HP TRIM

Software Version: 7.1

Web Service Reference Manual

Document Release Date: July 2011

Software Release Date: July 2011

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

Trademark Notices

Microsoft®, Windows®, Windows® XP and Windows Vista® are U.S. registered trademarks of Microsoft Corporation.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign-in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Web site at:

http://www.hp.com/go/hpsoftwaresupport

This Web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support Web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Contents

1 Which SDK?

Introduction

The HP TRIM Web Service is for customers who need to provide access to HP TRIM using HTTP through port 80.

After installation, you start your own development work based on the HP TRIM Web Service using the HP TRIM Software Development Kit (SDK).

This chapter clarifies information about SDK versions and helps you decide on a version of the HP TRIM SDK before starting development work.

SDK versions

Versions of the HP TRIM SDK currently available are the COM SDK and the .NET SDK:

COM SDK

This is the legacy HP TRIM SDK and is the only one which has been available for the entire life of HP TRIM 5 and 6.

There is a significant amount of code developed using this SDK version, both internally within HP and with external customers. This SDK makes a series of COM objects available to COM capable platforms and languages – in other words, Microsoft Windows centric development environments.

For development work against TRIM 5 or 6, you should still use the COM SDK

However, with the arrival of HP TRIM 7, HP Software will not develop the COM SDK any further.

.NET SDK

This is the new HP TRIM 7 SDK option.

You should use it for new development work against HP TRIM 7 or later.

Making the selection

What HP TRIM SDK version is right for a given task? The following points will guide you through the decision making process.

You have existing code using the HP TRIM COM SDK

HP will continue to support the COM SDK. Therefore, old code will not break; however, all new code should be written against the .NET SDK.

Not all new HP TRIM features will support the COM SDK; therefore, in some cases, you have to rewrite COM SDK code to use the .NET SDK.

You are using a language which cannot access .NET but can access COM objects and HP TRIM is installed on the same computer as the application will run on

Use the COM SDK version. The COM SDK provides a richer more interactive interface than the Web Service and performs well when installed on the same computer as the SDK application.

You are writing new code with a .NET language and have HP TRIM installed on the same computer as your application will run on

Use the .NET SDK.

You are using a language which supports Web Services based on HTTP, SOAP and XML and your application will run on a computer without HP TRIM installed on it

Use the HP TRIM Web Service, as it is optimized for remote network use.

You are using a language which does not support COM or .NET, but which is capable of using Web Services based on HTTP, SOAP, and XML

Use the HP TRIM Web Service.

Your language does not support COM, .NET or Web Services

Unfortunately, your chosen language is not capable of using the HP TRIM SDK.

Conclusion

Hopefully, this chapter has helped you decide which HP TRIM SDK is right for you. The rest of this document assumes that you make a decision to use the HP TRIM Web Service and documents how to use the Web Service provided by the toolkit.

2 Installation

Introduction

This chapter discusses the installation and configuration of the HP TRIM Web Service. It starts by discussing how to run the MSI installer and the supported platform for the HP TRIM Web Service. It then discusses manual deployment of the HP TRIM Web Service, before finishing up with answers to commonly asked questions.

Method

The default installation method installation for the HP TRIM Web Service is to use the MSI installation file provided by HP Software.

Prerequisites

See **TRIM7.1_Spec.pdf** for detailed requirements for the HP TRIM Web Service.

If the computer you are using has never run an ASP.NET v2 application, you may have to allow **ASP.NET v2.0.50727 Web Service Extensions**:

1. Open **Computer Management**
2. Expand **Internet Information Services (IIS) Manager**
3. Select **Web Service Extensions**
4. If the **ASP. NET v2.0.50727 status** is not set to **Allowed**, right-click it and select **Allow**.

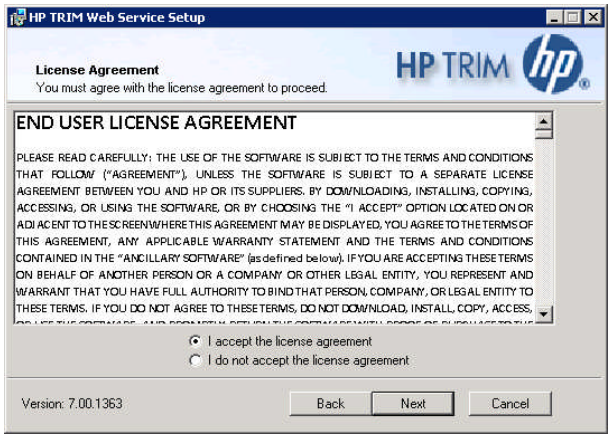
Installation process

1 Run the MSI installation file **HPTRM_WebService_xNN.msi**.

The Welcome screen appears:

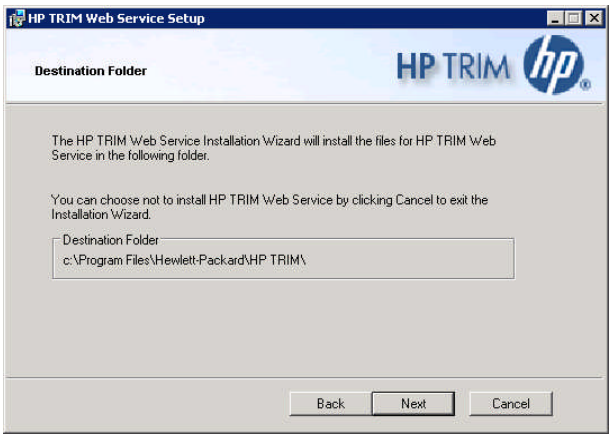
2 Click **Next**.

The **License Agreement** dialog box appears:



3 Select **I accept the license agreement** for the installer to continue and click **Next**.

The **Destination Folder** dialog box appears:

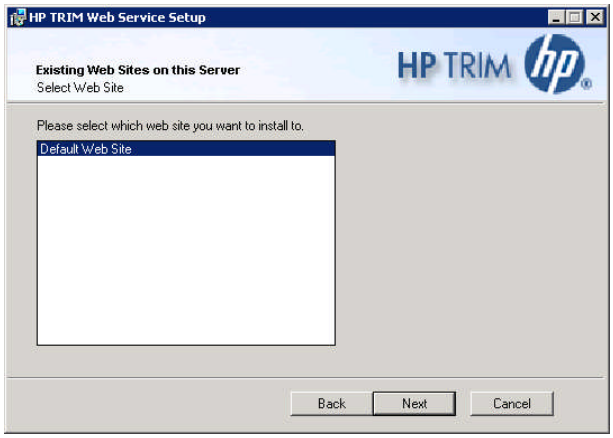


4 Click **Browse** to change the installation folder.

The folder specified here is generally the location of the HP TRIM installation.
This location will be used as the parent directory for the virtual directory that the HP TRIM Web Service installer will create.
This virtual directory is configured within Microsoft IIS and is where the HP TRIM Web Service binaries reside.

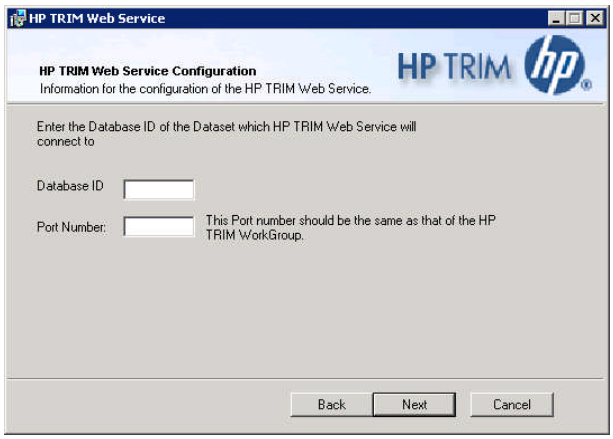
5 In the **Destination Folder** screen, click **Next**.

The **Existing Web Sites on this Server** dialog appears:



6 Select the Web site to install to and click **Next**.

The **HP TRIM Web Service Configuration** dialog box appears:

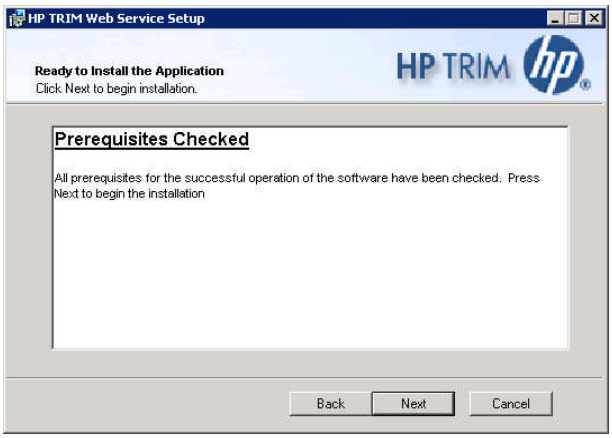


7 Type the details of the HP TRIM dataset that you want the HP TRIM Web Service to use.

The HP TRIM Web Service can only expose one dataset at a time.
If you want to support more than one HP TRIM dataset through the HP TRIM Web Service, then you need to install two copies of the HP TRIM Web Service using the manual deployment technique discussed later in this chapter.

8 Click **Next**.

The **Ready to Install the Application** dialog box appears:

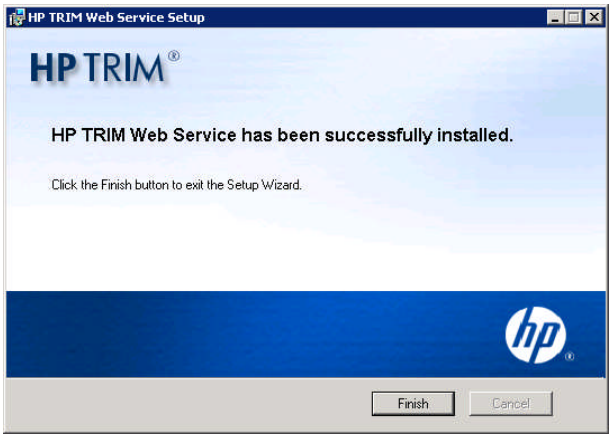


It displays the results of the prerequisites check.

The installer is now ready to run.

9 In the **Ready to Install the Application** screen, click **Next**.

When complete, the **HP TRIM Web Service has been successfully installed** dialog box appears:



Manual deployment

It is possible to deploy the HP TRIM Web Service without running the MSI installation file.

This is useful when the HP TRIM Web Service is already installed on a computer and you want to run a second instance of the toolkit on that computer, perhaps pointing to a different HP TRIM dataset.
It is also useful for deploying the HP TRIM Web Service to a large number of computers in a clustered Web environment.
The HP TRIM Web Service runs inside a virtual directory managed by Microsoft's IIS. This virtual directory needs to have some specific permissions configured, which are usually set by the .msi installation file.

Note

In IIS 7, an IIS application, not virtual directory, is required. For simplicity, this document will sometimes use the term **virtual directory** to refer to what is a virtual directory in IIS 6 and an application in IIS 7.

Deploying the HP TRIM Web Service manually

1. If this is a second Web service installation for the purpose of connecting to a different HP TRIM database than the original Web service, then create a copy of the Web service installation folder (e.g. **C:\Program Files\Hewlett-Packard\HP TRIM\TRIMWS** to **C:\Program Files\Hewlett-Packard\HP TRIM\TRIMWS-MyDbId**
2. Start IIS Manager
3. Create a new virtual directory (application in IIS 7)
4. Give it an alias
5. Enter the location where the HP TRIM Web Service is installed in the path – see step 1
6. In IIS 6, ensure that the virtual directory has the permissions **Read** and **Run Scripts (such as ASP)**
7. Set up authentication so that **Anonymous** access is unavailable and Windows Authentication is selected.

Note

We recommend that you do not enable anonymous access to the HP TRIM Web Service, as this would leave the toolkit unable to determine which user is connecting to HP TRIM. Effectively, all connections will appear to HP TRIM to be coming from the user configured in this section, if it is enabled.

We recommend that under **Authenticated access**, you select:

- **Integrated Windows authentication**
- **Basic authentication (password is sent in clear text)**

For more information, contact an IIS administration specialist.

Folder access

Grant the user group **Write** access permission to the folders:

- **C:\HPTRIM\WebServerWorkPath**
- **C:\Windows\Temp**

The configuration file

The configuration for the HP TRIM Web Service is stored in the file **web.config** in the HP TRIM Web Service directory.

There is a lot of configuration information in this file related to the ASP.NET v2 runtime environment and outside the scope of this document; but there are some values which are relevant to a deployment of the HP TRIM Web Service.

Here is an example of the relevant section of the configuration file:

```
<appSettings>
        <add key="dbid" value="IT" />
        <add key="transferTempPath" value="%SYSTEMP%" />
        <add key="traceMode" value="true" />
        <add key="readOnly" value="false" />
        <add key="logName" value="" />
</appSettings>
```

The applications setting tag is where the configuration for the HP TRIM Web Service resides. The keys are described in the following table:

Key	Meaning	Comments
dbid	The HP TRIM dataset to connect to	
transferTempPath	The location to store temporary files in	The special %SYSTEMP% resolves to the Microsoft Windows temporary directory. Every HP TRIM user using the Web Service needs write access to %SYSTEMP%. They also need access when you have changed this key value as the Web Service continues to use %SYSTEMP% for some operations. %SYSTEMP% is a Windows mapping to the system temporary folder, usually C:\Windows\Temp, a folder on the server that the Web Service is installed on.

traceMode	Turn on the active logging mode	This mode logs all events to occur in the toolkit, even if they are not errors. It helps HP Software support staff determine what is occurring in the toolkit for field support, but writes large amounts of information to the Windows Event Log.
logName	The name of the Windows event log to write to	An empty string means to use the default HP TRIM Web Service log source created by the .msi installation file. Another common option is **Application**, which writes to the Windows application event log source.
readOnly	If true, no updates can occur	This will force an error to be returned if the caller attempts to update any information in HP TRIM through this instance of the HP TRIM Web Service

Conclusion

In this chapter, we have discussed how to install the HP TRIM Web Service as well as the various configuration options.

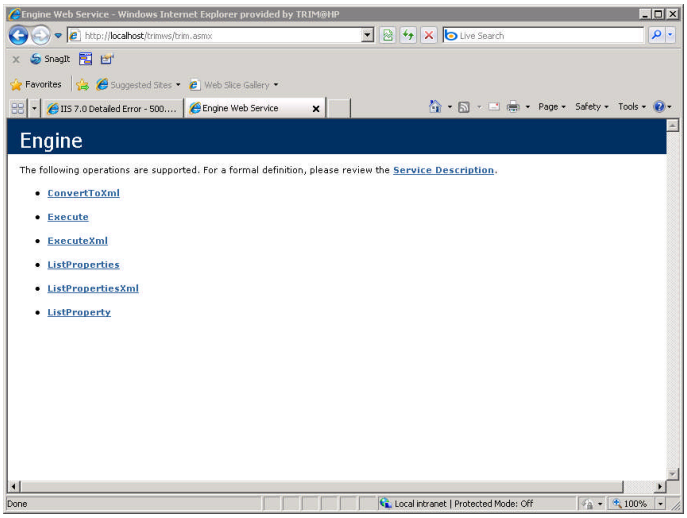
3 Developing

Getting started

1 The first thing to do before beginning development against the HP TRIM Web Service is to make sure you can access the file **trim.asmx**.

This can be done by pointing any Web browser at the host computer and supplying authentication details as needed.
These details should match the LAN logon for the user configured in HP TRIM.

Once authenticated, you should see a page like this:



It is worth mentioning that you do not have to develop on the computer that is running the HP TRIM Web Service. The server computer may be on the other side of the room, across the world or exist only as a virtual machine.

IDEs

For this guide, we are presuming that you are programming in Microsoft Visual Studio .NET. To describe its objects, the HP TRIM Web Service uses the Web Services Description Language (WSDL).

Plenty of other development IDEs support WSDL and will assist the developer in writing code, but will interpret WSDL differently.

This could mean that your client side objects may behave a little differently than described here. If you are having real trouble, contact the API support team at HP Software – trimsdk@hp.com
See Contacting Support.

To develop against the HP TRIM Web Service in Visual Studio, you need to install:

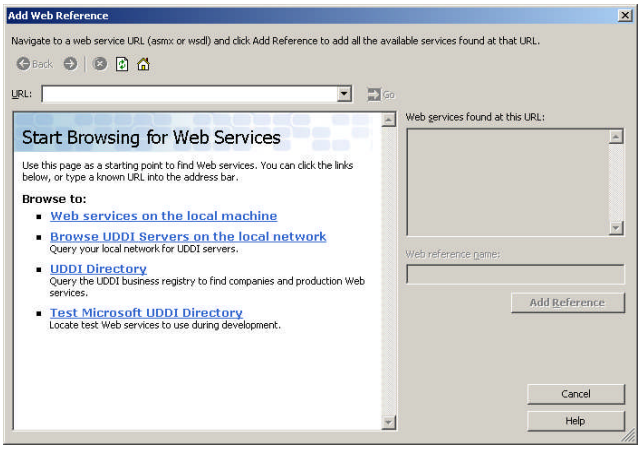
- .NET runtime version 1.1
- Web Service Extensions 3.0 (this is installed by the HP TRIM Web Service installer)

If you are not using Visual Studio, you will need the equivalents for your chosen IDE.

When using Java for development, it is recommended you use JAX-WS.

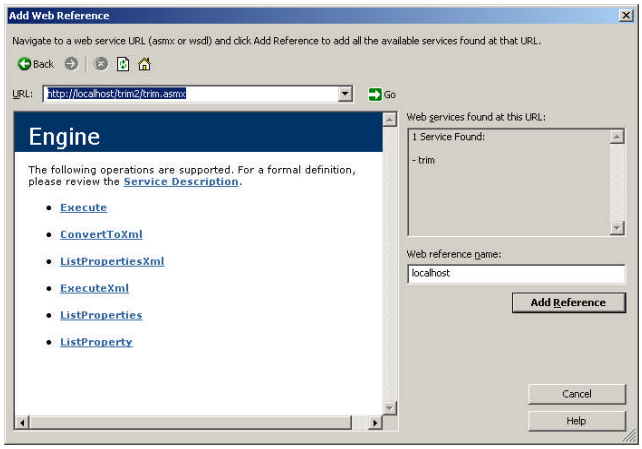
Creating a reference to the HP TRIM Web Service

1 From the **Project** menu, select **Add Web Reference**.
The **Add Web Reference** dialog appears.



2 **URL** - type the URL of your configured Web service, click the **arrow** button.
It is likely that a logon dialog prompts you for your user name and password.

Once you have been authenticated, the HP TRIM Web Service appears.



Note the word **Engine** here – the main object in the HP TRIM Web Service, which we will discuss more soon.

3 For now, click the **Add Reference** button to get Visual Studio to read the WSDL and create objects on your computer to help call the HP TRIM Web Service.

4 Developers should name of the Web reference something more meaningful than 'localhost' as above.

4 Basic searching

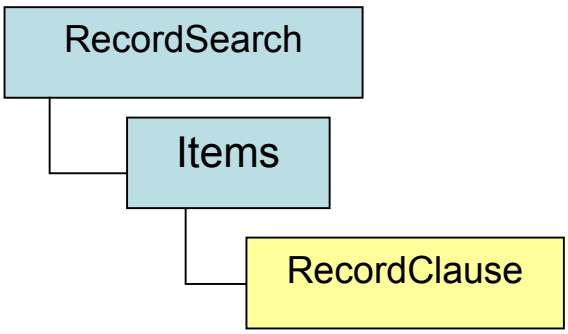
Introduction

Search is the most fundamental operation that any HP TRIM API can provide.

Due to the request/response nature of the HP TRIM Web Service architecture, every call to the Web Service is stateless. This means the HP TRIM Web Service will never remember what a client application was looking at previously. Almost any tasks done using the HP TRIM Web Service will, by necessity, require a search operation.

The RecordSearch operation

The RecordSearch operation exposes the HP TRIM record searching model. HP TRIM has a powerful and flexible model for searching for records, and in the HP TRIM Web Service, this is represented as a RecordSearch operation. The RecordSearch operation has no methods itself, just a collection of items that represent the search clauses.



You can have as many RecordClauses as you want

To conduct a search, the process is to:

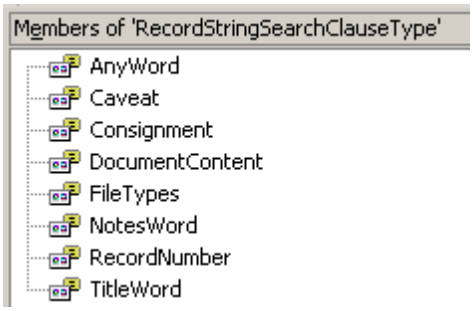
1. Create a RecordSearch operation
2. Create one or more RecordSearch clauses
3. Append those clauses to the RecordSearch object
4. Once that has been completed, append the search to the TrimRequest
5. Use the engine object to execute that request

About clauses

When you view the object browser in your IDE, you can see that the vast majority of objects created for the HP TRIM Web Service are RecordSearchClause objects.
These represent the different kinds of search you can do, and they differ in the number and types of parameter that they take to perform a search.
Most of these objects contain a property called “type” which allows you to further specify the kind of search you want to do.
A typical example, one using text search:

```
RecordStringSearchClause clause = new RecordStringSearchClause();
clause.Type = RecordStringSearchClauseType.TitleWord;
clause.Arg = "reef";
```

The “type” of property on RecordStringSearchClause takes a value from an enumeration of RecordStringSearchClauseTypes.
If we look at the enumeration in the object browser, we can see:



Generally speaking, if a clause has a type property, it will correspond to an enumeration which lists all the search clauses of that type.
There are exceptions – for instance, the RecordClassificationSearchClause does not have a type. Instead, it has a ClassificationUri property where you pass in the URI of the Classification you are looking for.
This is because there is only one record clause which takes a classification URI.

Example

Here is an example of setting up a basic search operation.
As with any HP TRIM Web Service operation, the search consists of a TrimRequest object, which has one or more operations.
The request is then passed through to the engine object, which returns a TrimResponse.

Example – Title Word search

```
// Construct a request object
TrimRequest request = new TrimRequest();

// Construct a RecordStringSearchClause, with type
// TitleWord, and argument "reef"
RecordStringSearchClause clause = new RecordStringSearchClause();
clause.Type = RecordStringSearchClauseType.TitleWord;
clause.Arg = "reef";

// Construct a record search, and put our search clause in it
RecordSearch search = new RecordSearch();
search.Items = new RecordClause[] { clause };
// If we had more than one clause, it would look like:
// search.Items = new RecordClause[] { clause1, clause2, clause3 }

// Put our search operation into our TrimRequest
request.Items = new Operation[] { search };

// Send it off. Whatever comes back will be in response
Engine engine = new Engine();
engine.Credentials = new System.Net.NetworkCredential(username, password);
TrimResponse response = engine.Execute(request);
```

In this example, the final line that executes the request would return two result objects – a SuccessResult, indicating that the search completed successfully, and an EndResult.

If an operation executes successfully, the Web Service will return you a SuccessResult.
If the operation fails, the Web Service will return an ErrorResult with an error message.

There will be a SuccessResult or ErrorResult for each operation you execute.

An EndResult indicates that the Web Service has dealt with everything, and has now forgotten about you.
There will always be a single EndResult.

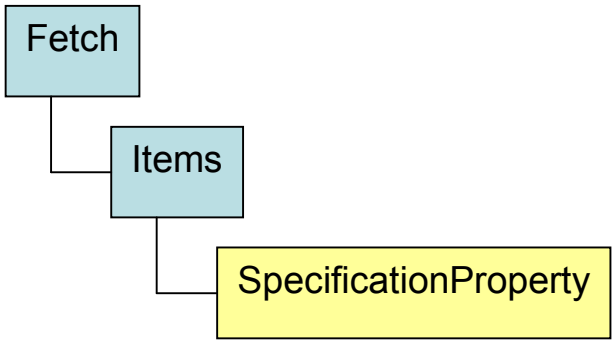
Getting results

In the previous example, we did not enquire about any of the properties of the records, the HP TRIM Web Service conducted a search, noted the successful completion of the search, and finished. This may seem ultra-pedantic, but the reality is that HP TRIM records can contain an enormous amount of data.

If you consider the relationship between records, Locations and other records, which are properties of a single record, the possibility for HP TRIM to get locked in a recursive death-spiral if it returned everything it knew about a record in the results is quite high.
To alleviate this, the developers have left it entirely up to you.
Additionally, only returning the data that you need is the most effective way of using network resources.

The Fetch operation

The Fetch operation is the way of telling the HP TRIM Web Service what information to return to the client application.
It is a comprehensive series of properties that can be used to obtain any information stored in HP TRIM about a record.



You can have as many SpecificationProperties as you want.

The Fetch operation is essentially a container for a collection of SpecificationProperty objects – for each piece of information you want to return from the Web Service, you need to supply a SpecificationProperty.

The SpecificationProperty contains several properties that determine the manner in which information is returned to you (more about those later); but probably the most important property to set when creating a SpecificationProperty is the Name property that uniquely identifies the value of the metadata you want to return.

Note that a Fetch is only useful when combined in a TrimRequest with a search operation.

Example 1

```
SpecificationProperty rectitleSpec = new SpecificationProperty();
rectitleSpec.Name = "rectitle";
```

This example tells the HP TRIM Web Service that we are looking for the title of the record to be present in the result set.

To complete this, we need to append the SpecificationProperty to the Fetch operation:

```
Fetch fetch = new Fetch();
fetch.Items = new SpecificationProperty[] { rectitleSpec };
```

You can produce a sample that will return the title of each record found by combining these code snippets with the first example by including the SpecificationProperty and adding the Fetch operation to the TrimRequest.

Example 2

A search returning record titles

```
// Construct a request object
TrimRequest request = new TrimRequest();

// Construct a RecordStringSearchClause, with type
// TitleWord, and argument "reef"
RecordStringSearchClause clause =
        new RecordStringSearchClause();
clause.Type = RecordStringSearchClauseType.TitleWord;
clause.Arg = "reef";

// Construct a record search, and put our search clause in it
RecordSearch search = new RecordSearch();
search.Items = new RecordClause[] { clause };

// This SpecificationProperty says we want the title
SpecificationProperty rectitleSpec = new SpecificationProperty();
rectitleSpec.Name = "rectitle";

// And this one says we want the date the record was Created
SpecificationProperty recdatecreatedSpec =
        new SpecificationProperty();
recdatecreatedSpec.Name = "recdatecreated";

// Create a new Fetch Operation, and add our SpecficationProperties
Fetch fetch = new Fetch();
fetch.Items = new SpecificationProperty[] { rectitleSpec };

// Put our search operation AND our fetch operation into the
// TrimRequest
request.Items = new Operation[] { search, fetch };

// Send it off. Whatever comes back will be in response
Engine engine = new Engine();engine.Credentials = new System.Net.NetworkCredential(username,
password);
TrimResponse response = engine.Execute(request);
```

Information on SpecificationProperties other than recTitle can be found at the end of this chapter.

Obtaining results using the TrimResponse

The TrimResponse object contains a loosely federated collection of results – each one relating directly to an operation, and one that indicates the batch is complete.

For example, if your request contains two operations, you will get three results back – one for each operation, and one to mark that processing is complete.

The best way to process results is to iterate over the items returned and inspect each one:

Example 3

Iterating through TrimResponse items

```
foreach (Result result in response.Items)
{
        switch (result.GetType().Name.ToString())
        {
                case "EndResponse":
                        // Code to handle the EndResponse
                        textBox1.Text += "EndResponse!\r\n"
                        break;

                case "SuccessResult":
                        // Code to handle the SuccessResult
                        textBox1.Text += "SuccessResult!\r\n"
                        break;

                default:
                        // Code to handle whatever else, etc…
                        textBox1.Text += "Other Result!\r\n"
                        break;
        }
}
```

If you are preparing a large and complex request, you can tag each operation with an ID, which will be returned against the corresponding result.
The ID can be any string identifier you want. This is especially useful for an asynchronous request, where it will tell you which request has finished

Example 4

TrimResponse items with IDs

```
RecordSearch search = new RecordSearch();
search.Id = "TheSearch";
search.Items = new RecordClause[] { clause };

// Additional code

foreach (Result result in response.Items)
{
        switch (result.GetType().Name.ToString())
        {
        case "SuccessResult":
                        if (result.Id == "TheSearch")
                        {
                                textBox1.Text += "It was my search that worked!\r\n";
                        }
                        break;

                default:
                        // Code to handle whatever else, etc…
                        textBox1.Text += "Code to handle other stuff\r\n";
                        break;
        }
}
```

There are several different kinds of results, each with different behaviors.

For now, we will only concentrate on the more commonly encountered results – SuccessResult, EndResult, FetchResult and ErrorResult.

The SuccessResult

For most operations that do not actively return information, the SuccessResult is the HP TRIM Web Service's way of saying that everything was performed without any errors.
The SuccessResult has no other information about an operation that completed other than the ID that may have been assigned to the calling process.
It is generally good practice to label all your operations with IDs, and to check them when receiving the response. This ensures that you can accurately handle results and operations.

The ErrorResult

If an operation failed, then an ErrorResult is returned in the place of a SuccessResult.
The ErrorResult object contains properties that will give a unique error code and a string description of the error. The HP TRIM Web Service's ability to handle batch operations means that the failure of one operation automatically results in the failure of any operation after the failed operation in the request.

Example 5

Handling an ErrorResult

```
TrimResponse response = engine.Execute(request);
foreach (Result result in response.Items)
```

```
{
        switch (result.GetType().Name.ToString())
        {
                case "Error":
                        ErrorResult err = (ErrorResult) result;
                        if (result.Id == "TheSearch")
                        {
                                textBox1.Text += "Search operation Failed\r\n";
                                textBox1.Text += "Error Code: " + err.ErrorNumber.ToString();
                                textBox1.Text += "Error Message was " + err.Message;
                        }
                        break;
                default:
                        // Code to handle whatever else, etc…
                        break;
        }
}
```

The FetchResult

The FetchResult contains all the information that has been requested about the results of a search.

All this data can be found in the objects collection – a collection of TrimObjects which contain information about the object returned by the HP TRIM Web Service.

It contains a collection of objects that map directly to the SpecificationProperties passed in as part of a fetch operation.

Additionally, the FetchResult contains the count (the number of records returned by the search) and the ResultsTruncated properties.

If your search exceeded the limit specified in the original Fetch specification, this Boolean will be set to true and any result after the limit specified in the Fetch will be missing.

Let's have a look at the TrimObject objects that make up the FetchResult. We can see that it is made up of a collection of Value objects, which contain the data asked for with the SpecificationProperties. It also contains the URI for the object in question, and information on its version.

Each value potentially also has another value collection associated with it in the children property. Iterating over the values collection will return to us the properties of our original SpecificationProperty objects attached to the original Fetch object.

The EndResponse

The EndResponse is simply confirmation that the HP TRIM Web Service has finished processing. An EndResponse will always be returned.

ListProperties

Now you know how to search for records and how to specify what it is that you want to see in your results.

The nature of the SpecificationProperty means that you need to specify the name of the property, but as we have previously discussed, there could be easily a thousand properties available at any time.

Originally, there was some discussion of perhaps shipping a book with the HP TRIM Web Service that listed all the properties, but as that was not exactly practical, there is a method included on the engine object called ListProperties.
To use it, you pass in the name of the object you are interested in, and it returns to you a list of all the properties available on the object.

Example 6

List Properties

```
textBox1.Text = "LIST PROPERTY\r\n";
textBox1.Text += "Building Engine\r\n";
Engine engine = new Engine();
engine.Credentials = new System.Net.NetworkCredential(username, password);

PropertyDescription[] descs = engine.ListProperties("classification");

foreach (PropertyDescription desc in descs)
{
textBox1.Text += desc.Name + " (" + desc.Caption + "): \r\n";
}
```

Note that this method is provided solely as reference material for developers. Providing a list of all the SpecificationProperties that exist for a record is of very limited use for an end user.

Other SpecificationProperty properties

Besides the name property, the SpecificationProperty object includes two additional properties, both of which deal with the fact that properties on HP TRIM objects can be objects themselves.
These two properties are the Children Collection and the ForView Property. ForView relates to a visible version of the string, instead of a parsable one (e.g. 32000 vs 32mbs).

Conclusion

A search through the Web Service consists of two components:

The **Search** object contains the information needed to find the object or objects in HP TRIM

The **Fetch** object contains one or more SpecificationProperties objects, each of which indicates a property of the object - or objects - to be returned

After executing a search, the TrimResponse object that is returned contains one or more **Result** objects:

- A SuccessResult indicates that the operation executed without errors
- An ErrorResult contains the details of why an operation failed

- A FetchResult consists of the values matching the SpecificationProperties passed in through the Fetch object
- Finally, an EndResult indicates the last Result object

5 Primary operations

Introduction

There are several generic operations in the HP TRIM Web Service – operations that are used the same way across different objects.

Searching is covered in depth in its own chapters, *Basic Searching* and *Advanced Searching*.

The operations covered in this chapter include preparing an object to be modified, creating a new object and deleting an existing object.

IsForUpdate

The IsForUpdate property on searches is a way of specifying if the results of a search are to be modified or not.

It was implemented because - as you will see later in this chapter - it is trivial to accidentally modify or delete a large number of objects in one go.

Example 1

Setting IsForUpdate

```
RecordStringSearchClause rss = new RecordStringSearchClause();
rss.Arg = "Reef";
rss.Type = RecordStringSearchClauseType.TitleWord;

RecordSearch search = new RecordSearch();
search.Items = new RecordClause[] {rss};
search.IsForUpdate = true;
```

Whenever an object is to be altered, the IsForUpdate property on the search that locates the object must be set to True.
If it is set to False (as it is by default), any following update or delete operation will fail.

SpecificationProperty children

Some properties on an object are objects themselves.

For example, the author of a record is actually a Location object.

A SpecificationProperty that is asked to return an object will return the URI of that object which can then be used in other searches or in an injector – see *Injection*.

Sometimes, the properties of the returned object will need to be accessed directly. That is what the idea of SpecificationProperty children is for.
Each SpecificationProperty is capable of taking a collection of SpecificationProperties.

This collection is applied to the object found by the parent property.

Example 2

Using SpecificationProperty

```
// Get the Location we want to look at
SpecificationProperty rectitleSpec = new SpecificationProperty();
rectitleSpec.Name = "recAuthorLoc";

// Get the property of the Location
SpecificationProperty locname = new SpecificationProperty();
locname.Name = "locSurname";

rectitleSpec.Children = new PropertyBase[] {locname};

// Create a new Fetch Operation, and add our
// SpecficationProperties
Fetch fetch = new Fetch();
fetch.Items = new SpecificationProperty[] { rectitleSpec};

// Put our search operation AND our fetch operation into the
// TrimRequest
TrimRequest req = new TrimRequest();
req.Items = new Operation[] { search,fetch };

TrimResponse resp = engine.Execute(req);
```

The above example will return the URI of the author Location, along with the surname of the Location.

SpecificationProperty children can be nested – so it would be possible to find the manager of the author in the example, and the manager's organization, and so on.

InputProperties

InputProperties are virtually identical to SpecificationProperties, but go the other way. While you use a SpecificationProperties to find data on an object, you use InputProperties to manipulate or add that data.
InputProperties have the same names (and so can be found the same way) as SpecificationProperties – using ListProperties.

Example 5

Creating an InputProperty

```
InputProperty title = new InputProperty();
title.Name = "recTitle";
title.Val = "New Record";
```

An InputProperty has a Name, which uses a SpecificationProperty name (so recTitle, locSurname, rtyName, etc), and a Val, which is the value you are suggesting for that property.

Modifying objects

Updating properties is the simplest way to modify the metadata of a record.

You simply assign a new value of the correct data type to the named property of the object. Field-level verification is carried out, and an error will be returned if the modification you attempted violated some of the HP TRIM business rules.

Modifying metadata with the Update operation

The simplest way to update data in a HP TRIM record is to modify the named properties on the object.

Obviously, you can only do so on properties that are not marked as read-only. This includes most of the date properties, certain Location properties (AuthorLoc, AddresseLoc and OtherLoc) and miscellaneous properties such as **External ID**, **Priority**, **Accession Number** and **Foreign Barcode** for the record object.

Assuming the property can be updated, it is the **Update** object that does the work.
An Update object takes in an array of InputProperties which it will apply to the object or objects in the search immediately preceding it.
These InputProperties are made up of two important values:

- the name of the property you are modifying – so “recTitle” or “locSurname”
- the value of the new property – so “Updated Timesheet” or “Smith”

Example 3

Updating metadata

```
Engine engine = new Engine();
engine.Credentials = new System.Net.NetworkCredential(username, password);
```

```
TrimRequest request = new TrimRequest();

RecordUrisSearchClause clause = new RecordUrisSearchClause();
clause.Uris = new string[] { "313" };
RecordSearch search = new RecordSearch();
search.IsForUpdate = true;
search.Items = new RecordClause[] { clause };

InputProperty rectitleInput = new InputProperty();
rectitleInput.Name = "rectitle";
rectitleInput.Val = "The New Title Is This!";

Update update = new Update();
update.Items = new InputProperty[] {rectitleInput};

request.Items = new Operation[] { search, update };

TrimResponse response = engine.Execute(request);

foreach (Result result in response.Items)
{
        switch(result.GetType().Name.ToString())
        {
                case "FetchResult":
                        FetchResult fetchResult = (FetchResult) result;
                        foreach (TrimObject obj in fetchResult.Objects)
                        {
                                foreach (Value property in obj.Values)
                                {
                                        txtOutput += "Got " + property.Name + ":";
                                        txtOutput += property.Val;
                                }
                        }
                        break;
                }
        }
}
```

Will it save? Verifying records

There are two operations to assist in making sure changes are valid:

- Verify
- VerifyCreate

These work like verifying a record prior to saving it in the COM SDK.

If you make an attempt to update a record that fails, the error returned by the HP TRIM Web Service will give details on why the operation failed – for example, **a value for a required field was not specified on a Create operation**.

Additional Fields

HP TRIM allows a large number of user-defined Additional Fields to be assigned to records and Locations.

Due to the loosely bound approach to updating properties that the HP TRIM Web Service takes, the act of updating Additional Fields is precisely the same as for updating a native HP TRIM metadata field:

1. Create an InputProperty
2. Assign it to the Additional Field
3. Give it the required value

If the attempt to update the field fails - for example, because the business rules limit the field to a particular series of lookup values and the value supplied is outside of those - the HP TRIM Web Service will return an error result for the update.

All Additional Fields are described in SpecificationProperties and InputProperties with the prefix **udf:**, followed by the Additional Field title.

Example 4 – specifying an Additional Field

For example, an Additional Field Current License, the code would look like

```
SpecificationProperty driver = new SpecificationProperty();
driver.Name = "udf:Current License";
```

A list of Additional Fields is returned along with the other SpecificationProperties when the ListProperties method is executed on an object.

Create

Being able to create new objects is vital to most HP TRIM setups.

During the development stage of the original HP TRIM Web Service, there was a Create object for each possible object. As each object requires a different minimum set of parameters to be created, this seemed like a good approach. However, it proved too unwieldy to implement and did not scale well to future development.

Instead, a single Create object has been implemented to cater for all objects. This Create operation is really just a special type of Update operation.

It is worth noting that since you are dealing with a new object, not one found through a search, there is no IsForUpdate property to set.

Create.Items

The Create object takes an array of InputProperties.
As mentioned previously, each object has a subset of properties that must be set in order for the Create operation to be successful.
This array of InputProperties can set every property on the object – but it must set that minimum subset.

Example 6 – creating a Location

```
//Build input properties
InputProperty lastname = new InputProperty();
lastname.Name = "locsurname";
lastname.Val = "Able";

//Build the CreateLocation object
Create create = new Create();
create.TrimObjectType = "location";
create.Items = new InputProperty[] {lastname};

//Build the request
request.Items = new Operation[] {create};
```

Errors

If the minimum subset for the object is not specified, then the operation will fail and an ErrorResult will be returned.
The ErrorResult will specify the property it failed on, but will only indicate one property. If multiple properties are missing, it will only report on the first it encountered.

Example 7 – failed Create error message

Could not save changes to (empty) with identifier 0: Please enter a name for this Location.

(TRIM Web Service unique id = 1000237)

Unfortunately, there is no way of knowing for certain what properties are needed to successfully create an object. Referring to the SDK documentation may be of assistance, but you may need to resort to trial and error.

Delete

Using a Delete object is fairly simple – create the object and pass it along after a search. It is important to remember that a delete is very powerful; the HP TRIM Web Service bypasses many of the checks done in the normal HP TRIM interface. It is assumed that the developer will implement similar safeguards to help protect users.

Example 8 – deleting a record

```
ShortcutRecordUri sru = new ShortcutRecordUri();
sru.Uri = "553";
sru.IsForUpdate = true;

Delete delete = new Delete();

TrimRequest request = new TrimRequest();
request.Items = new Operation[] {sru, delete};
```

As mentioned previously, the Delete operation was the main reason for the implementation of the IsForUpdate property. A Delete operation will not be successful if IsForUpdate on the associated Search object is not set to true.

Dates

The way that dates are handled in the HP TRIM Web Service has changed for the version released with HP TRIM 6. Dates are now formatted using the sdWebService string type from the HP TRIM COM SDK, which produces strings in this format:

Sun, 06 Nov 1994 08:49:37 GMT

The HP TRIM Web Service will always return the time in GMT. This complies with RFC 822 (used for SMTP and HTTP). You need to use this format when you pass strings into the HP TRIM Web Service as well. Most importantly, remember to include time zone information.

Conclusion

You should now have a good overview of the generic operations that can be performed by the HP TRIM Web Service. These operations include creating and deleting an object, how errors and dates are handled, and how to access Additional Fields.

6 Child Lists

Introduction

There are some objects within HP TRIM which can contain references to other objects. For example, there can be many Locations associated with a record object. Most SDK programmers will be familiar with the recAuthorLoc property for the Author Location for the record. This chapter introduces the mechanism underlying this shortcut property and documents a much more powerful method of accessing these associated objects.

Child objects

In the example above, the Locations associated with a record are called child Locations of that record.
Each record has a list of child objects, and the Locations will be referred to by objects of type recLocation in that list.
This is because it is possible that a single record will have more than one author.
To extract all of the authors of a record, an SDK application would iterate through the child list, looking for objects of type recLocation which are authors.

Using the HP TRIM Web Service to access these child list items is essentially the same as accessing any other property.

The biggest differences are:

- It is possible to have more than one property value returned for a single property request, for example, when there is more than one author
- You need to use conditional schematics to ensure that you only update one of these child list items at a time

Example 1 – fetching children

This code fetches a number of properties of a Location, including a child list item:

```
trimct.Engine engine = new trimct.Engine();
engine.Credentials = System.Net.CredentialCache.DefaultCredentials;

LocationUrisSelect lls = new LocationUrisSelect();
lls.Uris = new string[] {"1"};
lls.Limit = 1;

trimct.Fetch fth = new Fetch();
SpecificationProperty rts = new SpecificationProperty();
rts.Name = "locSurname";
SpecificationProperty rts2 = new SpecificationProperty();
rts2.Name = "locFullFormattedName";

SpecificationProperty rts3 = new SpecificationProperty();
rts3.Name = "locEmailAddress";
SpecificationProperty rts4 = new SpecificationProperty();
rts4.Name = "locGivenNames";
SpecificationProperty rts5 = new SpecificationProperty();
rts5.Name = "locGender";
SpecificationProperty rts6 = new SpecificationProperty();
rts6.Name = "locLocType";
SpecificationProperty rts7 = new SpecificationProperty();
rts7.Name = "child:locEAddress";

fth.Items = new SpecificationProperty[] {rts, rts2, rts3, rts4, rts5, rts6, rts7};

trimct.TrimRequest req = new trimct.TrimRequest();
req.HideVersionNumbers = true;
req.Items = new trimct.Operation[] {lls, fth};

textBox2.Text = engine.ConvertToXml(req);
// Execute the request
```

The SpecificationProperty with the name property set to **child:locEAddress** is a request for a child list item. All such requests will start with a **child:** in much the same way that requests for Additional Fields will start with **udf:** .

The return results will vary depending on how many locEAddresses exist in the child list for this Location. Here is an example in XML form:

Example 2 – XML response to example 1

```
<?xml version="1.0"?>
<TrimResponse xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SearchResult>
  <FoundCount>1</FoundCount>
 </SearchResult>
 <FetchResult ResultsTruncated="false" Count="1" TrimObjectType="location">
  <Objects>
   <TrimObject Uri="1" Version="2005-06-28T00:44:35+10:00">
    <Values>
     <Value Name="locSurname" Val="Skywalker" ErrorOccurred="false">
      <Children />
     </Value>
     <Value Name="locFullFormattedName" Val="Luke Skywalker" ErrorOccurred="false">
      <Children />
     </Value>
     <Value Name="locEmailAddress" Val="luke @gmail.com" ErrorOccurred="false">
      <Children />
     </Value>
     <Value Name="locGivenNames" Val="" ErrorOccurred="false">
      <Children />
     </Value>
     <Value Name="locGender" Val="0" ErrorOccurred="false">
      <Children />
     </Value>
     <Value Name="locLocType" Val="4" ErrorOccurred="false">
      <Children />
     </Value>
     <Value Name="child:locEAddress" Val="501" ErrorOccurred="false">
      <Children />
     </Value>
     <Value Name="child:locEAddress" Val="504" ErrorOccurred="false">
      <Children />
     </Value>
    </Values>
   </TrimObject>
  </Objects>
 </FetchResult>
 <EndResponse />
</TrimResponse>
```

The child list item being returned is the one with the name **child:locEAddress**. If there were more than one object to return, then the name will be repeated in the list of return results.

You can of course specify the sub-properties of a child list item, just as you can for any other property which is an object. Remember that the child list item is not the destination object type though, it has an intermediate type.

For example, a recLocation is not a Location, but it does point to a Location with other information as to the type of the linkage included as well.
For more information on accessing sub-properties, refer to the chapter *Primary Operations* earlier in this manual.

Changing just one child list item

Now that we have found a child list item, let's see how to change its properties. The change itself happens just like it does in the chapter 'Basic properties', but in this case, we need to specify which of the children to change. Otherwise, they will all be changed. This conditional behavior didn't exist in the first release of the HP TRIM Web Service for HP TRIM 5.2. This is by including an additional property to the InputProperty specifying the new value – the TargetChildUri. This value should be set to the URI of the child object to change.

Here is an example of how to do this:

Example 3 – changing a child list item

```
InputProperty childLocationInput = new InputProperty();
childLocationInput.Name = "rlcLocation";
childLocationInput.Val = "14";

InputProperty childRecLocationInput = new InputProperty();
childRecLocationInput.Name = "child:recLocation";
childRecLocationInput.Children = new PropertyBase[] {

childLocationInput };
childRecLocationInput.TargetChildUri = "221";

Update upd = new Update();
upd.Items = new InputProperty[] { childRecLocationInput };

req = new TrimRequest();
req.Items = new Operation[]{sru, upd};

// Execute the request
```

This will have the result of only changing the child list item of type recLocation, with the URI of 221.

Creating a child list item

The HP TRIM Web Service can also create child list items. This is done with the CreateChildItem operation, which takes a type for the child object, and a list of the input properties to set on the new object. The way this operation is used is to create or otherwise select the objects you want to create the child list item for, and then execute the CreateChildItem operation.

Here is an example:

Example 4 – creating a child list item

```
CreateChildItem cci = new CreateChildItem();
cci.ChildObjectType = "reclocation";

trimct.InputProperty loc = new trimct.InputProperty();
loc.Name = "rlcLocation";
loc.Val = "inject:katie";

trimct.InputProperty type = new trimct.InputProperty();
type.Name = "rlcRecLocType";
type.Val = "Contact";

cci.Items = new InputProperty[] { loc, type };

// Execute the request
```

In this example we are creating a new contact on a record. The new contact is the Location which was previously found using URI injection.

More information about URI injection is available in the *Injection* chapter in this document.

Conclusion

In this chapter we have discussed the purpose of child lists on HP TRIM objects, how to find child list items by treating them as properties, how to change the value of a child list item by using a conditional change, and finally how to create a new child list item.

7 Record operations

Introduction

This chapter will cover operations related to the record object in HP TRIM.
It will not cover any search objects. While there are many record-specific searches in the HP TRIM Web Service, these are covered in *The RecordSearch Operation*.

AddRendition

The AddRendition operation is similar to the CheckIn object, except it is used to add additional documents to a record. Before it can be used successfully, the new rendition document needs to be sent to the Web Service computer using the Upload method. Like CheckIn, AddRenditions needs the UploadId that is returned from the Upload operation (refer to *CheckIn and CheckOut* for more information on Uploading documents), and has a number of RenditionTypes that can be specified

Example 1 – adding a Rendition

```
//Perform an Upload, and then remember the UploadId
ShortcutRecordUri sru = new ShortcutRecordUri();
sru.Uri = "559";
sru.IsForUpdate = true;

AddRendition ar = new AddRendition();
ar.RenditionType = RenditionTypes.Redaction;
ar.Description = "This is the redacted version";
ar.UploadId = uplid;

req.Items = new Operation[] { sru, ar };
resp = engine.Execute(req);
```

AppendNotes

While a record's notes can be modified using the **recNotes** specification property, the AppendNotes operation provides slightly different functionality. The idea is you do not need to know the current value of the notes you are just appending. The **AddUserStamp** property adds the user's name and the time before the notes are appended.

Example 2 – appending notes

```
ShortcutRecordUri sru = new ShortcutRecordUri();
sru.Uri = "559";
sru.IsForUpdate = true;

AppendNotes an = new AppendNotes();
an.AdditionalNotes = "I've made those corrections";
an.AddUserStamp = true;

TrimRequest req = new TrimRequest();
req.Items = new Operation[] {sru, an};
```

AttachKeyword

This allows a Thesaurus term to be applied to a record. The KeywordUri is the URI of the Thesaurus term to be applied to the record.

Example 3 – attaching a keyword

```
ShortcutRecordUri sru = new ShortcutRecordUri();
sru.Uri = "559";
sru.IsForUpdate = true;

AttachKeyword ak = new AttachKeyword();
ak.KeywordUri = "6";

TrimRequest req = new TrimRequest();
req.Items = new Operation[] {sru, ak};
```

CheckInMailMessage

This object expands the normal CheckIn operation, allowing additional email-related information to be entered. The various recipients of the email can be specified through the Recipients collection, and whether the e-mail should be left checked out or if it is a new revision.

You can also handle any attachments associated with the mail message. These need to be uploaded prior to the CheckInMailMessage operation (as does the mail message itself – see *CheckIn and CheckOut*. Any attachments need to be handed to the CheckInMailMessage operation as MailAttachment objects.

Example 4 – checking in mail message

```
MailAttachment ma = new MailAttachment();
ma.DisplayName = "comparisons.jpg";
ma.DocumentExtension = "jpg";
ma.UploadId = attachmentUplid;

MailRecipient mr = new MailRecipient();
mr.MailAddress = "name@company.com";
mr.DisplayName = "Firstname Surname";
mr.Type = MailRecipientTypes.To;

CheckInMailMessage cim = new CheckInMailMessage();
cim.UploadId = emailUplid;
cim.Attachments = new MailAttachment[] {ma};
cim.Recipients = new MailRecipient[] {mr};
cim.SentDate = "27/9/05";

ShortcutRecordUri sru = new ShortcutRecordUri();
sru.Uri = "563";
sru.IsForUpdate = true;
```

CompleteCurrentAction

Include this operation in a Request to complete the current action on the associated record.

Example 5 – completing a current Action

```
ShortcutRecordUri sru = new ShortcutRecordUri();
sru.Uri = "561";
sru.IsForUpdate = true;

CompleteCurrentAction cca = new CompleteCurrentAction();

TrimRequest req = new TrimRequest();
req.Items = new Operation[] {sru, cca};
```

CreateCopy

CreateCopy duplicates an existing record. You can specify the type of copy you are making and make a suggestion for a new number.

Example 6 – creating a copy of an existing record

```
ShortcutRecordUri sru = new ShortcutRecordUri();
sru.Uri = "559";
sru.IsForUpdate = true;

CreateCopy cc = new CreateCopy();
cc.Type = CopyType.Part;
cc.SuggestedNumber = "19/05-02";

TrimRequest req = new TrimRequest();
req.Items = new Operation[] {sru, cc};

Engine engine = new Engine();
engine.Credentials = System.Net.CredentialCache.DefaultCredentials;

TrimResponse resp = engine.Execute(req);
```

DeleteRendition

If you need to remove a Rendition, the DeleteRendition object is the thing to use. It is used in conjunction with a search operation, much like the AddRendition operation mentioned earlier in this chapter. The URI of the Rendition needs to be specified as well.

Example 7 – deleting a Rendition

```
ShortcutRecordUri sru = new ShortcutRecordUri();
sru.Uri = "561";
sru.IsForUpdate = true;

DeleteRendition dr = new DeleteRendition();
dr.ChildUri = "502";

TrimRequest req = new TrimRequest();
req.Items = new Operation[] {sru, dr};
TrimResponse resp = engine.Execute(req);
```

Finalize

The Finalize operation is used to finalize a record. It is similar to the CompleteCurrentAction operation in that it can be passed across without any parameters being set. It can be considered simply a flag that can be set.

Example 8 – finalizing a record

```
ShortcutRecordUri sru = new ShortcutRecordUri();
sru.Uri = "561";
sru.IsForUpdate = true;

Finalize fin = new Finalize();

TrimRequest req = new TrimRequest();
req.Items = new Operation[] {sru, fin};
TrimResponse resp = engine.Execute(req);
```

Conclusion

In this chapter we covered the record-specific operations that are not covered in other parts of this documentation, specifically the chapters about searching. The operations covered included adding and removing a rendition, completing a current action and finalizing a record.

8 Non-Record Operations

Introduction

There are several operations which can be performed on objects within HP TRIM which are not records through the HP TRIM Web Service. This chapter covers the set of those operations which is not covered in other chapters of this manual. The operations are broken into a series of groups, each of which is discussed in turn.

Access Control

Access Control Lists (ACLs) can also be manipulated with the HP TRIM Web Service. For further information on how ACLs work in HP TRIM, refer to the HP TRIM SDK documentation.

HasAccess

The HasAccess operation is used to test if a given user has access to a given object. The operation takes the URI of the Location to check for access for (a Location often being a person within HP TRIM), and the type of ACL to check for.
For more information on how the various ACL list types in HP TRIM interact, refer to the HP TRIM SDK documentation for the HasAccessBase().

The HasAccessResult returns a text description of the type of access which was checked for in a string suitable for display to your users. It also contains a list of Access Control items describing the access.

The HasAccess operation is performed on the objects returned from the previous search.

RemoveAccess

To remove access to objects you would use the RemoveAccess operation. This operation takes the same arguments as the HasAccess operation – an ACL type, and the URI of the Location to operate on.

AppendAccess

To append a Location to an ACL, use the AppendAccess operation. This operation takes the ACL type, and a list of Location URIs to append to the ACL. It then performs the ACL manipulation without you needing to know what other items are in the ACL. This is because if you grabbed the ACL, modified it and then handed the new ACL to the HP TRIM Web Service, that would require at least two requests; and it is possible that someone else would have

changed the ACL while you were processing it before your second request. This would have resulted in their changes to the ACL being overwritten by yours.

This operation returns a SuccessResult.

Deleting an object

The HP TRIM Web Service allows you to delete one object at a time with the Delete operation. This operation operates on the result of the last search operation. As such, care should be taken not to accidentally delete a large number of objects.

Information about the HP TRIM dataset and current connection

ConnectionInfo

The ConnectionInfo operation returns information about the HP TRIM dataset in use and the currently connected user. The operation does not take any arguments, and returns a ConnectionInfoResult result. This result contains the following values:

Value returned	Description
CurrentUserUri	The URI of the currently connected user
CurrentUserNickName	The nickname of the currently connected user
CurrentUserLogin	The network login for the current user
LicenseName	The name of the license holder for this HP TRIM dataset
LicenseNumber	The license number for this installation

IsLicensed

Similarly, the IsLicensed operation is used to test if a given piece of HP TRIM functionality is licensed by the current HP TRIM installation. This can be used by client applications to determine if functionality within their applications should be enabled or disabled, or if in fact that given application will work in a given HP TRIM environment. The IsLicensed operation takes a LicenseTypes enumeration entry as an argument, and returns a Boolean wrapped in an IsLicensedResult if that functionality is enabled for the current HP TRIM dataset.

The LicenseTypes enumeration contains these items:

Workflow = 0,

RecordsManagement = 1,

SpaceManagement = 2,

EnterpriseCachedStore = 3,

DocumentContent = 4,

WebServer = 5,

GuestGateway = 6,

DocumentManagement = 7,

ClassifiedSecurity = 8,

CustomResourcesAvailable = 9,

AnnotateRedact = 10,

DocumentAssembly = 11,

MeetingManager = 12

UserLabels

UserLabels are a user specific grouping.

For more information on the mechanics of UserLabels, refer to the HP TRIM SDK documentation.

ApplyUserLabel

To apply a UserLabel to a set of objects, you use the ApplyUserLabel operation. This operation takes a UserLabel URI, and applies this UserLabel to all of the objects returned by the previous search operation.

RemoveFromUserLabel

Similarly, to remove a UserLabel, use the RemoveFromUserLabel operation. This operation takes the URI of a UserLabel, and removes that label from all of the objects returned by the previous search.

Conclusion

In this chapter we have covered the operations not covered by other chapters of the documentation. In several instances we have recommended that you refer to the HP TRIM SDK documentation – in any case because the HP TRIM Web Service is built on top of the HP TRIM SDK this is a good idea anyway.

9 CheckIn and CheckOut

Introduction

One of the great things about HP TRIM is that it lets you store various electronic documents. Any record may contain your current invoice list as a Word document or photos from the last company picnic. Documents can be checked out, modified and then checked back in while HP TRIM keeps track of the changes.

Naturally, this ability has been included in the HP TRIM Web Service, allowing a user on the other side of the world to download a document, modify it and then check it back in.

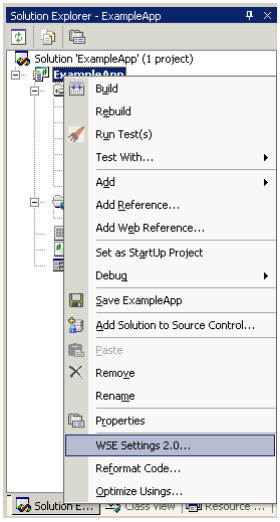
Web Service Enhancements 3

Web Service Enhancements, or WSE are a vital part of the file transfer and document manipulation objects in the HP TRIM Web Service if you are using a Microsoft programming language to call the Web Service.
It is needed to implement WS-attachments. Other language vendors will have equivalent products. The Inline transfer method exists for developers who do not want to or cannot use WSE.

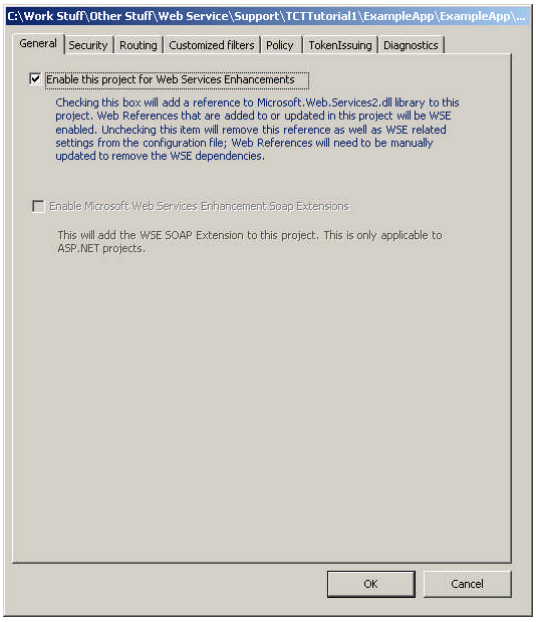
Prerequisites

1. Before any coding can be started or any example run, make sure you have WSE 3 or equivalent properly installed and set up.
2. WSE 3 can be downloaded from Microsoft at http://www.microsoft.com/downloads/.
3. If you install WSE 3 on a computer that you will be using to develop HP TRIM Web Service applications with Microsoft Visual Studio, you will need to select the **Visual Studio Developer** option.
4. Once installed, you need to restart Visual Studio.
   The WSE 3 will now be integrated with Visual Studio and can be included in projects.
5. Once you have opened or created the solution you want to use WSE with, go to the Solution Explorer and right-click the solution.
6. Near the bottom of the menu that appears, click **WSE Settings 3.0**.

The **Solution Explorer** dialog will appear:



7 Under the **General** tab, check the box **Enable this project for Web Services Enhancements**.



8 Click **OK** and update the Web reference.

Downloading

Download is used to retrieve a specific document from the HP TRIM database and transfer it to the client's computer.

1. Create the Engine object
2. Once the engine has been established, it is time to create a search of some kind to find the record with the attachment you want to download. For this example, we will just assume we already know the URI of the record we want.
3. Once the Download object has been created, a few parameters need to be set. Checkout is where you decide if you want to check the document out or get a local copy.
4. After the request has been built, handed over to the server and the request has been returned, you will need to loop through the results as normal, this time paying particular attention for the DownloadResult.

Example 1 – downloading via MTOM

```
   // some initial variables
string fileLoc = @"c:\junk\mydocument.doc";
bool finished = false;

// here the chunk size is small to demonstrate downloading
// a document in more than one chunk.  Depending on the
// network configuration the download chunk size would
// probably be big enough to download most reasonably sized
// documents in one chunk.
int DownloadChunkSize = 10000;


//Create the Engine object
Engine engine = new Engine();
engine.Credentials = CredentialCache.DefaultCredentials;

// Setup the number search
RecordStringSearchClause stringClause
    = new RecordStringSearchClause();
stringClause.Type = RecordStringSearchClauseType.RecordNumber;
stringClause.Arg = "DOC1-1";

//Create the record search
RecordSearch recSearch = new RecordSearch();
recSearch.Items = new RecordClause[] { stringClause };

// The download object must be of type MTOM and specify
//a download chunk size.
Download down = new Download();
down.Checkout = false;
down.TransferType = TransferTypeType.mtom;
down.MaximumTransferBytes = DownloadChunkSize;

// Create the TRIM request
TrimRequest request = new TrimRequest();
request.Items = new Operation[] { recSearch, down };

using (FileStream fs = File.OpenWrite(fileLoc))
```

```
{
    while (!finished)
    {
        //The TRIM request is sent repeatedly until the whole
        // document has been downloaded.
        TrimResponse resp = engine.Execute(request);

        foreach (Result chunkRes in resp.Items)
        {
            switch (chunkRes.GetType().Name.ToString())
            {
                case "ErrorResult":
                    ErrorResult err1 = (ErrorResult)chunkRes;
                    Console.WriteLine(err1.Message);
                    return;
                case "DownloadResult":
                    DownloadResult dnres
                        = (DownloadResult)chunkRes;
                    finished = dnres.FinalChunk;

                    fs.Write(
                        dnres.MTOMPayload,
                        0,
                        dnres.MTOMPayload.Length);

                    // setting the transfer inset tells the
                    // web service where in the
                    // file to start sending
                    down.TransferInset
                        = down.TransferInset
                        + DownloadChunkSize;

                    // The download Id allows the web service to
                    // identify which download for this record
                    // we are referring to.
                    down.DownloadId = dnres.DownloadId;
                    break;
            }
        }
    }
    fs.Close();
}
```

Example 2 – downloading via Inline

```
// Create an Engine object
Engine exec = new Engine();
exec.Credentials = System.Net.CredentialCache.DefaultCredentials;

// Assume we know the URI of the record
ShortcutRecordUri rus = new ShortcutRecordUri();
rus.Uri = "12";

// Create the Download object
Download down = new Download();
down.Checkout = false;
down.Comments = "TRIM Web Service Example";
down.MaximumTransferBytes = 0;
down.TransferInset = 0;
```

```
        down.TransferType = TransferTypeType.inline;

        req.Items = new Operation[] { rus, down };
        TrimResponse resp = exec.Execute(req);
        byte[] data;

        foreach(Result res in resp.Items)
        {
                if(res.GetType().Name.ToString() == "DownloadResult")
                {
                        DownloadResult dres = (DownloadResult) res;
                        data = Convert.FromBase64String(dres.Base64Payload);
                        using(FileStream output = new FileStream(@"pic.jpg",
FileMode.Append))
                        {
                                output.Write(data, 0, data.Length);
                                output.Flush();
                                output.Close();
                        }
                }
        }
```

Uploading

Once you have finished with the document you just downloaded – or if you have a new document you want to put into HP TRIM – you will need the Upload operation. Like Download, Upload requires WSE 3 to take advantage of MTOM transfers.

Upload takes an electronic document you specify and places it on the Web Server computer. You then attach the document to a record in HP TRIM, which moves it to the e-store. So obviously, you need a record in HP TRIM to attach the document to.

If there is not a record for the document already in HP TRIM, you have to create it first. Otherwise, you just need to find it, but remember to mark the record for update.

Example 2 – Upload and Checkin

Uploading a file to a TRIM server via the Web service is a two step process:

1. Upload the file
2. Check it in to a particular record.

The following class demonstrates this, first uploading the file and returning a tempory identifier, then using that identifier to check the document into a record.

```
    class FileManager
{
    private Engine engine = new Engine();
    private string upload(string fileLoc)
    {
        int uploadChunkSize = 10240;
        int startFrom = 0;

        Upload upload = new Upload();
        upload.TransferType = TransferTypeType.mtom;
        TrimRequest request = new TrimRequest();
        request.Items = new Operation[] { upload };
```

```
    using (FileStream file = File.OpenRead(fileLoc))
    {
        while (startFrom < file.Length)
        {
            if ((file.Length - startFrom) < uploadChunkSize)
            {
                uploadChunkSize
                    = Convert.ToInt32(file.Length - startFrom);
            }

            upload.MTOMPayload = new byte[uploadChunkSize];
            file.Read(upload.MTOMPayload, 0, uploadChunkSize);
            upload.BytesRead = uploadChunkSize;

            TrimResponse resp = engine.Execute(request);
            foreach (Result result in resp.Items)
            {
                switch (result.GetType().Name.ToString())
                {
                    case "ErrorResult":
                        ErrorResult err1
                            = (ErrorResult)result;
                        Console.WriteLine(err1.Message);
                        throw new Exception(err1.Message);
                    case "UploadResult":
                        UploadResult uploadResult
                            = (UploadResult)result;
                        upload.UploadId
                            = uploadResult.UploadId;
                        startFrom += uploadChunkSize;
                        upload.UploadId = upload.UploadId;
                        break;
                }
            }
        }

        return upload.UploadId;
    }
}

private void checkin(string uploadId)
{
    CheckIn checkin = new CheckIn();
    checkin.Comments = "test";
    checkin.DocumentExtension = ".doc";
    checkin.FailOnWarning = true;
    checkin.KeepCheckedOut = false;
    checkin.UploadId = uploadId;

    // Setup the number search
    RecordStringSearchClause stringClause
        = new RecordStringSearchClause();
    stringClause.Type
        = RecordStringSearchClauseType.RecordNumber;
    stringClause.Arg = "DOC1-1";

    //Create the record search
    RecordSearch recSearch = new RecordSearch();
```

```
    recSearch.IsForUpdate = true;
    recSearch.Items = new RecordClause[] { stringClause };

    TrimRequest request = new TrimRequest();
    request.Items = new Operation[] { recSearch, checkin };

    TrimResponse response = engine.Execute(request);
    foreach (Result result in response.Items)
    {
        switch (result.GetType().Name.ToString())
        {
            case "ErrorResult":
                ErrorResult err1 = (ErrorResult)result;
                Console.WriteLine(err1.Message);
                return;
        }
    }
}
public void fileCheckin(string fileLoc)
{
    engine.Credentials = CredentialCache.DefaultCredentials;

    string uploadId = upload(fileLoc);
    checkin(uploadId);
}
```

Conclusion

Manipulating documents through the HP TRIM Web Service requires Microsoft's Web Server Enhancements 3 or equivalent. Some users have encountered a problem within Visual Studio .NET where the WSE objects are not recognized, but this can be corrected by updating the solution's Web Reference.

Uploading and checking in a document needs to be done in two separate executes, as the Upload ID is needed for the CheckIn operation, but cannot be passed directly from the Upload operation.

10 Advanced searching

Introduction

In this chapter, we are going to explore some of the more complex search options available to the HP TRIM Web Service developer.

Record Location searches are useful when trying to locate records by Locations – such as the author or the current Location.

This chapter will also cover building searches over several properties. These Boolean searches could, for example, find all the records registered between two dates and which have the title word **reef** in them.

Record Location searches

Searching for records by their related Locations is extremely useful for tracking the life of a record – who first created it, who has modified it, and where the record currently is. To address this functionality, the HP TRIM Web Service has the RecordLocationTypeSearchClause.

Through the LocationType property, you can use the RecordLocationTypeSearchClause to find out the current Assignee of a record, the actual Owner Location or who has the record currently checked out.

Example 1 – searching records by Locations

```
RecordLocationTypeSearchClause ass = new RecordLocationTypeSearchClause();
ass.LocationUri = 17; //Peter Abbot
ass.Type = RecordLocationTypeSearchClauseType.Location;
ass.LocationType = SearchLocationTypes.Current;
```

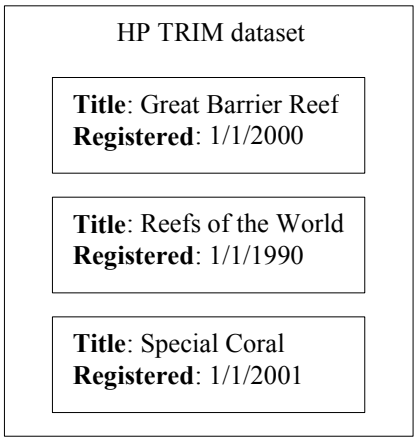
Boolean searching

In earlier chapters, we looked at how to search a HP TRIM dataset for a specific criterion. Boolean searching means searching over more than one criterion, and it is one of the more powerful features of the Web Service.

Logical clauses

Special clauses exist to combine two or more search clauses. Collectively, these are called the logical clauses, since each of them represents a logical separator. Logical separators are used to combine two items. One of the really nice things about logical separators is that you can combine two or more statements together to create some quite complex algorithms. There are two types of logical separators available in the HP TRIM Web Service – AND clauses and OR clauses.

For the following examples, imagine we have three records in our HP TRIM dataset - "Great Barrier Reef" registered in 1/1/2000, "Reefs of the World" registered on 1/1/1990 and "Special Coral" registered on 1/1/2001.



Sample records in a HP TRIM dataset

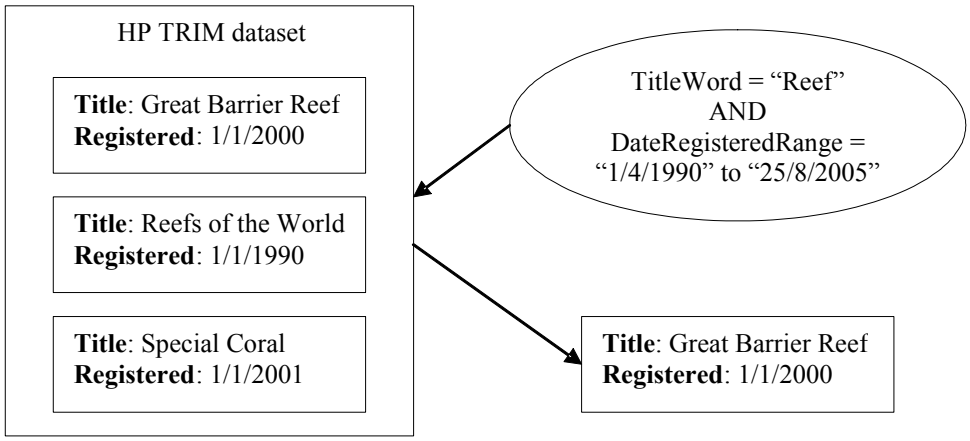
AND clause

An AND clause means that an object must meet both of the search clauses to be included in the found set.

An example of an AND search would be:

- looking for records with the title word reef AND registered between today and 1st April 1990.

  Of the three records in our sample database, the search would only return the record "Great Barrier Reef".



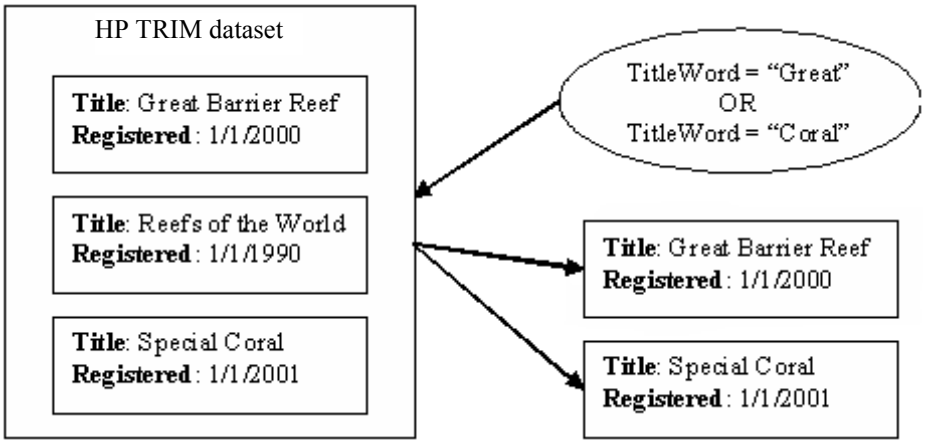
AND search results

OR clause

An OR clause is used to look for objects than meet either clause.
It is worth noting that an OR clause will also return an item that meets both criteria.
An example of an OR clause would be:

- looking for all the records with the title word of “great” OR “coral”.

The search in our sample database would return two records:



OR search results

If you used the same two criteria with an AND clause, neither of the records would have been found. Only a record with “Great” and “Coral” would be returned.

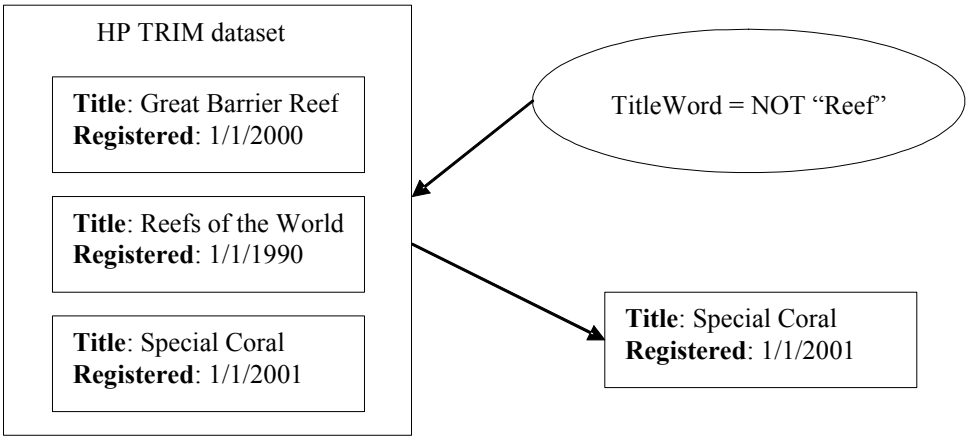
NOT clauses

There is a third type of logical clause we have not talked about yet.

Unlike AND and OR, the NOT clause is not a Boolean clause. Instead, think of a NOT clause as a modifier.

It returns the items that do not contain the value specified.

In our sample dataset, while a search for records with the title word **reef** would return two of three records, the same search with a NOT clause would return only one record, which is the only one that does not contain the title word **reef**.



NOT search results

Boolean search format

Boolean searches in the Web Service use the reverse Polish notation. A normal equation has the two parameters separated by an operator.
Adding together two numbers would look like this:

2 + 27

Reverse Polish works differently. Instead of parameter-operation-parameter, the reverse Polish format is parameter – parameter – operator.
The example above would be:

2 27 +

So when applied to searches in the HP TRIM Web Service, the notation looks like this:

```
SearchClause1 SearchClause2 LogicalClause
```

Building a Boolean search

You can combine any search clauses with a Boolean separator to produce a more complex search.

So, as we said earlier, you can search by the RecordTitle and DateRegistered, or two different surnames or any other combination of SpecificationProperties.

By this stage you should be able to construct two different searches, send off both of these searches and get back two separate datasets. It is fairly simple to combine the two searches.

As you may have realized, the AND, OR and NOT clauses are individual clauses in the HP TRIM Web Service. They are created in the same way as other Clause objects, except they have no parameters that need to be set.

Example 2 – creating an ANDed search

```
RecordStringSearchClause titleword = new RecordStringSearchClause();
titleword.Type = RecordStringSearchClauseType.TitleWord;
titleword.Arg = "Reef";

RecordDateRangeSearchClause date = new RecordDateRangeSearchClause();
date.Type = RecordDateRangeSearchClauseType.DateRegistered;
date.StartTime = "1/4/1990";
date.EndTime = "25/1/2005";

RecordAndSearchClause and = new RecordAndSearchClause();

search.Items = new RecordClause[] {titleword, date, and};
request.Items = new Operation[] {search, fetch};
```

Multiple Boolean conditions

So far, we have talked about how to use the logical search clauses to search across two parameters. But it is possible to construct even more complex search queries by using multiple Boolean clauses.

Example 3 – multiple Boolean clauses

```
{titleword, date1, and, titleword, date2, and, or};
```

Dealing with multiple Boolean clauses can become very confusing very quickly. It may be easier to think of the parameters in pairs.
So the example above would read as:

```
((titleword, date1, and), (titleword, date2, and), or)
```

If we translate the Polish notation into a more natural notation, we get:

```
((titleword and date1) or (titleword and date2))
```

Conclusion

While constructing Boolean searches can quickly become confusing if you are not careful, they are a powerful part of the HP TRIM Web Service.
They allow you to either refine or expand your search conditions to cater for a much larger group of potential searches than would be possible with individual searches.

Record Location searches allow records to be searched on by their associated Locations. This allows electronic documents to be tracked as they pass between different individuals or to help track down the current Location of physical records.

11 Injection

Introduction

There are potentially many situations where a developer using the HP TRIM Web Service will need to perform some operations on an object but will not know until runtime which specific object it is.
Based on previous chapters, the only option would be to perform two executes on the Engine – one to identify the specific object, and a second to carry out the operations.

To cater for this situation, the concept of 'injection' was developed.
While there are several different types of injection, the basic functionality is the same: Rather than passing a URI into a parameter, you can pass a string value that the HP TRIM Web Service can resolve for a URI. Injection works anywhere you would use a URI.

URI injection

URI injection is used in the situations where the URI of the object is being found with some type of search, and then passed through to the next operation. Take, as an example, dealing with a RecordType URI when creating a new record. It is possible to do a search for the RecordType, have its URI returned and then placed in a local variable before being passed into the Create object and sent off in a second execute.

Using URI injection means that these two executes can be grouped together – the search is carried out, and then the HP TRIM Web Service on the server side directly passes the result to the Create object using a FetchInjectionUri object.

The FetchInjectionUri object works a bit differently to a normal Fetch object. Instead of specifying what properties you want returned, you instead specify the FetchInjectionUri's ID. This ID can be any string, but it needs to be unique for the Execute. The FetchInjectionUri object then caches the URI found by the search for later use in the request.

Dealing with injected URIs

By now you should be familiar with at least some of the objects in the HP TRIM Web Service that take a URI as a parameter - for example, specifying the Record Type when creating a new record.

For URI injection to work, you first need a successful search. We will assume for this example that you at least know the name of the Record Type you need. So you will need a search that can find the Record Type. As mentioned earlier, you can set the ID of the injection object to any string, but it needs to be unique for the request.

Example 1 – creating a record using Inject

```
Engine engine = new Engine();
engine.Credentials = System.Net.CredentialCache.DefaultCredentials;

RecordTypeStringSelect rts = new RecordTypeStringSelect();
rts.Arg = "Document";
rts.Id = "recordtype";

InputProperty name = new InputProperty();
name.Name = "recTitle";
name.Val = "New Record";

InputProperty type = new InputProperty();
type.Name = "recRecordType";
type.Val = "inject:recordtype";

Create create = new Create();
create.Items = new InputProperty[] {type, name};
create.TrimObjectType = "record";

FetchInjectionUri fiu = new FetchInjectionUri();
fiu.Id = "recordtype";

TrimRequest req = new TrimRequest();
req.Items = new Operation[] {rts, fiu, create};
TrimResponse resp = engine.Execute(req);
```

The first InputProperty is pretty standard – it is just specifying the record title. The second InputProperty is what we are really interested in.
Notice that instead of specifying the URI, we have set it to 'inject:recordtype' instead. This is basically saying that this value will be set by the FetchInjectUri object.

As you have probably guessed, the reason the ID of the FetchInjectionUri object needs to be unique within a single request is that otherwise the HP TRIM Web Service does not know which URI to inject where. In fact if you reuse a FetchId, the old value will be silently overwritten with the new one.

Things to remember

URI injection replaces the need to search for an object and return its URI before using the URI in another request.
Naturally, this is only useful when dealing with a single URI. The operation will fail if the FetchInjectionUri has to deal with anything but a single URI, so the search returns either nothing or more than one URI. This prevents anything unwanted from happening if, for example creating a new record, your search returned two Record Types.

Name

If you know the name of the object, you can use the 'name:' injector.

The 'name:' injector saves the need to do a shortcut name or similar search. When the HP TRIM Web Service detects the 'name:' syntax, it carries out a relevant name-based search automatically by constructing the object using its name as you would in the HP TRIM SDK.

Example 2 – creating a record using Name

```
Engine engine = new Engine();
engine.Credentials = System.Net.CredentialCache.DefaultCredentials;

InputProperty name = new InputProperty();
name.Name = "recTitle";
name.Val = "New Record";

InputProperty type = new InputProperty();
type.Name = "recRecordType";
type.Val = "name:document";

Create create = new Create();
create.Items = new InputProperty[] {type, name};
create.TrimObjectType = "record";

TrimRequest req = new TrimRequest();
req.Items = new Operation[] {create};
```

Conclusion

Injectors are designed to help minimize executes and make life easier for the developer.

There are many situations where using an injector will not be useful – or even possible. For example, a common mistake is trying to use an injector to pass an upload ID (see *CheckIn and CheckOut*) to a CheckIn object. This will not work as the upload ID does not resolve to a URI.

12 Shortcut Operations

Introduction

To make development easier, several 'shortcut' objects are included in the HP TRIM Web Service.

These shortcuts are designed to make some of the more common tasks within HP TRIM easier to perform, saving on the need to construct complex clauses. This means they are ideally suited to be used in conjunction with updates, injections, deletions and similar operations.

Records

As records are really the key objects in the HP TRIM Web Service, it is natural that the majority of the Shortcut objects are related to records.

ShortcutRecordNumber

Every record in HP TRIM has a notionally unique record number within a given Record Type. This follows a pattern defined by the Record Type and can be manually entered by the user or set to be automatically generated by HP TRIM. Although the commonly used term is 'number', it is more correctly an identifier, as it is a string that may contain alphanumeric characters. To access this string, pass a SpecificationProperty with the name "recNumber" in with your fetch operation.

Example 1 - ShortcutRecordNumber

```
// Construct a request object
TrimRequest request = new TrimRequest();

ShortcutRecordNumber recNum = new ShortcutRecordNumber();
recNum.Id = "recordNumber";
recNum.RecordNumber = "2005/0059";

// Put our shortcut operation into our TrimRequest
request.Items = new Operation[] { recNum, fetch };
```

Note

HP TRIM stores the record number in two formats, expanded (e.g. "2002/0059") and compressed (e.g. "02/59"). Both can be passed to the number property.

ShortcutRecordTitle

If you know the title of a record, then this shortcut search will be of most use.

It works like the title word search in the HP TRIM UI – it will return matches based on one or more words, and you can incorporate wildcards.

Example 2 – ShortcutTitleNumber

```
// Construct a request object
TrimRequest request = new TrimRequest();

ShortcutRecordTitle recTitle = new ShortcutRecordTitle();
recTitle.Id = "recordTitle";
recTitle.TitleWord = "reef";

// Put our shortcut operation into our TrimRequest
request.Items = new Operation[] { recTitle, fetch };
```

ShortcutRecordUri

The Unique Row Identifier or URI of a record is an internal unique number that is transparent to the everyday user of HP TRIM. It is the primary key on the TSRECORD table in the database and provides an internal unique identifier for every record.

To instantiate a record by its URI, you can use the ShortcutRecordUri Operation. It works in much the same way as ShortcutRecordNumber.

Example 3 - ShortcutRecordUri

```
// Construct a request object
TrimRequest request = new TrimRequest();

ShortcutRecordUri recUri = new ShortcutRecordUri();
recUri.Uri = "785";

// Put our shortcut operation into our TrimRequest
request.Items = new Operation[] { recUri, fetch };
```

Once an instantiated record object has been returned by the either of these shortcut methods, the developer can access whatever properties were returned from the object.
Reading this data requires inspecting the FetchResult, as we have done in Chapter 4 - updating these requires a new operation that we have not yet explored – the Update operation.

ShortcutRecordUris

If you need to select multiple records, rather than use multiple instances of the ShortcutRecordUri object, then use a ShortcutRecordUris operation. All of the records found by this search will have the associated operations performed on them – for example a ShortcutRecordUris search that is accompanied by an operation to change the record author to "Joe Blogs" will alter all the matching records.

Example 4 - ShortcutRecordUris

```
// Construct a request object
TrimRequest request = new TrimRequest();

ShortcutRecordUris srus = new ShortcutRecordUris();
srus.Uris = new string[] {785, 786, 789};

// Put our shortcut operation into our TrimRequest
request.Items = new Operation[] { srus, fetch };
```

ShortcutRecordUris accepts an array of URIs, and so it is worth nothing that it can accept as few as one URI.

ShortcutRecordDateRegistered

The final shortcut method for finding records searches the HP TRIM dataset for records that were registered between a certain date range.

Example 5 - ShortcutDateRange

```
// Construct a request object
TrimRequest request = new TrimRequest();

ShortcutRecordDateRegistered srdr = new ShortcutRecordDateRegistered();
srdr.StartTime = "1/1/1990";
srdr.EndTime = "1/1/2005";

// Put our shortcut operation into our TrimRequest
request.Items = new Operation[] { srdr, fetch };
```

Locations

Currently there is only one Shortcut object for Locations.

ShortcutLocationName

ShortcutLocationName is a useful search for getting Locations with a specific name. It is worth noting that when search on 'people' Locations, the Locations are searched by their surnames – in fact, the SpecificationProperty that is searched on is named “locSurname”.

Example 6 - ShortcutLocationName

```
ShortcutLocationName sln = new ShortcutLocationName();
sln.Name = "smith";

TrimRequest request = new TrimRequest();
request.Items = new Operation[] {sln, fetch};
```

Conclusion

Shortcut operations are designed to save time for the developer by simplifying some of the more common search operations. Combined with *URI injections*, Shortcuts are particularly powerful, allowing a developer to design considerably more efficient code.

13 XML methods

Introduction

It is possible that some users of the HP TRIM Web Service will be unable to make use of the full SOAP object functionality of the HP TRIM Web Service.
For example, you may have dropped XML data from a database and would rather use XSLT to put it in the Web Service request.

The HP TRIM Web Service supports these users by allowing for an XML string version of the TrimRequest object to be passed to a special method named ExecuteXml().

ExecuteXml() and ConvertToXml()

The string taken by ExecuteXml() is the XML form of the TrimRequest that was otherwise passed and can in fact be generated by calling the ConvertToXml() method, which takes in a TrimRequest and returns an XML string.

For example, let's take a simple TrimRequest and convert it to XML.

Here is some code to set up a simple search:

Example 1 – converting to XML

```
TrimRequest req = new TrimRequest();
req.HideVersionNumbers = true;
RecordSearch recSearch = new RecordSearch();

// Now we need to make one or more search clauses for that search. We only make
// one here, as we are only going to do a title word search.
RecordStringSearchClause titleWordClause = new RecordStringSearchClause();
titleWordClause.Arg = "protective";
titleWordClause.Type = RecordStringSearchClauseType.TitleWord;

recSearch.Items = new RecordClause[] { titleWordClause };
req.Items = new Operation[] { recSearch };

Engine sdk = new Engine();
sdk.Credentials = System.Net.CredentialCache.DefaultCredentials;
m_SimpleXmlData = sdk.ConvertToXml(req);
```

Example 2 – XML result

```
<?xml version="1.0"?>
<TrimRequest xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <RecordSearch>
  <IsForUpdate>false</IsForUpdate>
  <Limit>0</Limit>
  <Sort1>None</Sort1>
  <Sort1Descending>false</Sort1Descending>
  <Sort2>None</Sort2>
  <Sort2Descending>false</Sort2Descending>
  <Sort3>None</Sort3>
  <Sort3Descending>false</Sort3Descending>
  <FilterFinalizedState>Both</FilterFinalizedState>
  <RecordStringSearchClause>
   <Arg>protective</Arg>
   <Type>TitleWord</Type>
  </RecordStringSearchClause>
 </RecordSearch>
 <HideVersionNumbers>true</HideVersionNumbers>
 <ProvideTimingResults>false</ProvideTimingResults>
 <ForceRealTimeCacheUpdate>false</ForceRealTimeCacheUpdate>
</TrimRequest>
```

You can use this XML as the basis for a more complicated XML string if it is needed. You can then execute this XML string like this:

Example 3 – executing XML

```
Engine sdk = new Engine();
sdk.Credentials = System.Net.CredentialCache.DefaultCredentials;
sdk.ExecuteXml(xmlString);
```

Which would be the same as executing the original TrimRequest with Execute(). You receive the results in XML form, in this case, using my test dataset, I get:

Example 3 – XML result

```
<?xml version="1.0"?>
<TrimResponse xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <SearchResult>
  <FoundCount>2</FoundCount>
 </SearchResult>
 <EndResponse />
</TrimResponse>
```

Because there is no fetch, nothing more is returned.

Conclusion

In this chapter, we have demonstrated how to use the XML versions of the HP TRIM Web Service Web methods.

14 Debugging and error logs

Introduction

This chapter covers what to do when things do not work out perfectly with the HP TRIM Web Service.

We provide some ideas about where to start with debugging your applications, how to determine if an error has occurred within the HP TRIM Web Service, and our preferred method of reporting errors to the API Support team, who are discussed in the *Contacting Support* chapter in this document.

Debugging

Debugging your use of proprietary software can sometimes be hard. HP has however put a lot of effort into automated regression and unit testing for the HP TRIM Web Service.
It is possible that you have encountered a bug in the HP TRIM Web Service however, and we discuss how to go about reporting those bugs if you find one.

The first step to debugging a problem with the HP TRIM Web Service is to step through your client code and see if you can determine what the error being returned by the HP TRIM Web Service is, if there is one. Often these error messages will say helpful things like you have to specify a record type to create a record. If you receive an error message with such a hint, try implementing the hint and seeing if that helps.

If you need help interpreting the hint, or a less helpful error message is returned, then contact the API Support team as detailed in the *Contacting Support* chapter for assistance with interpreting the message. All messages returned by the HP TRIM Web Service have a unique ID associated with them, so please make a note of them.

Logging

Another useful technique to debugging what is happening in the HP TRIM Web Service is to turn on the trace logging facility, as discussed in the Installation chapter. This will then write messages as to the progress of your request to the configured log in the Windows Event Log applet.

These logs can help you determine what is happening with the request, as well as providing useful information to the API Support team.

Conclusion

In this chapter, we have discussed some introductory debugging techniques for the HP TRIM Web Service. Contact the API Support team if you need more assistance.

A Contacting HP TRIM API Support

API Support

HP TRIM Software has a team dedicated to providing developers with assistance with the HP TRIM SDK. Recently their role has expanded to include providing support for the Web Service. This team may be contacted at trimsdk@hp.com and prefers questions to include sample code, where possible.

The List Serve

HP TRIM Software also hosts an e-mail list for developers using the HP TRIM SDK or the HP TRIM Web Service to discuss issues arising from development with HP TRIM. While HP supports this community, we attempt to let other developers answer questions on the list first, so that this community develops more effectively.

Contact the API Support team for more information on how to join the mailing list.

HP TRIM Helpdesk

HP Software runs a full time Helpdesk to deal with general HP TRIM questions and problems. Unless you have a question specifically about the HP TRIM Web Service or the SDK, the Helpdesk should be your first point of contact.
They can be contacted through your local HP TRIM office.