</Width>
                                    <Height>100</Height>
                              <Page>0</Page>
                              <Version>0</Version>
                              <Color>ff0000</Color>
                              <PageSequence>1</PageSequence>
                        </StickyInfo>
                        <ThreadMessageID xsi:nil="true" />
                        <ReplyToMessageID xsi:nil="true" />
                        <MessageStatus>None</MessageStatus>
                  </Message>
            </Messages>
            ...
      </MessageList>
      ...
</SendMessages>
```

Reply to a message (since 8.0)

Since Enterprise 8.0, a new message type is introduced, named ‘reply’:

```
<simpleType name="MessageType">
    <restriction base="string">
        <enumeration value="system" />
        <enumeration value="client" />
        <enumeration value="user" />
        <enumeration value="sticky" />
        <enumeration value="reply" />
    </restriction>
</simpleType>
```

And, the Message is extended with two properties, named ‘ThreadMessageID’ and ‘ReplyToMessageID’:

```
<complexType name="Message">
    <all>
        <element name="ObjectID" nillable="true" type="xsd:string" />
        <element name="UserID" nillable="true" type="xsd:string" />
        <element name="MessageID" nillable="true" type="xsd:string" />
        ...
        <element name="ThreadMessageID" type="xsd:string"
                  nillable="true" minOccurs="0" maxOccurs="1" />
        <element name="ReplyToMessageID" type="xsd:string"
                  nillable="true" minOccurs="0" maxOccurs="1" />
        ...
    </all>
</complexType>
```

Note that ThreadMessageID and ReplyToMessageID are made optional to support 7.6 clients talking to a 8.0 server. In future versions this might become mandatory (but remains nillable).

Although hierarchy in message replies is not required for 8.0, the data model is prepared for this future feature. For the GUI, a flat list can be shown, and in future versions, this can be made more advanced by showing a hierarchy.

Once a Sticky Note is placed, users can reply. The Sticky Note can be seen as the initiator of a ‘thread’. The sticky message has StickyInfo, but the replies don’t (set to nil). The sticky message has ReplyToMessageID set to zero, but replies have set it to the message ID of the previous reply or sticky. The sticky message and the replies have the ThreadID set to the sticky message ID.

Smart Connection: Reply to a Sticky Note on a page

In InDesign (or InCopy), the user replies at a Sticky Note on a page and saves the layout:

```
<SaveObjects>
      ...
      <Objects>
      <Object>
            <MessageList>
                  ...
                  <Messages>
                  <Message>
                        <ObjectID>27</ObjectID>
                        <UserID xsi:nil="true" />
                        <MessageID>6EFE668F-...-4823E18B9EBF</MessageID>
                        <MessageType>reply</MessageType>
                        <MessageTypeDetail></MessageTypeDetail>
                        <Message>I have made the text fit now.</Message>
                        <TimeStamp>2012-03-14T12:45:00</TimeStamp>
                        <Expiration xsi:nil="true" />
                        <MessageLevel xsi:nil="true" />
                        <FromUser>woodwing</FromUser>
                        <StickyInfo xsi:nil="true" />
                        <ThreadMessageID>8547...DD310</ThreadMessageID>
                        <ReplyToMessageID>8547...DD310</ReplyToMessageID>
                        <MessageStatus>None</MessageStatus>
                  </Message>
                  </Messages>
                  ...
            </MessageList>
      </Object>
      </Objects>
      ...
</SaveObjects>
```

Note that messages are hierarchical. The ThreadMessageID points to the initial Sticky Note. The ReplyToMessageID points to the message the user reacts on. For the first reply, these two ids are the same. For following replies the ThreadMessageID remains the same, but ReplyToMessageID could point to the Sticky Note or any of the replies. The request is composed in human reading order to simplify parsing it server side (by core or plug-ins), and so it may assumed that the two ids always refer back to messages listed ‘earlier’.

Content Station: Reply to a Sticky Note on a page

In the Publication Overview, the user replies to a Sticky Note on a page:

```
<SendMessages>
      ...
      <MessageList>
            ...
            <Messages>
                  ...
```

```
                  <Message>
                        <ObjectID xsi:nil="true" />
                        <UserID xsi:nil="true" />
                        <MessageID>6EFE668F-41ED-A226-4823E18B9EBF</MessageID>
                        <MessageType>reply</MessageType>
                        <MessageTypeDetail></MessageTypeDetail>
                        <Message>I have made the text fit now.</Message>
                        <TimeStamp>2012-03-14T12:45:00</TimeStamp>
                        <Expiration xsi:nil="true" />
                        <MessageLevel xsi:nil="true" />
                        <FromUser>woodwing</FromUser>
                        <StickyInfo xsi:nil="true" />
                        <ThreadMessageID>8547...DD310</ThreadMessageID>
                        <ReplyToMessageID>8547...DD310</ReplyToMessageID>
                        <MessageStatus>None</MessageStatus>
                  </Message>
                  ...
            </Messages>
            ...
      </MessageList>
      ...
</SendMessages>
```

Modify a Sticky Note (since 4.2) or a reply (since 8.0)

In InDesign (or InCopy) and/or in the Publication Overview of Content Station, when a user changes the text on their own replies the same request is sent as during the creation (except that the data is updated). The same happens when a user moves the Sticky Note or its anchor.

Note that this is only allowed when there are no replies to the message being modified. In other terms, if another user has replied in the meantime, the message can no longer be modified. The server throws an error on such an attempt but only for SendMessages service (called by Content Station).

The last update / operation for a message ‘wins’. So when user A updates all messages on a page, just after user B has updated his/her own message, the update of user A overwrites the update of user B. This was for 7.x (or older) a bigger problem than for 8.0 (or newer) since users are only allowed to update their own messages. Nevertheless, Smart Connection and Content Station therefore never send messages of other users, other than the messages of the current user. And, they only send messages that are really updated. Untouched messages are never sent to the server. Also, when sending messages, their properties are set to nil as much as possible (but, obviously respecting the WSDL) to avoid undoing earlier made changes.

Delete a Sticky Note (since 4.2) or a reply (since 8.0)

Smart Connection 7.x (or older)

In InDesign (or InCopy), the user deletes a Sticky Note or reply:

```
<SaveObjects>
      ...
      <ReadMessageIDs>
            <String/>8547...DD310</String>
      </ReadMessageIDs>
      ...
</SaveObjects>
```

Note that the ReadMessageIDs indicate that the messages needs to be deleted. Since 8.0 this request is made more explicit as written in next paragraph.

Smart Connection 8.0 (or newer)

In InDesign (or InCopy), the user deletes a Sticky Note or reply:

```
<SaveObjects>
	...
	<Objects>
		<Object>
			...
			<MessageList>
				...
				<DeleteMessageIDs>
					<String/>8547...DD310</String>
				</DeleteMessageIDs>
				...
			</MessageList>
			...
		</Object>
	</Objects>
	...
</SaveObjects>
```

Content Station 7.x (or older)

In the Publication Overview, the user deletes a Sticky Note or reply:

```
<SendMessages>
	...
	<Messages>
		<Message>
			<ObjectID>0</ObjectID>
			<UserID>0</UserID>
			<MessageID>8547...DD310</MessageID>
			<MessageType>sticky</MessageType>
			...
		</Message>
	</Messages>
</SendMessages>
```

Note that when both ObjectID and UserID are set to zero, the message gets deleted (from the database). Since 8.0 this request is made more explicit as written in next paragraph.

Content Station 8.0 (or newer)

In the Publication Overview, the user deletes a Sticky Note or reply:

```
<SendMessages>
	...
	<MessageList>
		...
		<DeleteMessageIDs>
			<String/>8547...DD310</String>
		</DeleteMessageIDs>
		...
	</MessageList>
</SendMessages>
```

Server Plug-ins: Migration to 8.0 (or newer)

Custom server plug-ins designed for Enterprise 7.x (or older) that implement the SendMessages connector to detect message deletions need to be migrated to the 8.0 method. The service layer transforms old service requests into new service requests, so the connectors will be called in the 8.0 way, regardless of how it gets called by clients.

Exceptions

Note that deleting replies is only allowed when there are no other replies to it. In other terms, if another user has replied in the meantime, the message can no longer be deleted. The server throws an error on such attempt for SendMessages (Content Station), but not for SaveObjects (Smart Connection).

Mark as read (since 8.0)

Smart Connection: Mark a Sticky Note (or a reply) as read

In InDesign (or InCopy), the user opens a layout that contains a Sticky Note:

```
<GetObjectsResponse>
      ...
      <Objects>
      <Object>
            ...
            <MessageList>
                  ...
                  <Messages>
                  <Message>
                        <ObjectID>27</ObjectID>
                        <UserID xsi:nil="true" />
                        <MessageID>8547...DD310</MessageID>
                        <MessageType>sticky</MessageType>
                        ...
                  </Message>
                  </Messages>
                  <ReadMessageIDs></ReadMessageIDs>
                  ...
            </MessageList>
            ...
      </Object>
      </Objects>
      ...
</GetObjectsResponse>
```

Note that for the GetObjectsResponse, the Server detects the client major version in the ClientAppVersion parameter of the LogOn request. When this version is 8 (or newer), the MessageList structure is used. When the version is 7 (or older), the obsoleted Messages structure is used instead.

In InDesign (or InCopy), the user marks a reply (to a Sticky Note on a page) as ‘read’ and saves the layout:

```
<SaveObjects>
      ...
      <Objects>
            <Object>
                  ...
                  <MessageList>
                        ...
                        <ReadMessageIDs>
                              <String/>8547...DD310</String>
                        </ReadMessageIDs>
                        ...
                  </MessageList>
                  ...
            </Object>
      </Objects>
      ...
```

```
</SaveObjects>
```

Content Station: Mark Sticky Note (or a reply) as read

In the Publication Overview, the user opens a preview that contains Sticky Notes and marks a Sticky Note as ‘read’:

```
<SendMessages>
      ...
      <MessageList>
            ...
            <ReadMessageIDs>
                  <String/>8547...DD310</String>
            </ReadMessageIDs>
            ...
      </MessageList>
</SendMessages>
```

Message workflow (since 8.0)

To implement a basic workflow for messages, the Message is extended with MessageStatus:

```
<complexType name="Message">
    <all>
        <element name="ObjectID" nillable="true" type="xsd:string" />
        <element name="UserID" nillable="true" type="xsd:string" />
        <element name="MessageID" nillable="true" type="xsd:string" />
        ...
        <element name="MessageStatus" type="tns:MessageStatus"
               nillable="true" minOccurs="0" maxOccurs="1"/>
        ...
    </all>
</complexType>
```

Note that the MessageStatus is made optional to support 7.6 clients talking to a 8.0 server.

The message can follow these statuses:

```
<simpleType name="MessageStatus">
    <restriction base="string">
        <enumeration value="None" />
        <enumeration value="Accepted" />
        <enumeration value="Cancelled" />
        <enumeration value="Completed" />
        <enumeration value="Rejected" />
    </restriction>
</simpleType>
```

Note that message statuses are pre-defined and cannot be customized.

History trail for Sticky Notes and replies (since 8.0)

To track for which object version a message was created, the Message is extended with ObjectVersion:

```
<complexType name="Message">
    <all>
        <element name="ObjectID" nillable="true" type="xsd:string" />
        <element name="UserID" nillable="true" type="xsd:string" />
        <element name="MessageID" nillable="true" type="xsd:string" />
        ...
        <element name="ObjectVersion" type="xsd:string"
               nillable="true" minOccurs="0" maxOccurs="1"/>
```

```
            ...
        </all>
</complexType>
```

To allow 7.6 clients talking to a 8.0 server, the ObjectVersion property is made optional. 7.6 clients send no ObjectVersion, but 8.0 client send an ObjectVersion set to nil. The reason is that for a future version this property will be made mandatory (but remains nillable).

When a Message is sent along with the CreateObjects or SaveObjects requests (fired by Smart Connection), the ObjectVersion is nil to let the Server determine and fill-in the created object version just before storing the message in the database. When a Message sent along the SendMessages request (fired by Publication Overview), the ObjectVersion is pre-filled by Content Station. The reason is that the user intension is to put the Sticky Note on the particular version he/she is looking at. So the version in known ahead of the operation.

Implicit deletion of Sticky Notes and replies

Some cases exist whereby Sticky Notes and replies are deleted implicitly.

On deletion of an object

When an object is sent to the Trash Can, nothing happens to the Sticky Notes nor their replies. But when the object gets removed from Trash Can (purged), they all get removed too.

On deletion of a user

This hardly happens, but the system administrator can remove a user (although it is a preferred method to de-activate users instead). Sticky Notes and their replies will -not- be removed from the layout pages implicitly. But, messages sent to the user that is removed, are removed implicitly. Basically, messages can be sent from a user who does no longer exist in the system. But for messages sent to a user, that user must exist. For all the above, the same counts for user groups.

On deletion of a Sticky Note

Deleting a Sticky Note implicitly deletes all its replies.

Planning integration

3rd-party planning systems (such as Journal Designer) connect through the planning interface (SmartEditorialPlan.wsdl), just like the clientplan.php tool (in the wwtest folder). Doing so, the Server sends messages to the layouts and adverts that are created or modified. Those messages are shown in a dialog in InDesign when opening the layout (for example with placed adverts). Those messages have the MessageType set to ‘system’ and MessageTypeDetail set to the operation, such as ‘ModifyLayout’.

Exceptions

The SendMessages service (as called by Content Station) can throw various types of errors. There are all kinds of exceptional situations that can occur in production:

- User A could purge a layout while user B is replying to a Sticky Note on one of the pages.
- User A modifies his/her last reply while user B is replying to it.
- ...

The SaveObjects service (as called by Smart Connection) does -not- throw errors for message related problems. This is to avoid failures on expensive save operations (especially for layouts). Nevertheless, erratic messages are skipped to make sure all messages are at least handled.

Access rights

The access rights listed in the table below are related to annotations: The actual values configured on the admin pages are returned through LogOn web service and are checked client side.

access right	internal name	introduced since	default value
Create and Reply Sticky Notes*	EditStickyNotes	4.2	enabled
View Sticky Notes	ViewNotes	8.0	enabled
Delete Sticky Notes	DeleteNotes	8.0	enabled

* Since 8.0, the access right “Edit Sticky Notes” is renamed to “Create and Reply Sticky Notes”, as shown on the Profiles Maintenance admin page. However, the internal name (communicated from server to clients) still remains the same, which is “EditStickyNotes”. This is done for backwards compatibility reasons (version 7 client talking to version 8 server).

Through the LogOnResponse, only differences compared to the default values are sent to clients. In other terms, when admin user did not change the default, nothing is sent. Since all three access rights options enabled by default, when the admin user disables one of these options, only those are returned.

For example, when all three options are disabled by the admin user, the following is returned:

```
<LogOnResponse>
      ...
      <FeatureProfiles>
            ...
            <FeatureProfile>
                <Name>My Profile</Name>
                <Features>
                    ...
                    <AppFeature>
                        <Name>ViewNotes</Name>
                        <Value>No</Value>
                    </AppFeature>
                    <AppFeature>
                        <Name>EditStickyNotes</Name>
                        <Value>No</Value>
                    </AppFeature>
                    <AppFeature>
                        <Name>DeleteNotes</Name>
                        <Value>No</Value>
                    </AppFeature>
                    ...
                </Features>
            </FeatureProfile>
            ...
      </FeatureProfiles>
      ...
</LogOnResponse>
```

n-cast messaging

With n-casting (broadcasting / multicasting) enabled, changes to messages are directly reflected between the InDesign layout and the Publication Overview. This is convenient, but also avoids data loss. It is expected that when user A saves a layout, after user B has

made changes to Sticky Notes in the Publication Overview, that the changes of user A overwrite the changes of user B. Chances that such a thing happens are slim when n-casting is enabled. Obviously, for remote workers this cannot be enabled, and so chances are relatively bigger.

The Messaging Specification gives full detail of n-casted messages and their fields. Since 8.0, the following fields are added to the SendMessage event: ThreadMessageID, ReplyToMessageID, MessageStatus, ObjectVersion and IsRead (boolean).

Trash Can & Clean Up [since 8.0]

Enterprise 7 allows system admin users to manage deleted objects through the admin pages. Enterprise 8 exposes that functionality to brand users and end users as well. Users can restore objects or delete objects permanently from the Trash Can.

Objects can be owned by Brands, assigned to Issues, classified by Categories, etc. Those entities cannot be removed as long they are occupied by objects, regardless of wether they in the workflow or in the Trash Can. Clean Up features help to move objects elsewhere or delete them.

Configuration and migration details of the Trash Can and Auto Purge feature can be found in the Admin Guide.

Object properties

There are two new object properties introduced to track deleted objects:

- ***Deleted On (Deleted):*** Date-time stamp when the object has been deleted. Used to filter objects in the Trash Can. This is similar to the Created On (Creator) and Modified On (Modifier) properties.
- ***Deleted By (Deleter):*** User who deleted the object. Used to determine which objects reside in the user's individual Trash Can. This is similar to the Created By (Creator) and Modified By (Modifier) properties.

When an object is deleted, both properties will be automatically filled in by the server. When an object is restored, the properties remain untouched, so they provide information on the last delete operation.

Workflow dialogs

Like the Delete operation (and unlike other workflow operations) there is no workflow dialog raised for Restore- and Delete Permanent operations. So there is no dialog configuration for it either.

The two new properties (Deleter and Deleted) are determined by the system. Therefore, they are always read-only when shown at workflow dialogs.

Dialog Setup / Query Setup

The following changes are made to the Action pull-down menu of the Dialog Setup page:

- ***Query Result Columns for Trash Can***: New option used to configure columns to show at the search results of the Trash Can. Unlike other query results, this one allows adding Deleted and Deleter columns.
- ***Query Result Columns for Web***: No longer available since the Web pages are superseded by Content Station.
- ***Query Result Columns for Smart Browser***: No longer available since Smart Browser is superseded by Content Station.

Access rights

Under the File Access section of the Profile Maintenance page, there is one additional option named "Delete Permanently" (internally called "Purge"). For existing, migrated and new profiles, this option is disabled by default. When enabled, it allows users to remove objects from the Trash Can.

The existing profile option "Delete" has an additional meaning. It also indicates whether the user can Restore objects from the Trash Can.

Note that access rights options only affect brand admins and end-users. Not system admins.

Integration

The server reports all kinds of problems back to clients so they can ask the user for confirmation or just inform the user to what extent the operation was successful.

The server is backwards compatible with v7 clients which only know about SOAP faults. Those clients simply raise errors instead of confirmations. No objects are listed in the dialogs. The server tries to apply the operation for all objects, even when an error has occurred.

The following changes are made to the workflow interface (SCEnterprise.wsdl):

- ***AreaType***: New type introduced to filter objects from the Trash Can. Currently supported values are Workflow and Trash.
- ***WorkflowMetaData***: Has new object properties: Deleter and Deleted that are set when the object is deleted.
- ***QueryObjects***: Request has a new Areas parameter used to query objects from the Trash Can.
- ***NamedQueries***: Queries for Trash Can do not travel through Named Queries. Use QueryObjects instead.
- ***RestoreObjects***: New service that restores objects from the Trash Can back into the workflow.
- **DeleteObjects service:** Since deleted objects do not leave the system yet, client applications might want to know what happened (for example to update the Trash Can that is currently still opened in the background). Therefore this service returns objects that are deleted (which is a change). A new “Areas” parameter tells if the object needs to be deleted from the workflow (v7.0 behavior) or needs to be deleted permanently from the Trash Can.

The combinations of parameter values result in the following system operations:

Permanent	Areas	System operation
TRUE	Workflow	Permanently delete
TRUE	Trash	Permanently delete
TRUE	Workflow + Trash	Error (not supported)
FALSE	Workflow	Delete (send to Trash Can)
FALSE	Trash	Error (makes no sense)
FALSE	Workflow + Trash	Error (not supported)

- ***GetObjects***: Request has a new “Areas” parameter that allows to get the thumbnail and preview for objects listed in Trash Can.
- ***GetDialog***: Request has a new “Areas” parameter that allows to get properties for objects listed in Trash Can as configured for the Preview dialog (at Dialog Setup).
- ***GetVersion***: Request has a new “Areas” parameter that allows to view a version of an object in the Trash Can.
- ***ListVersions***: Request has a new “Areas” parameter that allows to request for all versions of an object in the Trash Can.

Live updates / N-casting

To support live updates between Search results, Inbox and Trash Can, some changes have been made to the messages that are broadcasted or multi-casted over the network to all client applications listening for updates:

- ***DeleteObject (#4)*** - Before v8, the ID was sent only. Since v8, the following properties are sent: ID (object), Type (object) [1], Name (object), PublicationId, IssueIds, EditionIds, SectionId, StateId, Deleted, Deleter, RouteTo (user), LockedBy (user), Version (object), UserId and Permanent [9].
  - [9] The Permanent property is set 'true' when the object is deleted permanently, or 'false' when the object is deleted (sent to Trash Can).
- ***RestoreObject (#23)*** - New event, fired for each object that was restored from the Trash Can. Aside from the default set of properties, Deleted and Deleter properties are sent too.

Server Plug-ins

There are new connector types added that can be implemented by connectors of server plug-ins, and for two existing connector types the behavior has changed:

- ***WflRestoreObjects_EnterpriseConnector****: New connector type. Called when user restores objects from the Trash Can. Called only once for multiple selected objects.
- ***WflDeleteObjects_EnterpriseConnector****: Existing connector type. Now also called when a user deletes objects permanently from the Trash Can. So checking the new Areas parameter is required.
- ***WflSetObjectProperties_EnterpriseConnector***: Existing connector type. Now also called when an admin moves objects from one issue or category to another. Can also be called for objects in the Trash Can. So checking the new Areas parameter is required.

For the restore and delete connectors, the new Deleted and Deleter object properties will be set and are accessible through:

```
$response->Objects[n-1]->MetaData->WorkflowMetaData
```

Note that for all connectors, the ID, IDs and Params elements are flattened to just IDs. That means, no matter what client applications send, the server will make sure the IDs are set (resolved from others) and others are made null. When IDs is null, it means 'for all objects'.

Handling errors for multiple objects

Assume the user has made a multiple selection of some objects and does a Delete or Restore operation, for which some objects have no problems, while some others have errors.

v8 client with v8 server

For a v8 client talking to v8 server, the SOAP messages looks like the examples below.

The initial request:

```
<DeleteObjects>
        <Ticket>...</Ticket>
        <IDs><String>51</String><String>52</String><String>53</String></IDs>
        <Permanent>false</Permanent>
        <Params xsi:nil="true"/>
        <Areas><Area>Workflow</Area></Areas>
</DeleteObjects>
```

The response for which an error dialog and confirmation dialog is raised:

```
<DeleteObjectsResponse>
	<IDs>
		<String>51</String>
		<String>53</String>
	</IDs>

	<Reports>
		<ErrorReport>
			<BelongsTo>
				<Type>Object</Type>
				<ID>52</ID>
				<Name xsi:nil="true"/>
				<Role xsi:nil="true"/>
			</BelongsTo>
			<Entries>
				<ErrorReportEntry>
					<Entities/>
					<Message>Access denied (S1002)</Message>
					<Details/>
					<ErrorCode>S1002</ErrorCode>
					<MessageLevel>Warning</MessageLevel>
				</ErrorReportEntry>
			</Entries>
		</ErrorReport>
	</Reports>
</DeleteObjectsResponse>
```

Note that there can still be SOAP faults instead of responses, for example when the ticket has expired. Errors mentioned above are about individual objects causing problems only.

v7 client with v8 server

For a v7 client talking to a v8 server, the SOAP messages looks like the examples below.

The initial request:

```
<DeleteObjects>
	<Ticket>...</Ticket>
	<IDs><String>51</String><String>52</String><String>53</String></IDs>
	<Permanent>false</Permanent>
</DeleteObjects>
```

The responses for which one error dialog is raised:

```
<SOAP-ENV:Fault>
	<faultcode>SOAP-ENV:Client</faultcode>
	<faultstring>Access denied (S1002)\nObject locked (Sxxxx)</faultstring>
	<faultactor/>
	<detail>53 (D)\n52</detail>
</SOAP-ENV:Fault>
```

As you can see, there is no longer a DeleteObjectsReponse but a true SOAP fault. All errors are collected and returned once. When there are many objects, it would be very hard for users to resolve the problem(s) and retry. This happens during migrations so it is a temporary problem.

Empty the Trash Can

Client applications can use the Params element to implicitly specify all objects that fit in the current search results the user is facing. Or they can use IDs element to explicitly specify the objects the user has selected. For example, some objects selected by the user or, all objects that fit on the first page of the search results. When both IDs and Params elements are set to nil, Permanent is 'true' and Areas is 'Trash', then the entire Trash Can is emptied.

To prevent many server plug-ins from having to resolve which object IDs are involved, the server calls QueryObjects internally (on service level) to resolve the IDs. It then passes

through the IDs and nullifies the Params element. In other terms, server plug-in may assume that the IDs element is always set.

Empty the system's Trash Can:

```
<DeleteObjects>
	<Ticket>...</Ticket>
	<IDs xsi:nil="true"/>
	<Permanent>true</Permanent>
	<Params xsi:nil="true"/>
	<Areas><Area>Trash</Area></Areas>
</DeleteObjects>
```

Trash the current query results:

```
<DeleteObjects>
	<Ticket>...</Ticket>
	<IDs xsi:nil="true"/>
	<Permanent>false</Permanent>
	<Params>...</Params> <!-- current query params used -->
	<Areas><Area>Workflow</Area></Areas>
</DeleteObjects>
```

Empty objects from Trash Can that were deleted by someone specific:

```
<DeleteObjects>
	<Ticket>...</Ticket>
	<IDs xsi:nil="true"/>
	<Permanent>true</Permanent>
	<Params>
		<Param>
			<Property>Deleter</Property>
			<Operation>=</Operation>
			<Value>John</Value>
			<Special xsi:nil="true"/>
		</Param>
	</Params>

	<Areas><Area>Trash</Area></Areas>
</DeleteObjects>
```

Empty objects from Trash Can that are owned by a specific brand:

```
<DeleteObjects>
	<Ticket>...</Ticket>
	<IDs xsi:nil="true"/>
	<Permanent>true</Permanent>
	<Params>
		<Param>
			<Property>PublicationId</Property>
			<Operation>=</Operation>
			<Value>123</Value>
			<Special xsi:nil="true"/>
		</Param>
	</Params>

	<Areas><Area>Trash</Area></Areas>
</DeleteObjects>
```

Dossier Labels [since 9.1]

Object labels can be created for Dossiers (and Dossier Templates) in order to filter Dossiers by one or more of these labels so that only those objects are shown. Once a label is created, it can be updated (renamed) or deleted for the Dossier. When a Dossier gets created (CreateObjects) it inherits the labels from its template.

The following example creates a ‘Foo’ label for a Dossier:

```
<CreateObjectLabels>
    <Ticket>...</Ticket>
    <ObjectId>123</ObjectId> <!-- Dossier Id-->
    <ObjectLabels>
        <ObjectLabel>
            <Id xsi:nil="true"/>
            <Name>Foo</Name> <!-- The name of the Dossier label -->
        </ObjectLabel>
    </ObjectLabels>
</CreateObjectLabels>
```

```
<CreateObjectLabelsResponse>
    <ObjectLabels>
        <ObjectLabel>
            <Id>1</Id> <!-- The id of the created label -->
            <Name>Foo</Name>
        </ObjectLabel>
    </ObjectLabels>
</CreateObjectLabelsResponse>
```

This is how to rename the ‘Foo’ label into a ‘Bar’ label:

```
<UpdateObjectLabels>
    <Ticket>...</Ticket>
    <ObjectLabels>
        <ObjectLabel>
            <Id>1</Id>
            <Name>Foo</Name>
        </ObjectLabel>
    </ObjectLabels>
</UpdateObjectLabels>
```

```
<UpdateObjectLabelsResponse>
    <ObjectLabels>
        <ObjectLabel>
            <Id>1</Id>
            <Name>Bar</Name>
        </ObjectLabel>
    </ObjectLabels>
</UpdateObjectLabelsResponse>
```

Deleting the label goes like this:

```
<DeleteObjectLabels>
    <Ticket>...</Ticket>
    <ObjectLabels>
        <ObjectLabel>
            <Id>1</Id>
            <Name xsi:nil="true"/>
        </ObjectLabel>
    </ObjectLabels>
</DeleteObjectLabels>

<DeleteObjectLabelsResponse/>
```

When an object is contained by a Dossier, the object can be labelled as well. However, only labels that were created for the parent Dossier can be chosen. They therefore travel with the ‘contained’ object relation (with Dossier as parent and the object as child) whereas labels for Dossiers travel directly with the object.

For contained objects we speak of ‘adding’ and ‘removing’ labels (instead of ‘creating’ and ‘deleting’). These operations work with labels that already exist for the parent Dossier of the contained object.

This is how to add an existing label to a contained object:

```
<AddObjectLabels>
    <Ticket>...</Ticket>
    <ParentId>123</ParentId> <!-- Dossier Id-->
    <ChildIds>
        <String>456</String> <!-- id of object, contained by the Dossier (ParentId) -->
    </ChildIds>
    <ObjectLabels>
        <ObjectLabel>
            <Id>1</Id>
            <Name xsi:nil="true"/>
        </ObjectLabel>
    </ObjectLabels>
</AddObjectLabels>

<AddObjectLabelsResponse/>
```

And this way you can remove that label again (from the contained object):

```
<RemoveObjectLabels>
    <Ticket>...</Ticket>
    <ParentId>123</ParentId> <!-- Dossier Id-->
    <ChildIds>
        <String>456</String> <!-- id of object, contained by the Dossier (ParentId) -->
    </ChildIds>
    <ObjectLabels>
        <ObjectLabel>
            <Id>1</Id>
            <Name xsi:nil="true"/>
        </ObjectLabel>
    </ObjectLabels>
</RemoveObjectLabels>

<RemoveObjectLabelsResponse/>
```

Labels defined for a Dossier can be retrieved through the *GetObjects* service by using a new *RequestInfo* option named 'ObjectLabels' as follows:

```
<GetObjects>

    <Ticket>...</Ticket>
    <IDs>
        <String>123</String> <!-- Dossier id -->
    </IDs>
    ...
    <RequestInfo>
        ...
        <String>ObjectLabels</String> <!-- new option since 9.1 -->
        ...
    </RequestInfo>
    ...
```

Because labels are defined for the Dossier itself, the server returns them directly under the *Object* element as shown here:

```
<GetObjectsResponse>
    <Objects>
        <Object>
            <MetaData>
                <BasicMetaData>
                    <ID>123</ID> <!-- Dossier id -->
                ...
            </MetaData>
            ...
            <Relations>
                <Relation>
                    <Parent>123</Parent> <!-- Dossier id -->
                    <Child>456</Child> <!-- id of object, contained by the Dossier (Parent) -->
                    <Type>Contained</Type>
                    ...
                    <ObjectLabels>
                        <ObjectLabel>
                            <Id>1</Id>
                            <Name>Foo</Name>
                        </ObjectLabel>
                    </ObjectLabels>
                </Relation>
            </Relations>
            ...
            <ObjectLabels>
                <ObjectLabel>
                    <Id>1</Id>
                    <Name>Foo</Name>
                </ObjectLabel>
            </ObjectLabels>
            ...
```

Similar as for Dossiers, labels can be requested for ‘contained’ objects as well. For example:

```
<GetObjects>

    <Ticket>...</Ticket>
    <IDs>
        <String>456</String> <!-- id of object, contained by the Dossier -->
    </IDs>
    ...
    <RequestInfo>
        ...
        <String>ObjectLabels</String> <!-- new option since 9.1 -->
        ...
    </RequestInfo>
    ...
```

Because the labels are defined for the Dossier (and not for the ‘contained’ object) the server returns them under the ‘Contained’ *Relation* element with the Dossier as parent:

```
<GetObjectsResponse>
    <Objects>
        <Object>
            <MetaData>
                <BasicMetaData>
                    <ID>456</ID> <!-- id of object, contained by the Dossier -->
                ...
            </MetaData>
            ...
            <Relations>
                <Relation>
                    <Parent>123</Parent> <!-- Dossier id -->
                    <Child>456</Child> <!-- id of object, contained by the Dossier (Parent) -->
                    <Type>Contained</Type>
                    ...
                    <ObjectLabels>
                        <ObjectLabel>
                            <Id>1</Id>
                            <Name>Foo</Name>
                        </ObjectLabel>
                    </ObjectLabels>
                </Relation>
            </Relations>
            ...
            <ObjectLabels/> <!-- empty: no object labels for this object type -->
            ...
```

Suggestions and Auto-completion [since 9.1]

For a functional explanation of these two features and how to set them up, see the 'Tags' section of the Enterprise Server Online Help.

Enterprise Server offers two new connector interfaces for Server Plug-ins to implement:

- A suggestions connector interface (SuggestionProvider_EnterpriseConnector)
- An auto-complete connector interface (AutocompleteProvider_EnterpriseConnector)

When a plug-in implements the suggestions connector interface, it becomes a suggestions provider. And, when implementing the auto-complete connector interface, it becomes an auto-complete provider. For example: the OpenCalais plug-in is a suggestion provider, and the Drupal7 plug-in is an auto-complete provider.

When a user opens a Publish Form, Enterprise Server requests the installed providers whether or not they support all custom object properties that are configured for that form and includes these in the response.

The following example shows these attributes in bold:

```
<GetDialog2Response>
    ...
    <Tabs>
        ...
        <DialogWidget>
            <PropertyInfo> <!-- metadata setup -->
                <Name>C_DRUPAL_CITIES</Name>
                ...
                <!-- 9.1 -->
                <TermEntity>Cities</TermEntity>
                <SuggestionEntity>City</SuggestionEntity>
                <AutocompleteProvider>Drupal7</AutocompleteProvider>
                <SuggestionProvider>OpenCalais</SuggestionProvider>
                ...
```

The property attributes explained:

- ***TermEntity:*** Abstract name for a term. When given, it could be recognized by any auto-complete provider to help the user filling in the property.
- ***SuggestionEntity:*** Abstract name for a Suggestion. It defines the entity for the types of tags that need to be suggested by the Suggestion Provider, such as 'City', 'Movie', 'TVShow' and so on.
  Note that suggestion providers are loosely coupled regarding the publishing providers. For example, OpenCalais and Drupal integrations do not need to know from each other. The challenge here is that for a custom object property, only the publishing connector and the external publishing system know what it is about. For example, a Cities field from Drupal, the custom property could be named C_DRUPAL_CITIES, which means nothing to OpenCalais. To tackle this challenge, a new attribute is added to the *PropertyInfo* element and is named *SuggestionEntity*. Having this abstract information in place, both integrated systems know where the property stands for.
- ***AutocompleteProvider:*** The internal name of the Server Plug-in that has an auto-complete connector that supports this custom object property.
- ***SuggestionProvider:*** The internal name of the Server Plug-in that has an auto-complete connector that supports this custom object property.

Auto-complete

Now the Publish Form is shown, the user can fill in the field. Let's say that the user has already tagged "Amsterdam" in the Cities field of the Publish Form. Now the user types "Ams" to create another tag. Even though it matches with "Amsterdam" it is no longer suggested.

The client fires the following request:

```
<Autocomplete>
    <Ticket>...</Ticket>
    <AutocompleteProvider>AutocompleteSample</AutocompleteProvider> <!-- plugin name -->
    <PublishSystemId xsi:nil="true"/> <!-- 9.1 GUID -->
    <ObjectId>123</ObjectId> <!-- object id of Publish Form which the user is filling in -->
    <Property>
        <Name>C_DRUPAL_CITIES</Name> <!-- custom obj prop introduced by Drupal connector -->
        <Entity>City</Entity> <!-- entity of the property -->
        <IgnoreValues> <!-- tags that were already selected before in the Cities field -->
            <String>Amsterdam</String>
        </IgnoreValues>
    </Property>
    <TypedValue>Ams</TypedValue> <!-- user typed value (so far) for the custom Cities field -->
</Autocomplete>
```

The server returns the following response:

```
<AutocompleteResponse>
    <Tags>
        <!-- Amsterdam is not listed because it was requested to ignore -->
        <!-- Amstel is not listed because the entity is not a City -->
        <AutoSuggestTag>
            <Value>Amstelveen</Value>
            <Score>0.85</Score>
            ...
        </AutoSuggestTag>
        <AutoSuggestTag>
            <Value>Adamstown</Value> <!-- matches half-way -->
            <Score>50</Score>
            ...
        </AutoSuggestTag>
    </Tags>
</AutocompleteResponse>
```

Suggestions

Now the Publish Form is shown, the user can fill in the field. Let's say that the user has typed an article about Amsterdam and presses the 'Suggest' button:

```
<Suggestions>
    <Ticket>...</Ticket>
    <SuggestionProvider>OpenCalais</SuggestionProvider> <!-- internal plugin name -->
    <ObjectId>123</ObjectId> <!-- object id of Publish Form which the user is filling in -->
    <MetaData> <!-- CS sends strings, multilines and text components only -->
        <MetaDataValue>
            <Property>C_DRUPAL_HEAD</Property>
            <Values>
                <String>Facts about Amsterdam</String>
            </Values>
        </MetaDataValue>
        <MetaDataValue>
            <Property>C_DRUPAL_BODY</Property>
            <Values>
                <String>The Amstel river and the Noordzee canal run through this city.</String>
            </Values>
        </MetaDataValue>
    </MetaData>
    <SuggestForProperties> <!-- for optimization of the response -->
        <AutoSuggestProperty>
            <Name>C_DRUPAL_CITIES</Name>
            <Entity>City</Entity>
            <IgnoreValues xsi:nil="true"/>
        </AutoSuggestProperty>
        <AutoSuggestProperty>
            <Name>C_DRUPAL_RIVERS</Name>
            <Entity>Natural Feature</Entity>
            <IgnoreValues xsi:nil="true"/>
        </AutoSuggestProperty>
    </SuggestForProperties>
</Suggestions>
```

The server returns the following response:

```
<SuggestionsResponse>
    <SuggestedTags>
        <!-- "Noordzee canal" is not returned; OpenCalais tells it has entity "Facility" -->
        <EntityTags>
            <Entity>City</Entity>
            <Tags>
                <AutoSuggestTag>
                    <Value>Amsterdam</Value>
                    <Score>0.52</Score>
                    ...
                </AutoSuggestTag>
            </Tags>
        </EntityTags>
        <EntityTags>
            <Entity>Natural Feature</Entity>
            <Tags>
                <AutoSuggestTag>
                    <Value>Amstel river</Value>
                    <Score>0.38</Score>
                    ...
                </AutoSuggestTag>
            </Tags>
        </EntityTags>
    </SuggestedTags>
</SuggestionsResponse>
```

Download files directly from Content Source [since 9.7]

In Enterprise 9.7, improvements have been made in order to make communications with external file sources faster. It allows for clients to request direct file links to a Content Source, with which they can download file content directly from the Content Source, instead of it having to go through Enterprise Server first.

This omits Enterprise Server having to download the file content from the Content Source to a temporary folder and sending it to the requestor through DIME or Transfer Server.

In order to facilitate this new feature, the *GetObjects* service has been expanded with two extra options:

- ***RequestInfo*** has been expanded with a *ContentSourceFileLinks* option. Setting this option tells Enterprise Server that direct content source file links are requested by the client.
- ***SupportedContentSources*** has been added as a property. It contains a list of Content Sources that the requestor can directly access in order to download the file content.
  Note: If the requested object does not match any of the content sources given here, Enterprise will default to the old behaviour.

Aside from this, the *Attachment* element has a new *ContentSourceFileLink* property. This property may contain a direct link to the Content Source file when this has been requested.

Below is an example using Elvis as the Content Source. Requesting an object that originates from Elvis can be done as follows:

```
<GetObjectsRequest>
	...
	<RequestInfo>ContentSourceFileLinks</RequestInfo>
	...
	<SupportedContentSources>ELVIS</SupportedContentSources>
	...
</GetObjectsRequest>
```

Which returns a response like the following:

```
<GetObjectsResponse>
	<Objects>
		<Object>
			...
			<Files>
				<Attachment>
					...
					<FileUrl xsi:nil="true"/>
					...
					<ContentSourceFileLink>http://...</ContentSourceFileLink>
					...
				</Attachment>
				...
			</Files>
			...
		</Object>
	<Objects>
</GetObjectsResponse>
```

When talking to an older version of Enterprise Server or when the direct download feature is not supported by the Content Source, the returned *ContentSourceFileLink* is set to nil. In that case, the client should fall back to the *FileUrl* or *Content* elements instead to download the file via resp. the File Transfer Server or Enterprise Server (DIME).

Automated Print Workflow [since 9.8]

This feature is about Images and Articles contained in a Dossier that can be automatically placed on a Layout without the need to open the document in the InDesign client.

To make this happen, text frames and graphic frames on the Layout should be labelled with Element Labels and grouped into so-called InDesign Articles.

The feature requires the following minimum versions:

- Enterprise Server 9.8.0
- Smart Connection 10.2.1 for ID/IDS CC2014
- Content Station 9.8.0

Prepare layouts

When saving a Layout in Smart Connection the InDesign Articles and their frames are communicated to Enterprise Server:

```
<SaveObjects>
	<Objects>
		<Object>
			...
			<Relations>
				<Relation>
					...
					<Placements>
						...
			...
			<Placements>
				...
			<InDesignArticles>
				<InDesignArticle>
					<Id>246</Id>
					<NameArticle 1</Name>
				</InDesignArticle>
				...
```

In the *Object* structure there are two places where *Placements* can be found. If a frame contains a text component of an Enterprise Article object or a graphic of an Enterprise Image object, it is defined under *Objects->Relations->Placements*. Or else, when the frame belongs to any of the InDesign Articles of the layout, it is defined under *Object->Placements*. This way, one frame will never be communicated in both places.

Place dossier

The contents of a Dossier can be placed on a Layout. The user clicks the Create Layout button in the channel view of a Dossier and picks a Layout. Content Station requests for the available InDesign Articles for that layout as follows:

```
<GetObjects>
	...
	<RequestInfo>
		<String>InDesignArticles</String>
		...
	...
```

Enterprise Server returns the InDesign Articles:

```
<GetObjectsResponse>
	<Objects>
		<Object>
			...
			<InDesignArticles>
				<InDesignArticle>
					<Id>246</Id>
					<NameArticle 1</Name>
				</InDesignArticle>
				...
```

The user picks one of the listed InDesign Articles. Content Station can not edit Layouts. Instead, it locks the Layout for editing (LockObjects) and indirectly 'places' the Dossier on the Layout by creating an Operation for the Layout:

```
<CreateObjectOperations>
	<Ticket>...</Ticket>
	<HaveVersion>
		<ID>1178</ID>
		<Version>0.2</Version>
	</HaveVersion>
	<Operations>
		<ObjectOperation>
			<Id>9f7dd5f4-281a-e885-3707-b642faa3c248</Id>
			<Name>PlaceDossier</Name>
			<Type>AutomatedPrintWorkflow</Type>
			<Params>
				<Param>
					<Name> => EditionId
					<Value> => 1
				</Param>
				<Param>
					<Name> => DossierId
					<Value> => 1181
				</Param>
				<Param>
					<Name> => InDesignArticleId
					<Value> => 246
				</Param>
			</Params>
		</ObjectOperation>
	</Operations>
</CreateObjectOperations>
```

Note: Instead of picking a Layout, the user can pick a Layout Template. In that case Content Stations calls the *InstantiateTemplate* service instead.

The server queries for any server plug-in connectors that implement this interface:

AutomatedPrintWorkflow_EnterpriseConnector

For each connector found, it calls the *resolveOperation()* function to let the connector change the Operations (when needed) before they are actually stored in the database for the Layout. In this example, the *AutomatedPrintWorkflow* plug-in is installed which queries the database and finds out that the Dossier has one Article text component and one Image that could be matched with the InDesign Article of the Layout. Therefore it 'resolves' the *PlaceDossier* operation into two other operations, *PlaceArticleElement* and *PlaceImage*.

```
<CreateObjectOperationsResponse>
	...
	<Operations>
		<ObjectOperation>
			<Id>08a8d9e6-236f-57f4-e677-e71ab95e77f6</Id>
			<Name>PlaceArticleElement</Name>
			<Type>AutomatedPrintWorkflow</Type>
			<Params>...</Params>
		</ObjectOperation>
		<ObjectOperation>
			<Id>1cd68eb0-d5d2-6e05-c68a-e88e64fb1c3a</Id>
			<Name>PlaceImage</Name>
			<Type>AutomatedPrintWorkflow</Type>
			<Params>...</Params>
		</ObjectOperation>
	</Operations>
</CreateObjectOperations>
```

After a while, Content Station releases the Layout lock (UnlockObjects) to give way for further processing. When the IDS Automation feature is enabled, an *IDS_AUTOMATION* job is created during the Operation creation. This job gets picked up from the queue and is processed in the background. Basically it requests InDesign Server (through SOAP) to run a very simple Javascript module (indesignserverjob.jsx). The script requests Smart Connection to LogOn, open and save the Layout and LogOff again. This is to let Smart Connection for InDesign Server process the Object Operations for the Layout.

After Smart Connection does the LogOn, it also requests (through the admin web services) for any so-called sub-applications:

```
<GetSubApplicationsRequest>
	<Ticket>...</Ticket>
	<ClientAppName>InDesign Server</ClientAppName>
</GetSubApplicationsRequest>
```

In this example, the *AutomatedPrintWorkflow* plug-in hooks in the *runAfter()* function of this web service and provides a JavaScript module that is provided by a server plug-in:

```
<GetSubApplicationsResponse>
	<SubApplications>
		<SubApplication>
			<ID>SmartConnectionScripts_AutomatedPrintWorkflow</ID>
			<Version>1.6</Version>
			<PackageUrl>
				http://127.0.0.1/Enterprise/server/plugins/AutomatedPrintWorkflow/
idscripts.zip
			</PackageUrl>
			<DisplayName>Automated Print Workflow</DisplayName>
			<ClientAppName>InDesign Server</ClientAppName>
		</SubApplication>
	</SubApplications>
</GetSubApplicationsResponse>
```

Smart Connection downloads and extracts the package that contains a JavaScript module and loads it into IDS. The indesignserverjob.jsx script is still running and continues with opening the Layout. While opening, Smart Connection downloads and locks the Layout for editing. It finds the Operations sent along with the Layout (GetObjects) and provides those Operations to the JavaScript module. The module uses the information of the Operations to do the actual placing. In the example, this is where the Article text components and the Images are placed. The script continues and saves the Layout. Smart Connection generates page previews and PDFs (if needed) and creates a new version in Enterprise and releases the lock.

When the IDS Automation feature is disabled, the Layout does not get processed directly, but has to wait until someone opens the Layout in InDesign client. Then the exact process takes place in InDesign client as described above for InDesign Server. It could happen that

many operations are created before the Layout gets opened. That slows down opening the Layout since all those operations needs to be processed. For this performance reason it is recommended to enable the IDS Automation feature when the Automated Print Workflow feature is enabled.

Customizations

As you may have noticed in the previous paragraph, the Automated Print Workflow feature is built into a server plug-in. While Enterprise introduces a mechanism to build such a feature, it also provides the *AutomatedPrintWorkflow* plug-in that implements the default behaviour (business logics) and does place operations.

Basically, all you need is a server plug-in that provides a server module and client module. The server module queries the text components and images in the database and the client module places them in the InDesign Article frames.

This allows you to:

- Disable the *AutomatedPrintWorkflow* plug-in and build your own solution with different or customer specific business logics that determines which text component or image should be placed on which InDesign Article frame.
- Add another plug-in that introduces more operations. See also example on Labs.

Default behaviour

The *AutomatedPrintWorkflow* plug-in implements the default behaviour. It works when Element Labels are unique within each InDesign Article (e.g. there should be one head, intro, body and graphic) and unique within Dossiers. It has the following reasoning:

1. Resolve the Layout frames that belong to the selected InDesign Article.
2. Exclude duplicate InDesign Article frames; Those can not be uniquely matched.
3. Collect frame types (graphic or text) from the resolved InDesign Article frames.
4. Determine object types (Article, Image) that are compatible with the frame types.
5. Take the Issue to which the Layout is assigned to and take the Edition from the InDesign Article frame.
6. Search in the Dossier for Articles and Images that are assigned to the same Issue and Edition via the Dossier (relational targets).
    a. Exclude Articles and Images for which the user has no read access rights.
    b. Exclude Articles and Images in Personal status and routed to another user.
    c. Exclude Articles that are placed already and the user has no Allow Multiple Article Placements access rights.
7. For each candidate Article:
    a. Resolve the Layout frames (placements) of the found Article.
    b. Exclude frames with duplicate element label; Those can not be uniquely matched.
    c. Exclude graphic frames; This is not supported by Smart Connection.
    d. Exclude frame when the same frame label also exists in the same or other Article.
    e. Match the frame element labels with the InDesign Article frame element labels.
8. For each candidate Image:
    a. Only make a match when InDesign Article has one graphic frame and exactly one Image was found in the Dossier.
9. For each match that could be made, add new Object Operations to the Layout.
10. For each frame that could not be matched, add the *ClearFrameContent* operation.

System Admin in action

Adding Sub-Applications to Content Station [since 9.0]

This chapter won't tell you how to develop your sub-application in Flex nor how to physically embed it into Content Station. Instead it describes how to enable admin users to configure access rights for your own sub-applications that run inside Content Station and how to provide a download URL which enables users to install or upgrade your application with ease.

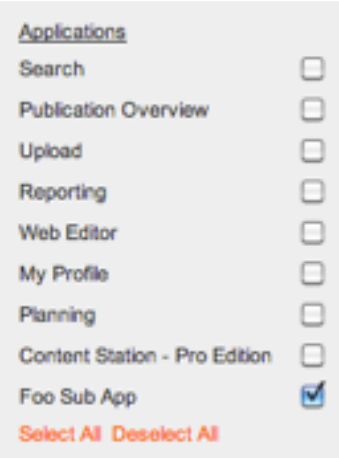
First you need to develop a Server Plug-in. Since Enterprise 9.0 that is as simple as creating a folder in the Enterprise/config/plugins folder with a name (camel case) followed by running the Server Plug-ins page. In the folder you'll find a new PluginInfo.php file that was generated for you. In that file you should uncomment the following option (by removing the leading // slashes):

```
// 'SysGetSubApplications_EnterpriseConnector',
```

Save the file and run the Server Plug-ins page again, which now generates a connector that implements the SysGetSubApplications_EnterpriseConnector interface. Open the <YourPluginName>_SysGetSubApplications.class.php file and implement the runAfter() function as follows:

```
final public function runAfter( SysGetSubApplicationsRequest $req, SysGetSubApplicationsResponse
&$resp )
{
	if( is_null($req->ClientAppName) || // request for all clients?
		$req->ClientAppName == 'ContentStation' ) { // request for this client only?

		require_once BASEDIR.'/server/interfaces/services/sys/DataClasses.php';
		$subApp = new SysSubApplication();
		$subApp->ID = 'ContentStation_FooSubApp';
		$subApp->Version = '1.0.0 Build 1';
		$subApp->PackageUrl = 'http://foosubapp.com';
		$subApp->DisplayName = 'Foo Sub App';
		$subApp->ClientAppName = 'ContentStation';
		$resp->SubApplications[] = $subApp;
	}
}
```



Having the Server Plug-in activated, when the admin user now runs the Profile Maintenance page (under Access Profiles menu), the SysGetSubApplications service is called and so the code snippet above is executed. As a result, your sub-application is listed under the Applications section at the Profile Maintenance page. This enables the admin user to set up a profile that gives end-users access to your sub-application. On the Brand Maintenance page under User Authorizations, the admin user can select that profile for certain user groups.

With the access profiles in place, when the end-user logs in to Content Station, Enterprise Server returns the profiles that are configured for that user. By default, all applications are assumed to be enabled. For such applications, simply no information is returned in the profile at all. Only when an application has been disabled in a profile will an entry be added to the LogOnResponse for that AppFeature. The value for it is set to "No" as shown in the following fragment of the LogOnResponse:

```
<LogOnResponse>
	...
	<FeatureProfiles>
	...
		<FeatureProfile>
			<Name>Access Sample</Name>
			<Features>
				...
				<AppFeature>
					<Name>ContentStation_FooSubApp</Name>
					<Value>No</Value>
				</AppFeature>
```

In the very same response, Content Station checks which profiles have been configured for the Publications (= Brands). The following fragment shows how access profiles are organized per Brand:

```
<LogOnResponse>
	...
	<Publications>
		<PublicationInfo>
			...
			<FeatureAccessList>
				<FeatureAccess>
					<Profile>Access Sample</Profile>
					<Issue xsi:nil="true"/>
					<Section xsi:nil="true"/>
					<State xsi:nil="true"/>
				</FeatureAccess>
			</FeatureAccessList>
			...
```

Looking at both fragments above, if any PublicationInfo (=Brand) has a FeatureAccess (=profile reference) that corresponds with a FeatureProfile (=profile definition) for which the sub-application is not listed, Content Station will show your sub-application since the user has access to it (through at least one of the Brands). Obviously that is only the case when your sub-application is returned through the SysGetSubApplications service.

Enterprise services API

Comments added to the WSDL documents explain on a very detailed level the meaning of operation parameters. Some services are simply too complex to comment that way. Those are explained in this section.

Workflow dialogs

Through Dialog Setup pages, system administrators can configure workflow dialogs per Brand, object type and action. When a client is about to show a workflow dialog, such definitions are requested through the GetDialog workflow service (as specified in the SCEnterprise.wsdl). Whenever the user selects another Brand, Issue, Category or status, the client requests for that dialog definition (requesting for GetDialog again) and redraws the entire dialog.

For Brand and overrule Issue selections, the redraw is needed because the entire workflow definition and dialog configuration could be different. For (normal) Issue selections the list of requested Dossiers could differ. For status- and Category changes, the Route To selection could differ. And, in all cases, the editable/read-only status could differ due to access rights.

System integrators can overrule the standard server behavior of the GetDialog service through server plug-ins. This is why clients should have no logics nor assume or predict certain behavior. Instead, they should listen to the GetDialog response and fully rely on that. The LogOn response should no longer be used by any v7 client to get definitions in relation to workflow dialogs. Also, there is no reason anymore to call the GetStates service since that is embraced by the GetDialog service.

History

- Since v5.0, the GetDialog service was introduced and used by the Web Editor only.
- Since v6.0, Publication Channel awareness has been added to the GetDialog service. Content Station started using it (but is still using the LogOn response too to build dialogs).
- Since v7.0, new features have been added to the service and InDesign / InCopy clients started using the service. The service itself is also improved and logics are moved from the clients to the service:
    - The returned PublicationInfo element tree has been pruned, clarifying what response information should be used by the clients.
    - The enabled status of properties is determined by the service (instead of the clients).
    - The Route To pre-selection is determined by the service (instead of the clients).

GetDialog service

Element structure	Description
Ticket	[Mandatory] The ticket as retrieved through the LogOn response.
ID	Object ID. Nil for Create action. Mandatory for any other action.
Publication	Publication ID. Nil when ID, Layout or Template is provided, but is mandatory to redraw dialog.
Issue	Issue ID. Nil for initial draw. Mandatory to redraw the dialog.
Section	Section ID. Nil for initial draw. Mandatory to redraw the dialog.
State	[v7.0] Status ID. Nil for initial draw. Mandatory to redraw the dialog.
Type	Object type.
Action	Workflow operation. See Action definition for possible values. The empty value is not allowed in this context. For Save-As operations, the Create action should be used.
RequestDialog	Request for Dialog element at response time.
RequestPublication	Request for Publications and PublicationInfo elements at response time.
RequestMetaData	Request for MetaData element at response time. (Has no impact on Dialog->MetaData.)
RequestStates	Request for GetStatesResponse element at response time.
RequestTargets	[v6.0] Request for the Targets property (PropertyUsage element) at response time. Clients should pass “true” when they support the complex target widget that holds the Issues and Editions. The server determines where the Targets property should be placed in the dialog. Clients should then ignore the Issues and Editions property positions.
DefaultDossier	[v7.0] Dossier object ID. Request to populate the Dossiers element at response time as well as to return the Dossier property at the Dialog definition. The given Publication, Issue and Section are used to get Dossiers. The DefaultDossier is also used to set the default value as the Dossier property. If DefaultDossier is nil (or left out) no Dossier property nor Dossiers will be returned. See also [1].

Element structure	Description
Parent	[v7.0] Parent object ID. In the specific case when creating objects that are already placed (such as creating articles from a layout) the client knows that the object will be placed, but the server does not know yet. (Existing placement relations are resolved server-side.) When the Parent has targets, the RelatedTargets will hold targets of this additional parent too. And, instead of the current Issue, the parent's Issue is preselected (returned through GetDialogResponse -> Targets). This is because placed objects are assumed to travel along with their parents. Also, the Brand / Category are taken from the Parent when Publication / Section are not given. When the Parent is a Layout Module (or Layout Module Template) the Brand, Issue and Editions are inherited. The Publication and Issue parameters are then ignored.
Template	[v7.0] Template object ID. Should only be given when Action = "Create"; else nil. When creating objects, most MetaData should be taken from a template. Those should be pre-filled for the new object in the Create workflow dialog. Provide the object ID of the template (that was picked by user) to let the server inherit its MetaData structure. Nil (or left out) means that the object is not created from the template. For Save As operations, the Template parameter can be used to pass the original object ID from which the MetaData will be inherited. Unlike the Parent parameter, the template's Issue is not used for preselection. This is because in most cases there is a special templates issue, or templates do not have an Issue assigned.

[1] 'Dossier' is a property introduced in v7.0 and can be configured in the Dialog Setup Maintenance page. When a dialog is not customized, a default dialog is assumed by the server, which now also includes the Dossier property. Unlike other properties, the Dossier property is only shown in the dialogs when useful. That is, when the client and server can both handle it. If one doesn't, the Dossier property won't be shown in the dialog.

For client side, that means it is able to:

- Parse the Dossiers element in the GetDialog response.
- Draw a pull-down widget populated with those Dossiers.
- Pass the chosen Dossier ID (by the end user) through the object relations of the next request. For the Create dialog, that is the CreateObjects request.

For server-side, that means it is able to:

- Support the Object-in-Dossier relation. For example, for v7.0, Dossier-in-Dossier is not supported, and so Dossier and DossierTemplate objects won't get a Dossier property.
- Support implicit Dossier creation for 'next' called service. Since v7.0. this is implemented for the CreateObjects service, and so, only for Create dialogs, the Dossier property is shown.

When a client does meet all its criteria listed above, it passes 0 (zero) or a valid Dossier ID for the DefaultDossier parameter; else nil. When the server received a 0 (zero) or valid Dossier ID, and all its criteria listed above are met, (and the Dossier is configured in the Dialog Setup, or default dialog is picked), it returns the Dossier property (through the GetDialogResponse->Dialog elements).

Clients do not worry about server criteria, so they pass 0 (zero) or a valid Dossier ID for any object type, including Dossier objects. Let's assume that v8 supports Dossier-in-Dossier relations. When a v7 client is talking to a v8 server, the Dossier property will then be shown in the dialog, and will work well.

Let's assume that for Enterprise 8.0 the CopyTo dialog supports the Dossier property. Any v8 client passes a value for DefaultDossier parameter and the v8 Server returns the Dossier property, which is shown in the dialog. The 8.0 client then passes the chosen Dossier ID (by end user) through the CopyObjects service. But, when a v7 client is talking to a v8 server, the DefaultDossier parameter is set to nil, and no Dossier property is shown. And vice versa, when a v8 client is talking to a v7 server, the client passes a value for DefaultDossier parameter, but the server does not return the Dossier property, and so it is not shown in the dialog.

When the user is about to Create a new object, the server resolves and returns the Dossiers to let the user pick one. Even when there are no Dossiers found, the Dossier property is returned. Note that the "New Dossier" item (ID=-1) always exists in the list of Dossiers. This allows users to create a new Dossier, even when there are none found. There is also an empty item (ID=0) which means there is no need to create a Dossier. When a user submits the dialog, the client should pass the Dossier in the object relations (type Contained) of the CreateObjects request. (This should be done for ID <> 0 only.)

The Publication, Issue, Section and State parameters indicate the user's selection in the workflow dialog. By passing those parameters, clients ask the server how the dialog should be drawn for the given publication/Issue, regardless of the actual publication/issue of the object as stored in the database. So, these parameters are 'strongest' and overrule any other sever logics deriving publication/Issue from parent objects, template objects, configured current Issue, etc, etc. Important is that clients pass -no- Publication parameter the 'first time'. That way, server logics can kick in, but more over, custom server plug-ins can even overrule those logics by choosing specific publication/Issue. The ones that are initially picked by the server can be read from the GetDialogResponse -> Dialog -> MetaData.

The Publication, Section and State parameters imply what access profile is respected. For example, the "Change Pub/Issue/Category" option is resolved and so the properties mentioned are made read-only or editable (by GetDialog service).

The table below shows what combination of parameters are expected. Other combinations are not supported. As you can see, when asked for a Create action, no object ID should be passed, but a Type is required. For other actions, it is the other way around. And, only for redraw operations, Publication, Issue and Section should be passed. Except for creating new objects without Parent/Template; the client should pass a Publication, which should be taken from the context the user is working in. For example: the last used publication, or the publication currently selected in the Search pane.

Event	ID	Type	PT(2)	Pub	Issue	Section	Status
Action: Create(1) - initial	✘	✔	✘	✔	✔/✘ (4)	✘	✘
Action: Create(1) - initial	✘	✔	✔	✘	✘	✘	✘
Action: Other - initial	✔	✘	✔	✘	✘	✘	✘
Change: pub/overrule(3)	-	-	-	✔	✔/✘ (4)	✘	✘
Change: issue/sec/stat(5)	-	-	-	✔	✔	✔	✔

Legend:

✔ = valid ID (not empty, not zero)

✘ = nil

- = depends on the action (inherited from one of the first three lines above)

1) For Save As operations, the Action parameter should be set to Create. Note: Although the dialog caption says Check-In, technically we speak of a Create action.
2) Whether or not a Parent (ID) or Template (ID) was provided. This is used to derive the publication (and overrule Issue) from.
3) When user selects other Brand or overrule Issue, a GetDialog request is fired and the dialog is redrawn. The entire dialog definition can change, so a full redraw is required.
4) For overrule Issue: ✔, but for normal Issue: ✘.
5) When a user selects (normal) Issue, section or status, GetDialog request is fired to let the server reflect access rights profile settings to the dialog fields, and to refill certain lists, such as Dossiers, sections and statuses.

GetDialogResponse

Element structure	Description
Dialog	Workflow dialog definition.
↳ Tabs	Defines tabs and widgets to draw. The tab sequence represents the z-order with first the tab on top. For each tab, widgets are listed. The sequence needs to be respected with the first widget on top. For each widget, the usage is defined; whether or not to show read-only, mandatory, etc. When client supports the complex Targets widget, it should combine the Issues and Editions widgets and draw the Targets widget instead. The Targets position should be respected. If the client does not support the Targets widget, it should ignore it and simply draw Issues and Editions widgets.

Element structure	Description
↳ MetaData	For each widget under the Tabs element, a value is given to pre-fill / pre-select in the dialog.
Publications	The Brands (ids and names) for which user has access. Used to populate the Brand pull-down menu.
PublicationInfo	The configuration definition of the current Brand. For new objects, this is the requested Brand (through GetDialog -> Publication). For existing objects, this is the object's Brand.
↳ PubChannels	The Brand's publication channels, Issues and Editions definitions. Used to populate the Issues and Editions pull-down menus. When requested for an overrule Issue (through GetDialog -> Issue), the corresponding IssueInfo holds its Categories and Editions definitions, which is used to populate the pull-down menu instead.
↳ Categories	The Categories (ids and names) configured on Brand level. Used to populate the Category pull-down menu.
MetaData	The current object properties as stored in the database (regardless of the requested Brand/Issue). Used to create or update an object once the dialog is submitted. This way, calling the GetObjects service first is not needed. When requested for a template (GetDialog -> Template) much of the template's metadata is inherited. Typically used to create a new article from the article template.
GetStatesResponse	Statuses, users and user groups to which the object can be sent. Used to populate the Status and Route To pull-down menus.

Element structure	Description
Targets	For existing objects, these are the object targets as stored in the database. Except when a user is switching to/from another Brand or Overrule Issue. In that case, the object targets are replaced with one target to the current Issue of the default channel. For new objects, these are the initial targets derived from the given parent (layout), Dossier, or template. If there are no such derivable objects given, the current Issue of the default channel is the server's best guess. These targets should be shown (by the clients) when -initially- showing the dialog. And also, when a user has selected another Brand or overrule Issue, clients should clear the currently user built target list shown in the dialog and respect the targets return from the server. But, when a user selects a normal Issue, Category or status, a redraw is required too, in which case the current user targets are preserved (and the returned targets from the server are ignored) by clients.
RelatedTargets	The parent's targets, in case the object is placed and/or Dossier's relational* targets in case the object is contained by a Dossier. For example, the layout's targets in case an article is placed on that layout. There can be zero, one, or many parents/ Dossiers involved. A maximum of 5 parents are returned and a maximum of 5 Dossiers. When these limits are exceeded, a dummy item is returned named "...". * These are -not- the Dossier's targets, but the targets set for the contained object within the Dossier (called relational targets).
Dossiers	List of Dossiers (ids and names) available within the requested Brand/Issue. Only provided when requested for (GetDialog -> DefaultDossier). Used to populate the Dossiers pull-down menu.

Exceptional standard dialog behavior

Disabling the Editions property

For Task, Hyperlink, Library, Plan and Other objects, Editions are always disabled.

Disabling Brand / Issue / Editions property for Layout Module placements

For objects placed on Layout Modules (or Layout Module Templates), the Brand, Issue and Editions properties are disabled. This is because Editions are inherited from the module and are not allowed to change (in order for its placements to preserve data integrity).

Disabling Brand / Issue / Editions / Category property for placements

The Create dialogs for Articles and Image objects show disabled Brand, Issue, Editions and Category properties when they are created from a Layout.

Disabling Status property and the Personal Status

For existing objects, the Change Status access option is respected, and so you will risk having a disabled Status property. This is correct, except when the Personal status is currently selected. (Reason: no-one else can see the object other than the user to whom the object is routed to, but that user cannot change the status, and so an admin user would need to be asked to solve this problem, which is unwanted.) So, when an object is in the Personal Status, the Status property is enabled.

Disabling properties in the Set Properties dialog

When an object is already locked for editing, the Set Properties dialog shows all disabled properties. This is done by the client, not by the server (because InDesign/InCopy clients do pessimistic locking and so in both cases the object is locked and the server cannot tell the difference). This implies that the Server Plug-ins cannot alter the enabled status of the properties in this specific situation.

Automatic user input focus

For workflow dialogs, the focus (caret) is set to the Status property. For the Create and Copy To dialogs, or when the Status property is disabled, the focus is set to the Name field.

Limited Edition items

When an object is placed on a Layout (or LayoutModule or Layout Template), the Create and Check-In workflow dialogs list limited Editions items: only those Editions that are currently assigned to the Layout.

Inactive, but assigned Issues

Inactive Issues are not listed in the workflow dialogs, except for those Issues (and their Editions) that are currently assigned to the object. The user can deselect such inactive Issues (because they are assigned). However, the next time the workflow dialog raises, unassigned Issues that are inactive are no longer listed.

‘Change’ access rights not applied to Create and Copy To dialogs

The Access Profiles Maintenance page contains the following 'Change' access rights options:

- Change Brand/Issue/Category
- Change Edition
- Change Status

By toggling the Brand/Category/Status properties in the Create dialog, because of the access rights mentioned, those properties could 'suddenly' get disabled. There would be

no escape from that point. To avoid such deadlocks, the Brand/Issue/Category/Edition/Status properties are enabled for the Create dialog (except in case there are other rules telling to disable the properties, such as Create Article on a Layout Module for which they should be disabled). In short, the Create dialog ignores the 'Change' access options.

Property inheritance for Create and Copy To dialogs

When objects are about to be copied, or created from a template or directly created onto a parent layout, properties are inherited (from the source/template/parent object) and pre-filled in the dialog. This is done for all properties, except for the following:

- ID
- Document ID
- Name, Type
- Content Source
- Deadline
- Urgency
- Creator
- Created
- Modifier
- Modified
- Comment
- Status
- Route To
- Locked By
- Version
- Deadline Soft
- Rating

Those properties are cleared, and so empty properties are shown in the dialog. Nevertheless, some of those properties are automatically re-determined, such as: Name, Type, Status, Route To and Version. Those are pre-filled (with possibly a different value than the source object).

GetDialog2 service [since 8.0]

Up to and including v7, InDesign, InCopy and Content Station call the GetDialog service each time when user changes a property value (dialog field) for which the 'Refresh' flag is enabled. The request contains the new user typed value. That way, a custom server plug-in is able to act on the change.

Since v8, there is a new service, called GetDialog2, which supersedes the GetDialog service. Its request and the response are simplified to make it easier for clients, especially when it comes to refreshing dialogs. Basically it does the same, but instead of separate parameters, structured data trees are round-tripped through MetaData and Targets. This takes away the need to cache user typed data, which is quite complicated to merge on the way back. This makes clients rely even more on the server behavior, which results into more consistent behavior between clients.

At least for the 8.x versions, the GetDialog is still supported to allow clients to migrate to GetDialog2. Clients are encouraged to do so.

This is where clients pick-up data from server to show at dialog:

```
GetDialog2Response
        Dialog
                ...
                MetaData
                        MetaDataValue
        ...
        Targets
                Target
```

When redrawing the dialog, the user typed data is round-tripped through the new parameters as follows:

```
GetDialog2
        ...
        MetaData
                MetaDataValue
        Targets
                Target
```

For client convenience, the MetaData tree is not used (with BasicMetaData, etc), but instead, the list structure is used (with MetaDataValue). Custom properties are supported and are prefixed with "C_".

In v7, the following properties are round-tripped through a fixed set of GetDialog service parameters:

```
GetDialog
        ...
        ID
        Publication
        Issue
        Category
        Status
        Type
        ...
```

There is one exceptional field; The RouteTo field should respect the Brand's Routing configuration which is taken care of by the core server. The round-trips caused by the Refresh fields should not disturb this existing feature.

To optimize data traffic and server-side execution speed, clients can request for less data, in case they don't need that much. In v7, this is done as follows:

```
<GetDialog>
        ...
        <RequestDialog>true</RequestDialog>
        <RequestPublication>true</RequestPublication>
        <RequestMetaData>true</RequestMetaData>
        <RequestStates>true</RequestStates>
        <RequestTargets>true</RequestTargets>
        ...
```

and as a result, the following data structure is filled in (or not):

```
GetDialogResponse
        Dialog
        Publications
        PublicationInfo
        MetaData
        GetStatesResponse
        Targets
        RelatedTargets
        Dossiers
```

Since v8, the response structure has changed into this:

```
GetDialog2Response
        MetaData
        Targets
        RelatedTargets
```

As you can see, there are much less data elements returned, since that data is now moved to the MetaDataValue structure with id-name pairs. And so, there is no longer need to have filtering parameters, which are therefore not present at the GetDialog2 request.

Show display names for internal values [since 8.0]

For configured data structures, such as Brands, Issues, etc, the client and server communicate DB ids, while showing display names to end users. To do such, both sides need to know about the configured data structures, which is very dedicated and therefore limited to built-in (known) properties. In v7, there is no common structure that can hold both internal values and display names. With v8 there is, and so custom properties can use those, but also the built-in properties can be expressed in a more common way. This takes out some client logics, which makes things less dependent and specific.

The structure that holds the values, is defined as follows, whereby v8 changes are in bold:

```
<complexType name="MetaDataValue">
	<all>
		<element name="Property" type="xsd:string"/>
		<element name="Values" type="tns:ArrayOfString"
			minOccurs="0" maxOccurs="1"/>
		<element name="PropertyValues"
			type="tns:ArrayOfPropertyValue"
			minOccurs="0" maxOccurs="1"/>
	</all>
</complexType>
```

The following definitions are added since v8 as well:

```
<complexType name="PropertyValue">
	<all>
		<element name="Value" type="xsd:string"/>
		<element name="Display" type="xsd:string"
			minOccurs="0" maxOccurs="1"/> <!-- optional -->
	</all>
</complexType>
<complexType name="ArrayOfPropertyValue">
	<complexContent>
		<restriction base="soap-enc:Array">
			<attribute ref="soap-enc:arrayType"
			wsdl:arrayType="tns:PropertyValue[]"/>
		</restriction>
	</complexContent>
</complexType>
```

v7 example of usage, which is dis-encouraged:

```
<MetaDataValue>
	<Property>Publication</Property>
	<Values>
		<String>1</String>
	</Values>
</MetaDataValue>
```

v8 example of usage:

```
<MetaDataValue>
	<Property>Publication</Property>
	<PropertyValues>
		<PropertyValue>
			<Value>1</Value>
			<Display>WW News</Display>
		</PropertyValue>
	</PropertyValues>
</MetaDataValue>
```

Sending values and display names (like above) are applied to Publications, PubChannels, Issues, Categories, Editions, Users and UserGroups.

Refresh dialog fields [since 8.0]

Workflow dialogs guide users through their workflow. Even though dialogs are already made very flexible, Enterprise 8 takes another step; It enables application engineers to redraw the dialog in specific cases for which some fields needs to be ‘refreshed’.

For example, a customer might wish that, when an end-user tags a checkbox ‘Department’ on a workflow dialog, the listed items at the Route To field gets filtered automatically, showing only the users and groups working for the same department as the current user.

Or the other way around, when the Route To field is changed by the end user, the customer might want to have a readonly ‘Department’ text field to be filled in automatically, with the value configured for the selected user.

There could also be a need to let one field depend on two others, and those fields do not necessarily have to be custom. For example, when the Issue field is set to a print channel and the Edition is set to ‘north’ only, the customer might wish to enter a simplified workflow and so less statuses needs to be shown at the Status field.

The triggers can be both ways, or relate one-to-many and/or custom fields or built-in fields can be involved. The rules and dependencies can be developed through a custom server plug-in. Admin users can not interfere with such complicated rules other than enabling or disabling the whole feature through the Server Plug-ins admin page.

A new flag added to the workflow WSDL, named ‘Refresh’:

```
<complexType name="PropertyUsage">
	<all>
		<element name="Name" type="xsd:string"/>
		<element name="Editable" type="xsd:boolean"/>
		<element name="Mandatory" type="xsd:boolean"/>
		<element name="Restricted" type="xsd:boolean"/>
		<element name="Refresh" type="xsd:boolean"/>
	</all>
</complexType>
```

This indicates that when the field is changed by the end-user, the GetDialog service needs to be called by client applications to redraw the dialog. The core server sets this flag for the Brand, overrule Issue and Status fields, which represents the current behavior.

Multiple objects support [since 9.2]

Since Enterprise 9.2, a new parameter 'MultipleObjects' is added to the request. This boolean field can be used to specify whether the dialog is being drawn for a single object (false or nil) or for multiple objects (true).

For backwards compatibility with older clients, the parameter is optional. A request without this parameter is interpreted as a Set Properties operation for a single object.

Based on the parameter value, advanced business rules may apply when determining the dialog options, only returning those properties that apply when handling multiple objects.