9 Designing and Writing Output Connectors

This chapter covers

- Designing output connectors
- Writing output connectors
- Output connector techniques for improving crawling efficiency

In this chapter, we'll develop an output connector that writes documents to an example repository. We'll learn how to handle content, metadata and security information, and map input attributes to attributes in the target repository. When this basic functionality works, we'll also learn how an output connector can potentially improve the performance of the entire crawler by providing appropriate document characterization services.

9.1 Designing an output connector

When you next talk with your friend Bert, he's very happy. With the addition of a custom authority connector, he was easily able to integrate documents from arch-nemesis Frank's content repository into his search application in a manner that preserved their access security.

But it seems that more storm clouds are forming. Frank, astounded at how easily Bert tackled the security integration with the new Frank-en-Stein repository, has thrown down a final gauntlet: he has managed to give Bert the task of migrating content from other repositories into the new one. Furthermore, the migration is scheduled to take place over an extended period of time. That is, during the transition, the legacy repositories are required to remain accessible while users learn how to work with Frank-en-Stein.

Bert's not really quite sure how best to tackle this challenge, but he has confidence that ManifoldCF will fit in this picture somewhere. And, sure enough, you tell Bert that although this is not exactly the task that ManifoldCF was designed for, the ManifoldCF technology is perfectly capable of meeting the design goals as the manner they have been described. The way to do it is by writing and using a custom output connector. The tricky part will be in figuring out how to map arbitrary information from the various source repositories into entities that the target repositories understand. For a content migration task, this will especially be a challenge for security information, because in this use case the security must be translated, not merely projected as would be done if the target were a search engine. Taking care of this problem in a general manner will require some careful planning and design.

For this chapter's example, we'll be using Docs4U as a target repository of an output connector. Since Docs4U has no ability to store user or group based metadata, this will present a realistic challenge to the task of mapping security tokens, as we will see. It will also require some though as to how to handle each document's URL. We'll talk about that first.

9.1.1 What should we do with the document URL?

ManifoldCF uses a document's URL as a unique identifier for the document within the target repository. Therefore, it is essential that the URL for a document be stored alongside the document. Even more important, it must be the case that outputting a new document with the same URL as an older document should cause the older document to be replaced, and it should also be possible to delete a document by simply describing its URL.

Database aficionados will recognize these characteristics. In the database lexicon, the URL would be called a document's *primary key*. Almost all search engines and repositories also have a similar notion. The only problem is that sometimes the search engine or repository has its own idea of what the primary key for a document should be, and it's not the document's URL.

When this happens, your output connector really has no recourse other than to store the URL in some metadata field, and use whatever search functionality exists in the search engine or repository to look up the document's real primary key given the URL. It is possible that here, too, you will find yourself blocked by lack of functionality in the chosen target. It may not support metadata, or it may support metadata but not allow you to look up documents based on it. If that happens, with today's ManifoldCF you are simply out of luck; you cannot write an output connector for that repository.

Fortunately, this situation has never arisen, that I am aware of. Realistic search engines naturally handle metadata, and allow search on the metadata, which content repositories are considered pretty lame if they don't allow lookups of this kind as well.

In the case of Docs4U, we're going to presume that there's a metadata attribute which will contain the document's URL. The name of the metadata attribute will be selected as part of the output specification information, which the user will supply via the Crawler UI when the job is defined. We'll lay out the details when we design the output specification information, later in this section.

9.1.2 Deciding on how to handle access tokens

The first and most difficult design challenge for an output connector is figuring out how it should deal with security. As we've mentioned already, there are two different models we could use, which depend strongly on the capabilities of the target repository. The first model, which is called *projection*, requires that access tokens get stored somewhere in the target repository or index. The application that accesses content then becomes responsible for enforcing the security, using the ManifoldCF Authority Service as a support for doing this. See figure 9.1.

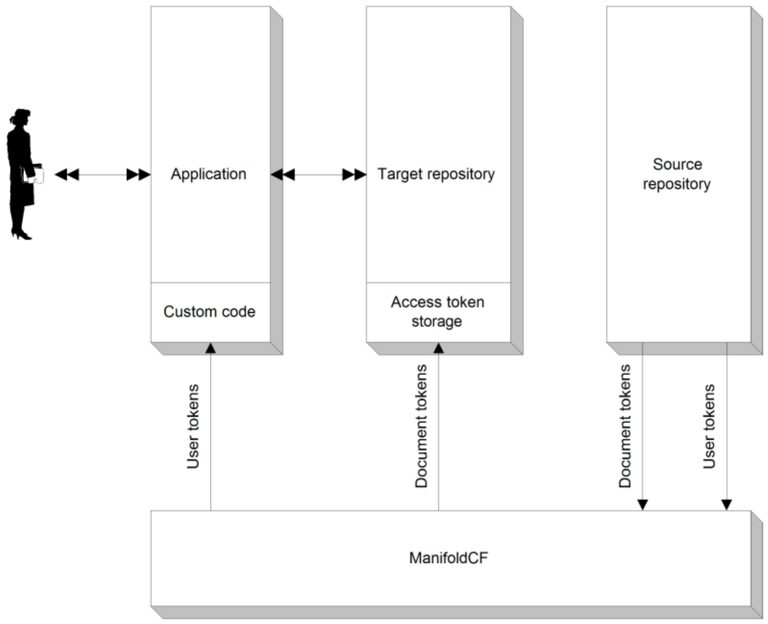


Figure 9.1 The *projection* security model. The application that end-users interact with must be modified to request a user's access tokens from ManifoldCF, which then gets them from the source repository. In addition, the target repository must have some way of storing document access tokens.

HANDLING ACCESS TOKENS IN THE PROJECTION MODEL

We've talked about the projection model at some length in Chapter 4 and Chapter 8, although we had not yet given it a formal name. It is the standard model ManifoldCF uses for handling repository security. For an output connector, the design decisions that you will need to make for the projection model are limited to what classes of access tokens to store, and where to store them.

For example, the Solr output connector specifically presumes that certain hardwired document attributes will be used to contain the document access tokens (`allow_token_document`), the document deny access tokens (`deny_token_document`), the document parent access tokens (`allow_token_parent`), the document parent deny access tokens (`deny_token_parent`), the share access tokens (`allow_token_share`), and the share deny access tokens (`deny_token_share`). Documents that have folder-associated access tokens are rejected for indexing by the Solr output connector at this time, because no known repository connectors use such tokens, and because including them would greatly complicate the Solr search query needed to enforce security.

The ability to reject specific documents for indexing gives an output connector the ultimate power to enforce security. This is really the only way to guarantee that, despite some oversight, documents won't fall into the wrong hands. When you design an output connector for the projection model, it's therefore very important to consider the situations in which the target repository is unable to properly project or enforce security, and appropriately block documents that would be improperly secured.

HANDLING ACCESS TOKENS IN THE TRANSLATION MODEL

But what if there is no possibility of modifying the target repository's applications to allow them to support projection as a security model? Or, perhaps, the target repository is not flexible enough to allow access tokens from ManifoldCF to be stored with each document. In both cases, we're left with only one other choice – explicitly mapping access tokens from each source repository to the corresponding ones in the target repository. This approach is called *translation*. See figure 9.2.

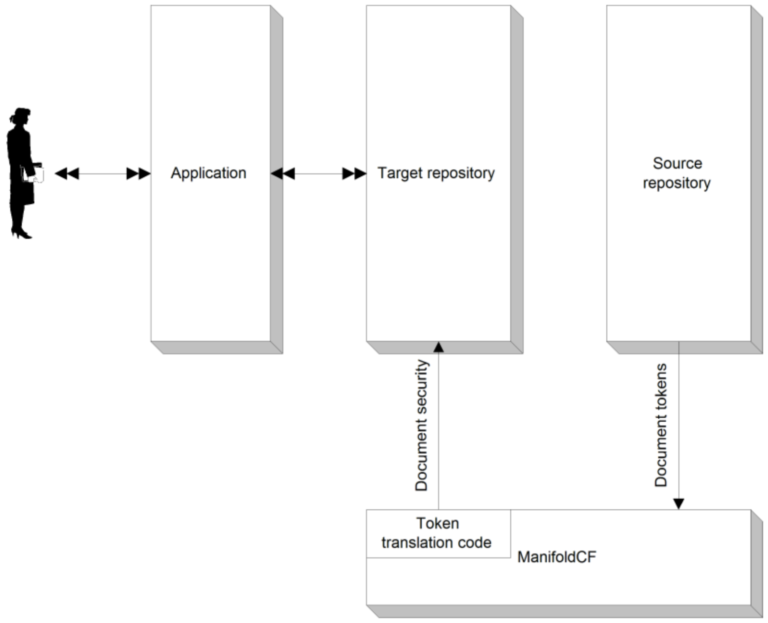


Figure 9.2 The *translation* model of security. In this model, the application is unmodified and knows nothing about ManifoldCF. The target repository only needs to concern itself with native security information. The burden of translating source repository security into target security is handled by the output connector.

In the translation model, every access token that an output connector sees must somehow be mapped to the corresponding target repository entity. If the mapping can't be done, then the output connector has a responsibility to reject the document and not hand it off to the target repository.

INCLUDING THE MAPPING IN THE OUTPUT SPECIFICATION DATA

There are several ways in which the actual mapping can be performed. The most brute force method is to allow a job to be specified in such a way as to include a list of tuples that specify the precise mapping of access token to target repository entity. An output connector has a natural way to handle this kind of job-specific configuration information, called *output specification* information.

Output specification information is similar in many respects to a repository connector's document specification information, except that instead of specifying what documents to include in the crawl, it mainly describes the mappings between repository data and the target repository. It would therefore seem perfect for keeping track of the access token translation information.

There is, however, one point that should be considered, namely the size of the data. The access tokens that can potentially be mentioned in the various documents handed to your output connector may number, in total, in the tens of thousands. Each one of them needs to be translated. Unless there are some logical rules you can specify, those access tokens represent a lot of data. All of that data needs to be processed at least once for each document. Even worse, it needs to be collected and maintained by someone.

In the case of our Docs4U output connector, we must carefully consider what form the access tokens arrive in, to see if it is possible to use rules to accomplish what we need to. The form of the access token from any individual repository connector is, however, up to that connector to determine. While the access tokens from some repositories might be in a form that is conducive to a rules-based mapping, in other cases these are simply going to be identifiers meant only for machine consumption, where the meaning of each token can be determined only by using a connector-dependent reverse lookup of the token within the original repository. This argues that the mapping must be, in general, specified by hand!

Fortunately, however, our analysis has overlooked a critically important point. The very act of building a target repository that has entities that correspond to entities in a source repository means that somebody must have either done the laborious process of writing down a mapping by hand, or have found a rules-based way to do it. Thus, either way, the materials must already be available in **some** form. The Docs4U output connector will presume that the mapping is done by rule: that is, each access token string can be directly mapped to a corresponding group or user within Docs4U. This will make our implementation convenient and compact.

EMBEDDING THE MAPPING IN THE TARGET REPOSITORY

There is a more elegant way to manage the necessary access token mapping from source to target repository. Users and groups in repositories often have the ability to have arbitrary metadata assigned to them. You can use this feature to attach access token values to target repository groups and users. Then, it's not too hard for the output connector to look up the appropriate target repository user or group given an access token.

This approach has the benefit of allowing the mapping to be managed naturally using the target repository's user and group management tools. A possible downside, however, is the time required in looking up users and groups given access tokens. A lookup done for each access token for each document may very well result in an output connector that runs fairly slowly. But performance might be improved by the use of caching.

Now that we've considered how access token mapping will work, we're ready to specify the connector's configuration information and output specification. We'll start that discussion now.

9.1.3 Specifying the connector's configuration information

An output connector's configuration information consists of that information which describes how to establish a connection with the target repository. We've seen how this is done for repository connectors in Chapter 7, and for authority connectors in Chapter 8. Output connectors are quite similar in design; indeed, the basic information needed for a connection to a given repository is likely to have only slight differences based on the type of connector it is.

The Docs4U repository thus will require only a `rootdirectory` configuration parameter, which is the path of the repository root, exactly like the repository connector and authority connector required. This is sufficient to allow a connection to be established. All other specification information will be part of the connector's output specification. We'll talk about that in the next section.

9.1.4 Choosing the connector's output specification information

An output connector's output specification information is much like a repository connector's document specification, in that it is specific to a given job. Thus, like document specifications, there are related tabs which can appear in a job's UI which allow the job's output specification information to be edited. But while a job's document specification information is centered around specifying a set of documents which are to be included as part of the job, the purpose of the output specification is a little harder to readily define.

In practice, the output specification information largely concerns the mapping of document information, such as metadata and security data, to the appropriate entities in the target repository. We've already discussed how this might include the mapping of access tokens to target security entities, but it can also include the mapping of metadata attribute names from the source repository to the target repository as well. And, finally, it's essential to allow the user to specify the name of the output attribute into which the URL of each document gets written. It is an output connector's responsibility to perform these mappings, by convention and by need.

DOCS4U OUTPUT SPECIFICATION

The Docs4U output connector will need mapping information for both security as well as for the names of metadata values. For the access token mapping, we could try to make use of the `MatchMap` class, which is what ManifoldCF's regular expression user name mapper happens to use. But this approach would be unnecessarily unfriendly for metadata, where a simple mapping of repository metadata name to target metadata name should suffice. So, for metadata we'll plan on just allowing a user to edit a more basic mapping. The only twist will be that a special node must exist to describe the attribute which will contain the URL.

The entire `MatchMap` structure for security mapping can be included in one output specification node. Let's choose a node name of `securitymap`, and a string value (representing the serialized `MatchMap` object) in the `value` attribute. The metadata mapping will consist of multiple nodes, each of them with a source attribute and target attribute. Let's choose `metadatamap`, `source`, and `target` as the node name and the two attributes, respectively. Last but certainly not least, the name of the URL metadata attribute will be specified in a node called `urlmetadataname`, and the corresponding name will be in the `value` attribute.

Now that we've laid out what the output specification needs, we can start planning how we'll implement this. That's next.

9.2 Preparing to implement an output connector

Now that we've examined all the externally-facing decisions we'll need to consider for our output connector, we'll need to look at the internally-facing ones. These decisions are driven by the code we'll actually need to write. So let's start by looking at the interface we'll need to implement, and work from there.

9.2.1 The IOutputConnector interface

All output connectors must implement the `IOutputConnector` interface, whose full class name is `org.apache.manifoldcf.agents.interfaces.IOutputConnector`. This interface extends `IConnector`, so that much of what it needs to do should now be familiar to you. The `IConnector` part of the interface includes basic configuration and connection pooling functionality as is described in Chapter 6. Table 9.1 describes the methods that are unique to `IOutputConnector`, and their meanings.

Table 9.1 IOutputConnector methods and their meanings

Method	Meaning
`getActivitiesList()`	Returns the list of the activity names which the connector will use when it records history records.
`outputSpecificationHeader()`	Output the HTML code meant for the <HEAD> section of a page, for this connector's output specification tab presentation in the Crawler UI.
`outputSpecificationBody()`	Output the HTML code meant for the <BODY> section of a page, for this connector's output specification tab presentation in the Crawler UI.
`processSpecificationPost()`	Process the form data posted by `outputSpecificationBody()` above, and modify an output specification accordingly.
`viewSpecification()`	Format output specification information into HTML for viewing.
`requestInfo()`	Called from the ManifoldCF API Service, to allow retrieval of connection-specific information, usually for the purpose of building an external user interface.
`checkMimeTypeIndexable()`	Given a mime type, respond as to whether or not that mime type can be indexed by the current output connection.

`checkDocumentIndexable()`	Given a document (as a File), determine whether or not the file can be indexed by the current output connection.
`checkLengthIndexable()`	Given a document length, determine whether or not the document can be indexed by the current output connection.
`chekcURLIndexable()`	Given a document URL, determine whether or not the document can be indexed by the current output connection.
`getOutputDescription()`	Given an output specification, convert the specification into a string that can be used for comparison to determine if documents will need to be reindexed because of output specification changes.
`addOrReplaceDocument()`	Add or replace a document in the target repository or index.
`removeDocument()`	Remove a document from the target repository or index.
`noteJobComplete()`	Perform any necessary processing that should be done when a job completes which uses the current connection.

9.2.2 What activities should be supported?

An output connector, like a repository connector, has the ability to record events that take place during the execution of a crawl, so that they appear in the crawler UI as part of the crawling history for a repository connection. It's very easy to use this feature; all you need to do is the following:

- Return the names of all the activities your connector records by means of the `getActivitiesList()` connector method
- At appropriate times, write the records corresponding to those activities through the methods provided
- Make sure that the names of your activities are limited to a maximum length of 29

Note The reason for the activity name length limitation of 29, instead of the repository connector activity name maximum of 64, is because the activity name returned by the output connector is not the complete one that will be used. The history is not attached to the output connection, but rather to the repository connection, and so multiple output connections may be involved. ManifoldCF thus converts the output connector's activity names to the form "*<activity_name>* (*<output_connection_name>*)" when they are recorded.

As with repository connectors, each of the important indexing methods – `addOrReplaceDocument()` and `removeDocument()` – receives an argument of a type

`IxxxActivity`, where the 'xxx' is a stand-in for 'OutputAdd' and 'OutputRemove', respectively. These arguments have a number of methods that your connector can use to interact with the framework, including methods that record history. If you examine the activity interfaces and the interfaces they extend, you'll eventually find the method that does that, in `IOutputHistoryActivity`:

```
public void recordActivity(Long startTime, String activityType,
  Long dataSize, String entityURI, String resultCode,
  String resultDescription)
  throws ManifoldCFException;
```

For the Docs4U output connector, it will be convenient to see an event be recorded for both the saving of a document into the repository, and the deletion of a document from the repository. We'll therefore plan on having two activities, `save` and `delete`.

9.2.3 What documents should be rejected?

An output connector has the ability to decide whether to accept any document or reject it. Rejections are permanent, in that the document that is rejected is not retried unless it is rediscovered at a later time.

The way that a connector rejects a document is by returning the appropriate status code from the `addOrReplaceDocument()` method. The status codes for this method are listed in table 9.2.

Table 9.2 Status codes that the addOrReplaceDocument() method can return

Status code	Meaning
`DOCUMENTSTATUS_ACCEPTED`	The document has been successfully submitted to the target system.
`DOCUMENTSTATUS_REJECTED`	The document could not be accepted by the target system, and this situation is not a temporary one.

Astute readers will notice that there is a third possibility, which is for the `addOrReplaceDocument()` method to throw a `ServiceInterruption` exception. As we discussed in Chapter 7, `ServiceInterruption` exceptions are used to describe transient failures, which may be resolved if sufficient time elapses. Output connectors can throw these exceptions from most `IOutputConnector` methods, in which case the document in question is re-queued, exactly as it would have been had the exception been thrown by an repository connector method.

So, what good is it for an output connector to reject a document? Usually, this ability is reserved for situations where a document cannot be accepted by the target system due to configuration reasons. Sometimes it is because the document is too large, or perhaps it is a type that the target system does not know how to process. Or it is possible that the connector cannot find out the precise reason, but the target system has signaled that the document is invalid.

Figuring out when to reject a document is important, because if your connector throws a `ServiceInterruption` exception inappropriately, ManifoldCF will retry the same document to no good end, using up resources that would be better allocated to productive crawling. Indeed, judicious use of document rejection can dramatically improve the overall throughput of the crawler, as we'll discuss later in the chapter.

In the case of the Docs4U output connector, the target repository has no restrictions on what kinds of documents it can accept. Therefore, you might naively conclude that the connector will not need to reject any documents. But that is not true, because we've forgotten about the situation where a document cannot be saved because the security information that is provided does not map to the Docs4U repository's security model. In such cases, the only correct thing to do is to reject the document. Any other response risks compromising the security of the data.

9.2.4 How much effort do the document characterization methods deserve?

Bert has looked at the `IOutputConnector` interface and has some comments. Four of the methods, `checkDocumentIndexable()`, `checkMimeTypeIndexable()`, `checkLengthIndexable()`, and `checkURLIndexable()`, seem to require quite a bit of work for him to implement. He doesn't really see the point, since the Frank-en-Stein repository will certainly let his output connector know if it tries to save a bad document.

You admit that these four *document characterization* methods require further consideration. Their purpose is to help the repository connector decide whether or not to include documents in a crawl, for the purposes of crawling efficiency. Thus, it is up to Bert to decide exactly how much effort to put into their implementation.

The `checkDocumentIndexable()` method is supposed to consider a file on disk and decide if that file can be indexed. If this method is used before the repository connector attempts to index the document, then you might save the target search engine or repository considerable effort. For example, consider a target search engine and a file system repository connection. The file system is likely to have a great deal of assorted content in it, including large un-indexable files such as movies. By blocking the indexing of such files right up front, the search engine is spared the work of copying them around, perhaps even into memory. The value of implementing this method therefore depends largely on how well the target repository deals with large documents that have no utility to it.

> **Note** There is no reason that the `checkDocumentIndexable()` method must be 100% accurate in its assessment of a document. It is perfectly acceptable for it to identify only a subset of the documents that should not be indexed. It should only rarely exclude indexable documents, however.

Unlike the `checkDocumentIndexable()` method, which requires a repository connector to fetch the document before considering it, the other three methods (`checkMimeTypeIndexable()`, `checkLengthIndexable()`, and `checkURLIndexable()`) allow repository connectors to make decisions in advance about whether to fetch a document in the first place. If the repository connector can accurately find the mime type of a document without fetching it, this method can save the overall system a huge amount of work. Once again, there is no need for 100% accuracy, as long as most of the errors committed are errors of incorrect inclusion.

For the Docs4U output connector, we are outputting documents into a general document repository. Docs4U doesn't care what the content actually is; it's all just binary data as far as it's concerned. Therefore, there's no point in document characterization; all documents should be accepted.

9.2.5 Authority group qualification of access tokens

If you recall, we discussed what an authority-group qualified access token was in Chapter 4, and why it was needed. What we didn't discuss at that time was where, exactly, that qualification process took place. It turns out that this, too, is a responsibility of the output connector.

The structure that ties all of this together is the `RepositoryDocument` object. You've seen this before in Chapter 7, where the repository connector that we wrote created a `RepositoryDocument` object, filled it in with content, metadata, and security information, and then handed it off to the `IProcessActivity` object for processing. Now, on the other end, the same `RepositoryDocument` object is handed to our output connector, whose job it is to make sense of it all and do the indexing. But the security information we receive in this object is not yet qualified by the governing authority, so the output connector must somehow do this step.

Luckily, the `addOrReplaceDocument()` method receives an extra argument just for this purpose. This argument is the name of the governing authority. Each output connector is expected to form an appropriate final qualified token using a method of the provided `IOutputAddActivity` object, namely `qualifyAccessToken()`. For a target where the security is projected, this is a crucial step which must not be overlooked if the output connector is to function correctly.

However, in the case of a target that does not use projected security such as our Docs4U output connector, this qualification step only serves to complicate matters, and conveys no benefit. We never intend to use the ManifoldCF Authority Service in such situations, so any authority qualification is completely superfluous. Our example connector code does not do this step.

9.2.6 Connector-specific database tables

As is the case for all other kinds of connectors, output connectors can have their own database tables. These are created and destroyed using the `IConnector` methods `install()` and `deinstall()`.

In most cases, output-connector-specific database tables are not needed. Indeed, as of this writing none of the standard output connectors supplied with ManifoldCF use connector-specific database tables at all. Nevertheless, it is conceivable that such a table may prove useful. For example, a database table might furnish a reasonable way of caching user identifiers for a target repository that has tens or hundreds of thousands of users and groups. Access token mapping on such a repository might otherwise perform too poorly. But, because user lookup will be more or less random, the memory-based ManifoldCF caching services might well perform poorly, requiring either too much memory, or not delivering much of a performance boost.

DOCS4U CONNECTOR DATABASE TABLES

Since it maps access tokens to user or group names and thence to user or group identifiers, the Docs4U output connector could conceivably benefit from some sort of caching of Docs4U user or group identifiers. There's no firm indication of whether to use memory-based or disk-based caching, since Docs4U is not a real repository with a real workload that we can assess. We've no idea how many users we're likely to have, for instance. Memory-based caching would certainly be faster, but if there are a large number of users the cache might not be large enough to help sufficiently. But since it could go either way, for demonstration purposes we're going to write our connector to use a database table to cache the Docs4U mapping of users and groups to identifiers.

The connector-specific table we'll create will be called `usergrouplookup`, and it will have the columns as described in table 9.3.

Table 9.3 The usergrouplookup schema

Column	Purpose
`repositoryroot`	The path to the repository which a record is associated with.
`usergroupname`	The name of the user or group.
`usergroupid`	The id of the user or group.
`expirationtime`	The time, in milliseconds since January 1, 1970, when the record is no longer valid.

Since this table will tend to grow over time, the connector's `poll()` method will be used to remove expired records at regular intervals. This will have the effect of limiting the table size to only those users and groups that are currently active. But there will be no limit to the number of those users.

Now that we've made the decision to use a database table for user and group ID caching, we've made all that internally-facing decisions we need to. We're now ready to begin coding.

9.3 Coding the connector

Coding your output connector is similar in many respects to coding repository connectors or authority connectors. Like we did in Chapter 7 and Chapter 8, we're going to create a class that implements `IOutputConnector` which extends a standard base class, in this case `org.apache.manifoldcf.agents.output.BaseOutputConnector`. The base class provides standard implementations of all the `IConnector` methods, which will save us a lot of time.

There is one major difference between writing an output connector and writing a repository or authority connector. Specifically, output connectors effectively run as part of the Agents process, but are not part of the Pull Agent. They can thus technically be shared by all registered agents, not just the Pull Agent. We'll discuss this architecture in detail in Chapter 10.

> **Note** This means that your output connector should not have any dependencies on classes in `mcf-pull-agent.jar`, which includes the packages `org.apache.manifoldcf.crawler` and `org.apache.manifoldcf.authorities`. Violating this dependency limitation may cause your connector class to not be loadable in future versions of ManifoldCF.

Let's start with writing the `IConnector` methods, and trying those out.

9.3.1 Writing the IConnector methods

Table 7.3 describes the `IConnector` methods provided by the base class implementation. As per the discussion so far, we'll need to override the `install()`, `deinstall()`, `connect()`, `disconnect()`, `poll()`, `setThreadContext()`, `clearThreadContext()`, and `check()` methods in the Docs4U output connector. We will also need implementations for `outputConfigurationHeader()`, `outputConfigurationBody()`, `processConfigurationPost()`, and `viewConfiguration()`. Many of these methods are substantially identical to the same methods in the Docs4U repository connector, such as the configuration UI methods, `connect()`, `disconnect()`, and `check()`, so I will save you some tedium and not include those in the listing. You can see the entire class at http://manifoldcfinaction/svn/examples/edition_2_revised/output_connector_example/src/org/apache/manifoldcf/agents/output/docs4u/Docs4UOutputConnector.java. Please examine listing 9.1.

Listing 9.1 IConnector-related method implementations for the Docs4U output connector, partial

```
protected UserGroupLookupManager userGroupLookupManager = null;         #1
protected ILockManager lockManager = null;                              #2

public void install(IThreadContext threadContext)                       #3
  throws ManifoldCFException                                            #3
{
  super.install(threadContext);
  new UserGroupLookupManager(threadContext).initialize();               #4
}

public void deinstall(IThreadContext threadContext)                     #5
  throws ManifoldCFException                                            #5
{
  new UserGroupLookupManager(threadContext).destroy();                  #6
  super.deinstall(threadContext);
}

public void clearThreadContext()
{
  userGroupLookupManager = null;                                        #7
  lockManager = null;                                                   #7
  super.clearThreadContext();
}

public void setThreadContext(IThreadContext threadContext)
  throws ManifoldCFException
{
  super.setThreadContext(threadContext);
  userGroupLookupManager = new UserGroupLookupManager(threadContext);  #8
  lockManager = LockManagerFactory.make(threadContext);                 #8
}

protected Docs4UAPI getSession()
  throws ManifoldCFException, ServiceInterruption
{
  if (session == null)
  {
    try
    {
      session = D4UFactory.makeAPI(rootDirectory);
    }
    catch (D4UException e)
    {
      Logging.ingest.warn("Docs4U: Session setup error: "+              #9
        e.getMessage(),e);                                              #9
      throw new ManifoldCFException("Session setup error: "+
        e.getMessage(),e);
    }
  }
  // Reset the expiration time
  sessionExpiration = System.currentTimeMillis() +
    SESSION_EXPIRATION_MILLISECONDS;
  return session;
}
```

#1 Handle to the class that manages the database table
#2 Handle to the lock manager
#3 Override the install method
#4 Create the database table and indexes
#5 Override the deinstall method
#6 Destroy the database table and indexes
#7 Null out all thread-specific objects
#8 Create a thread-specific manager instance
#9 Use the "ingest" logger for output connector logging

At #1, we define the class member variable that will point to an appropriate table manager for the user and group lookup table, when present. We also have a handle to the lock manager, at #2, which will be used later to prevent a race condition. We override the `install()` method at #3, augmenting it to create the database table an indexes at #4. Similarly, we override the `deinstall()` method at #5, destroying the table and indexes at #6. When the thread context is cleared, we must be sure to clear out the member variables that were created with it, at #7. The creation actually takes place at #8. Finally, when we do logging, the proper logger to use with output connectors is the `Logging.ingest` logger.

The database portion of the code relies on a database manager class to perform the actual database table management. Portions of this class can be seen in listing 9.2. The entire class can be found at http://manifoldcfinaction.googlecode.com/svn/examples/edition_2_revised/output_connector_example/src/org/apache/manifoldcf/agents/output/docs4u/UserGroupLookupManager.java.

Listing 9.2 Database table creation and destruction methods from UserGroupLookupManager class

```
protected final static String DATABASE_TABLE_NAME = "usergrouplookup"; #A

protected final static String repositoryRootField = "repositoryroot";  #B
protected final static String userGroupNameField = "usergroupname";    #B
protected final static String userGroupIDField = "usergroupid";        #B
protected final static String expirationTimeField = "expirationtime";  #B

protected IThreadContext threadContext;

public UserGroupLookupManager(IThreadContext threadContext)
  throws ManifoldCFException
{
  super(DBInterfaceFactory.make(threadContext,                         #C
      ManifoldCF.getMasterDatabaseName(),                              #C
      ManifoldCF.getMasterDatabaseUsername(),                          #C
      ManifoldCF.getMasterDatabasePassword()),                         #C
    DATABASE_TABLE_NAME);                                              #C
  this.threadContext = threadContext;
}

public void initialize()
  throws ManifoldCFException
{
  Map<String,ColumnDescription> columnMap =
        new HashMap<String,ColumnDescription>();
      columnMap.put(repositoryRootField,
        new ColumnDescription("VARCHAR(255)",false,false,null,null,false));
      columnMap.put(userGroupNameField,
        new ColumnDescription("VARCHAR(255)",false,false,null,null,false));
      columnMap.put(userGroupIDField,
        new ColumnDescription("VARCHAR(32)",false,false,null,null,false));
      columnMap.put(expirationTimeField,
        new ColumnDescription("BIGINT",false,false,null,null,false));
      performCreate(columnMap,null);                                     #D

      performAddIndex(null,new IndexDescription(true,                    #E
        new String[]{repositoryRootField,userGroupNameField}));          #E
      performAddIndex(null,new IndexDescription(false,                   #F
        new String[]{expirationTimeField}));                             #F
    }

    public void destroy()
      throws ManifoldCFException
    {
      performDrop(null);                                                 #G
    }
```

#A The name of the table
#B The schema fields
#C Create table manager using standard database handle
#D Create the table
#E Unique index on repository root and user name
#F Non-unique index on expiration time
#G Drop the table when told to do so

TRYING IT OUT

As usual, we'll make sure that the configuration part of the UI works before we code anything else. First, build the Docs4U output connector example, and install the sample target repository, using the following commands:

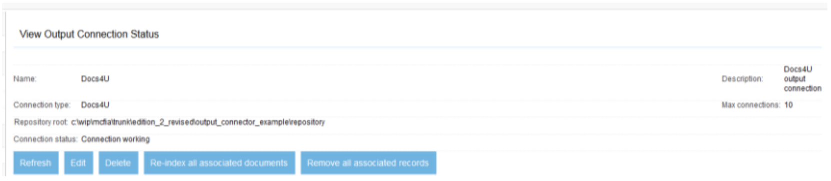
```
   cd output_connector_example
   ant jar
   ant sample-repository
```

Next, copy `lib/docs4u-example.jar` and `build/jar/d4u-output-connector.jar` to the `dist/example/connector-lib` folder of your ManifoldCF installation. Modify the `dist/example/connectors.xml` file to include the following line:

```
     <outputconnector name="Docs4U"
class="org.apache.manifoldcf.agents.output.docs4u.Docs4UOutputConnector"/>
```

Finally, start ManifoldCF. It should start without any errors, but if there are any, please remember that it is during this time, when the connector is being unregistered and reregistered, when the connector's database tables are dropped and created.

Once started, you should be able to enter the Crawler UI with a browser, and navigate to the `List Output Connections` link. Create an output connector, give it a name, click the `Type` tab, and select `Docs4U` from the connection type pull-down. Then click the `Continue` button. The `Repository` tab should appear. Click on it, and enter the full path to your sample target repository, which will be the `repository` folder underneath the directory `output_connector_example`. Then, click the `Save` button. If you did everything correctly, you should see something similar to figure 9.3.


View Output Connection Status

Name: Docs4U
Description: Docs4U output connection
Connection type: Docs4U
Max connections: 10
Repository root: c:\wip\mcfia\trunk\edition_2_revised\output_connector_example\repository
Connection status: Connection working

Refresh
Edit
Delete
Re-index all associated documents
Remove all associated records

Figure 9.3 A successfully set up Docs4U output connection.

Congratulations! You've now demonstrated that the configuration portion of the Docs4U output connector crawler UI methods are working! Next, we'll tackle the `IOutputConnector` specification UI methods.

9.3.2 Coding the IOutputConnector output specification UI methods

The overall structure of the `IOutputConnector` output specification UI portion of the connector class is similar in many respects to the document specification UI portion we wrote for the Docs4U repository connector back in Chapter 7. See listing 9.3. In the interests of brevity I've omitted the methods that are used solely for managing the access mapping tab.

Listing 9.3 The Docs4U connector IOutputConnector output specification selected UI methods

```
protected final static String NODE_SECURITY_MAP = "securitymap";       #1
protected final static String NODE_METADATA_MAP = "metadatamap";       #1
protected final static String NODE_URL_METADATA_NAME =                 #1
  "urlmetadataname";                                                   #1
                                                                       #1
protected final static String ATTRIBUTE_VALUE = "value";               #1
protected final static String ATTRIBUTE_SOURCE = "source";             #1
protected final static String ATTRIBUTE_TARGET = "target";             #1

public void outputSpecificationHeader(IHTTPOutput out, Locale locale,
  OutputSpecification os, List<String> tabsArray)
  throws ManifoldCFException, IOException
{
  tabsArray.add("Docs4U Metadata");                                    #2
  tabsArray.add("Docs4U Security");                                    #2
  Messages.outputResourceWithVelocity(out,locale,                      #3
    "SpecificationHeader.html",null);                                  #3
}
```

```
public void outputSpecificationBody(IHTTPOutput out, Locale locale,
  OutputSpecification os, String tabName)
  throws ManifoldCFException, IOException
{
  outputMetadataMappingTab(out,locale,os,tabName);                     #4
  outputAccessMappingTab(out,locale,os,tabName);                       #5
}

public String processSpecificationPost(IPostParameters variableContext,
  OutputSpecification os)
  throws ManifoldCFException
{
  String rval = processMetadataMappingTab(variableContext,os);         #6
  if (rval != null)
    return rval;
  rval = processAccessMappingTab(variableContext,os);                  #7
  return rval;
}

 public void viewSpecification(IHTTPOutput out, Locale locale,
   OutputSpecification os)
   throws ManifoldCFException, IOException
 {
   Map<String,Object> velocityContext = new HashMap<String,Object>();
   fillInMetadataMappingTab(velocityContext,os);                       #8
   fillInAccessMappingTab(velocityContext,os);                         #9
   Messages.outputResourceWithVelocity(out,locale,                    #10
     "SpecificationView.html",velocityContext);                       #10
 }

 protected void outputMetadataMappingTab(IHTTPOutput out,
   Locale locale, OutputSpecification os, String tabName)
   throws ManifoldCFException, IOException
 {
   Map<String,Object> velocityContext = new HashMap<String,Object>();
   velocityContext.put("TabName",tabName);
   fillInMetadataMappingTab(velocityContext,os);                      #11
   fillInMetadataMappingTabSelection(velocityContext);                #12
   Messages.outputResourceWithVelocity(out,locale,                    #13
     "Specification_Docs4U_Metadata.html",velocityContext);           #13
 }

 protected static void fillInMetadataMappingTab(
   Map<String,Object> velocityContext, OutputSpecification os)
 {
   String urlMetadataName = "";
   List<Map<String,String>> mappings =
     new ArrayList<Map<String,String>>();
   Set<String> usedAttributes = new HashSet<String>();

   int i = 0;                                                         #14
   while (i < os.getChildCount())                                     #14
```

```
    {
      SpecificationNode sn = os.getChild(i++);
      if (sn.getType().equals(NODE_URL_METADATA_NAME))                 #15
        urlMetadataName = sn.getAttributeValue(ATTRIBUTE_VALUE);
      else if (sn.getType().equals(NODE_METADATA_MAP))                 #16
      {
        String metadataRecordSource =
          sn.getAttributeValue(ATTRIBUTE_SOURCE);
        String metadataRecordTarget =
          sn.getAttributeValue(ATTRIBUTE_TARGET);
        usedAttributes.add(metadataRecordTarget);                      #17
        Map<String,String> row = new HashMap<String,String>();
        row.put("source",metadataRecordSource);                        #18
        row.put("target",metadataRecordTarget);                        #18
        mappings.add(row);                                             #18
      }
    }
    velocityContext.put("urlmetadataname",urlMetadataName);            #19
    velocityContext.put("metadatarecords",mappings);                   #19
    velocityContext.put("usedattributes",usedAttributes);              #19
  }

  protected void fillInMetadataMappingTabSelection(
    Map<String,Object> velocityContext)
  {
    try
    {
      String[] matchNames = getMetadataNames();                        #20
      velocityContext.put("urlmetadataattributes",matchNames);         #21
      velocityContext.put("error","");                                 #22
    }
    catch (ManifoldCFException e)
    {
      velocityContext.put("error","Error: "+e.getMessage());           #23
    }
    catch (ServiceInterruption e)
    {
      velocityContext.put("error","Transient error: "+e.getMessage()); #24
    }
  }

  protected String processMetadataMappingTab(
    IPostParameters variableContext, OutputSpecification os)
    throws ManifoldCFException
  {
    removeNodes(os,NODE_URL_METADATA_NAME);                            #25
    String urlMetadataName =                                           #26
      variableContext.getParameter("ocurlmetadataname");               #26
    if (urlMetadataName != null)                                       #27
      addUrlMetadataNameNode(os,urlMetadataName);                      #27

    removeNodes(os,NODE_METADATA_MAP);                                 #28

    String recordCountString =                                         #29
```

```
      variableContext.getParameter("ocmetadatacount");                 #29
    if (recordCountString != null)                                      #30
    {
      int recordCount = Integer.parseInt(recordCountString);           #31

      int i = 0;                                                        #32
      while (i < recordCount)                                           #32
      {
        String suffix = "_"+Integer.toString(i++);
        // Only add the name/value if the item was not deleted.
        String metadataOp =                                             #33
          variableContext.getParameter("ocmetadataop"+suffix);          #33
        if (metadataOp == null || !metadataOp.equals("Delete"))         #34
        {
          String metadataSource =
            variableContext.getParameter("ocmetadatasource"+suffix);
          String metadataTarget =
            variableContext.getParameter("ocmetadatatarget"+suffix);
          addMetadataMappingNode(os,metadataSource,metadataTarget);
        }
      }
    }

    String operation = variableContext.getParameter("ocmetadataop");   #35
    if (operation != null && operation.equals("Add"))                   #36
    {
      String metadataSource =
        variableContext.getParameter("ocmetadatasource");
      String metadataTarget =
        variableContext.getParameter("ocmetadatatarget");
      addMetadataMappingNode(os,metadataSource,metadataTarget);
    }

    return null;
  }

  protected static void addMetadataMappingNode(OutputSpecification os,
    String metadataSource, String metadataTarget)
  {
    SpecificationNode sn = new SpecificationNode(NODE_METADATA_MAP);
    sn.setAttribute(ATTRIBUTE_SOURCE,metadataSource);
    sn.setAttribute(ATTRIBUTE_TARGET,metadataTarget);
    os.addChild(os.getChildCount(),sn);
  }

  protected static void addUrlMetadataNameNode(OutputSpecification os,
    String urlMetadataName)
  {
    SpecificationNode sn = new SpecificationNode(NODE_URL_METADATA_NAME);
    sn.setAttribute(ATTRIBUTE_VALUE,urlMetadataName);
    os.addChild(os.getChildCount(),sn);
  }
```

```
protected static void removeNodes(OutputSpecification os,
  String nodeTypeName)
{
  int i = 0;
  while (i < os.getChildCount())
  {
    SpecificationNode sn = os.getChild(i);
    if (sn.getType().equals(nodeTypeName))
      os.removeChild(i);
    else
      i++;
  }
}
```

#1 Declare specification node names and attributes
#2 A tab for security and a tab for metadata
#3 Output Javascript using Velocity template
#4 Output metadata tab body
#5 Output security tab body
#6 Process metadata tab variables
#7 Process security tab variables
#8 Fill in Metadata Mapping values
#9 Fill in Access Mapping values
#10 Output view template using Velocity
#11 Fill in Metadata Mapping values
#12 Get the set of metadata names from repository
#13 Output editing template using Velocity
#14 Scan all outer-level SpecificationNodes
#15 If URL metadata node, record the value
#16 If mapping node, create a record
#17 Also keep track of which metadata names have already been used
#18 Add a the new record to the record list
#19 Put records, used attributes, and URL metadata name into Velocity context
#20 Get metadata names from repository
#21 Put names into Velocity context
#22 Set error message to empty
#23 If exception, set error message accordingly
#24 Don't forget to handle ServiceInterruption also
#25 Remove old URL metadata nodes
#26 Look for url metadata name in POST data
#27 If data found, create a URL metadata node
#28 Remove all existing mapping nodes
#29 Look for the record count variable in POST data
#30 If found, create new mapping nodes
#31 Parse the count variable
#32 Loop through form data records
#33 Look for the operation form variable
#34 If the operation is not "Delete", then add a mapping node
#35 Look for the global operation form variable
#36 If the global operation is "Add", the add a new mapping node at end

The specification nodes and attributes are declared as constants at #1. At #2, we move on to the `outputSpecificationHeader()` method, and specify the tabs we'll present –

one for security, one for metadata, both adhering to the output connector tab naming standards, and we use Velocity to render the Javascript for both tabs at #3.

The `outputSpecificationBody()` method is next. At #4, we call a protected method to output the `Docs4U Metadata` tab body, and at #5 we do the same for the `Docs4U Security` tab.

Processing the tab form data happens in the `processSpecificationPost()` method. At #6, we process the data for the `Docs4U Metadata` tab, and at #7 we do the same for the `Docs4U Security` tab.

The `viewSpecification()` method outputs all the tabs using one template. At #8 we fill in the Velocity context data for the `Docs4U Metadata` tab, and do the same for the `Docs4U Security` tab at #9. Then we render both tabs using a Velocity template at #10.

Rendering the edit page for the `Docs4U Metadata` tab takes place beginning at #11, where we fill the Velocity context with data from the document specification. For this tab, we also need the full set of metadata names from the repository, so at #12 we call a protected method to obtain that as well. At #13 we render the tab using a Velocity template.

Filling in the Velocity context with the `Docs4U Metadata` tab's document specification information takes place at #14, where we start off by scanning all the outer-level `SpecificationNode` objects. If we see a URL metadata node (at #15), we grab its value. If we see a mapping node (at #16) we get its values. We add the metadata name to the set of metadata names that are in use, at #17. At #18 we create the new record and add it to a list. Finally, at #19, we add all the structures we created into the Velocity context.

The next method places the current list of metadata names into the Velocity context starting at #20, where we try to obtain the list from the repository. If successful, we put the names into the context, at #21, along with an empty error value, at #22. If we fail, then we set the error value to something that is non-empty, at #23 and #24.

Processing the POST data is next. At #25, we remove old URL metadata nodes from the document specification, and at #26 we look for a URL metadata value that has been posted. If we find one, then at #27 we create a new `SpecificationNode` representing it and add it to the document specification. Similarly, we remove all mapping nodes at #28, then we look for a posted mapping count variable at #29. If we find it, at #30, we parse it at #31, and then scan the posted records at #32. For each record, we first look for the record operation at #33, and if it is not a "Delete", we add a `SpecificationNode` representing the mapping, at #34. Then, we look for the global operation form variable at #35, and if it is "Add", we create a new `SpecificationNode` representing the added mapping, at #36.

The Velocity templates for the two tabs are significantly more complex than any we've seen so far. In an effort to save space, I've chosen to subsequently present only the code that supports for the `Docs4U Metadata` tab. We'll start with the template that outputs the Javascript that this tab needs. See listing 9.4.

Listing 9.4 The SpecificationHeader.html Velocity template

```
<script type="text/javascript">
<!--

function checkOutputSpecification()
{
  if (ocCheckMetadataMappingTab() == false)                              #A
    return false;
  if (ocCheckAccessMappingTab() == false)                                #B
    return false;
  return true;
}

function checkOutputSpecificationForSave()
{
  if (ocCheckMetadataMappingTabForSave() == false)                       #C
    return false;
  if (ocCheckAccessMappingTabForSave() == false)                         #D
    return false;
  return true;
}

function ocSpecOp(n, opValue, anchorvalue)                               #E
{
  eval("editjob."+n+".value = \""+opValue+"\"");
  postFormSetAnchor(anchorvalue);
}

function ocCheckMetadataMappingTab()                                     #F
{                                                                        #F
  return true;                                                           #F
}                                                                        #F

function ocCheckMetadataMappingTabForSave()                              #G
{                                                                        #G
  if (editjob.ocurlmetadataname.value == "")                             #G
  {                                                                      #G
    alert("URL metadata name cannot be blank.");                         #G
    SelectTab("Docs4U Metadata");                                        #G
    editjob.ocmetadataname.focus();                                      #G
    return false;                                                        #G
  }                                                                      #G
  return true;                                                           #G
}                                                                        #G

function ocMetadataDelete(n)                                             #H
{
  ocSpecOp("ocmetadataop_"+n, "Delete", "ocmetadata_"+n);                #I
}

function ocMetadataAdd(n)                                                #J
{
  if (editjob.ocmetadatasource.value == "")                              #K
  {
    alert("Metadata source name cannot be blank.");
```

```
      editjob.ocmetadatasource.focus();
      return;
    }
    if (editjob.ocmetadatatarget.value == "")                            #L
    {
      alert("Please select a metadata target value first.");
      editjob.ocmetadatatarget.focus();
      return;
    }
    if (editjob.ocmetadatatarget.value == editjob.ocurlmetadataname.value) #M
    {
      alert("URL metadata name must differ from all mapping metadata "+
        "names.");
      editjob.ocmetadatatarget.focus();
      return;
    }
    ocSpecOp("ocmetadataop", "Add", "ocmetadata_"+n);                    #N
  }

  function ocCheckAccessMappingTab()                                     #O
  {                                                                      #O
    if (editjob.ocsecurityregexp.value != "" &&                          #O
      !isRegularExpression(editjob.ocsecurityregexp.value))              #O
    {                                                                    #O
      alert("Security mapping regular expression must be a valid regular "+#O
        "expression");                                                   #O
      editjob.ocsecurityregexp.focus();                                  #O
      return false;                                                      #O
    }                                                                    #O
    return true;                                                         #O
  }                                                                      #O

  function ocCheckAccessMappingTabForSave()                              #P
  {                                                                      #P
    return true;                                                         #P
  }                                                                      #P

  //-->
  </script>
```

#A Check if Metadata Mapping tab okay with repost
#B Check if Access Mapping tab okay with repost
#C Check if Metadata Mapping tab okay with save
#D Check if Access Mapping tab okay with save
#E Helper method that sets form variable and reposts
#F Metadata Mapping tab always allows repost
#G Metadata Mapping tab only blocks save if metadata name is empty
#H Method called when Delete button is clicked
#I Set the appropriate form variable to signal deletion, and post
#J Method called when Add button is clicked
#K Source can't be blank
#L Target can't be blank
#M Target can't be the same as the metadata name
#N If all is OK, set form variable to signal addition, and post
#O Access Mapping tab blocks repost if security regexp is illegal

#P Access Mapping tab never blocks save

Next, let’s have a look at the Velocity template that renders the metadata tab’s HTML. This is fairly long because I’ve tried to limit the user’s choices to the minimal set of allowed possibilities within the set of target repository attributes. See listing 9.5.

Listing 9.5 The Specification_Docs4U_Metadata.html Velocity template

```
#if($TabName == 'Docs4U Metadata')                                         #A

<table class="displaytable">
  <tr><td class="separator" colspan="2"><hr/></td></tr>

  #if($error != '')                                                        #B

  <tr class="formrow"><td class="message" colspan="2">
    $Encoder.bodyEscape($error)</td></tr>

  #else                                                                    #C

  <tr>
    <td class="description"><nobr>Docs4U URL attribute:</nobr></td>
    <td class="value">
      <select name="ocurlmetadataname">
    #if($urlmetadataname == '')                                            #D
        <option value="" selected="true">                                  #D
    #else                                                                  #D
        <option value="">                                                  #D
    #end                                                                   #D
          -- Select URL metadata attribute --
        </option>

    #foreach($urlmetadataattribute in $urlmetadataattributes)              #E

      #if($urlmetadataattribute == $urlmetadataname)                       #F
        <option value="$Encoder.attributeEscape($urlmetadataattribute)"    #F
          selected="true">                                                 #F
      #else                                                                #F
        <option value="$Encoder.attributeEscape($urlmetadataattribute)">  #F
      #end                                                                 #F
          $Encoder.bodyEscape($urlmetadataattribute)
        </option>

    #end

      </select>
    </td>
  </tr>

  #end

  <tr>
    <td class="description"><nobr>Mapping:</nobr></td>                    #G
    <td class="boxcell">
      <table class="formtable">
```

```
       <tr class="formheaderrow">
         <td class="formcolumnheader"></td>
         <td class="formcolumnheader">
           <nobr>Source metadata name</nobr></td>
         <td class="formcolumnheader">
           <nobr>Docs4U metadata name</nobr></td>
       </tr>

 #set($recordpresent = false)                                        #H
 #set($k = 0)                                                        #H

 #foreach($metadatarecord in $metadatarecords)                       #I

   #if($velocityCount % 2 == 0)                                      #J
       <tr class="oddformrow">                                       #J
   #else                                                             #J
       <tr class="evenformrow">                                      #J
   #end                                                              #J
         <td class="formcolumncell">
           <input type="hidden" name="ocmetadataop_$k" value=""/>    #K
           <input type="hidden" name="ocmetadatasource_$k"
        value="$Encoder.attributeEscape($metadatarecord.get('source'))"/>
           <input type="hidden" name="ocmetadatatarget_$k"
        value="$Encoder.attributeEscape($metadatarecord.get('target'))"/>
           <a name="metadata_$k">
             <input type="button" value="Delete"                     #L
               onClick='Javascript:ocMetadataDelete("$k")'           #L
               alt="Delete mapping #$k"/>                            #L
           </a>
         </td>
         <td class="formcolumncell">
           <nobr>
             $Encoder.bodyEscape($metadatarecord.get('source'))
           </nobr>
         </td>
         <td class="formcolumncell">
           <nobr>
             $Encoder.bodyEscape($metadatarecord.get('target'))
           </nobr>
         </td>
       </tr>

   #set($recordpresent = true)                                       #M
   #set($k = $k + 1)                                                 #N
 #end

 #if(!$recordpresent)                                                #O
       <tr class="formrow"><td class="formcolumnmessage" colspan="3">  #O
         No mappings specified</td></tr>                             #O
 #end                                                                #O

       <tr class="formrow"><td class="formseparator" colspan="3">
         <hr/></td></tr>

 #if($error != '')                                                   #P
```

```
          <tr class="formrow"><td class="message" colspan="3">
            $Encoder.bodyEscape($error)</td></tr>

    #else                                                               #Q
      #set($nextk = $k + 1)

          <tr class="formrow">
            <td class="formcolumncell">
              <nobr>
                <a name="find_$k">
                  <input type="button" value="Add"                      #R
                    onClick='Javascript:ocMetadataAdd("$nextk")'        #R
                    alt="Add new mapping"/>                             #R

                  <input type="hidden" name="ocmetadatacount" value="$k"/> #S
                  <input type="hidden" name="ocmetadataop" value=""/>      #S

                </a>
              </nobr>
            </td>
            <td class="formcolumncell">
              <nobr>
                <input type="text" size="32" name="ocmetadatasource"
                  value=""/>
              </nobr>
            </td>
            <td class="formcolumncell">
              <select name="ocmetadatatarget">
                <option value="" selected="true">
                  --Select target attribute name --</option>

      #foreach($urlmetadataattribute in $urlmetadataattributes)         #T

        #if($usedattributes.contains($urlmetadataattribute) == false)   #U

                <option
                  value="$Encoder.attributeEscape($urlmetadataattribute)">
                  $Encoder.bodyEscape($urlmetadataattribute)
                </option>
        #end

      #end

              </select>
            </td>
          </tr>

    #end

        </table>
      </td>
    </tr>
  </table>

  #else                                                                 #V
```

```
    #set($k = 0)
    #foreach($metadatarecord in $metadatarecords)

  <input type="hidden" name="ocmetadatasource_$k"
    value="$Encoder.attributeEscape($metadatarecord.get('source'))"/>
  <input type="hidden" name="ocmetadatatarget_$k"
    value="$Encoder.attributeEscape($metadatarecord.get('target'))"/>
      #set($k = $k + 1)

    #end
  <input type="hidden" name="ocmetadatacount" value="$k"/>
  <input type="hidden" name="ocurlmetadataname"
    value="$Encoder.attributeEscape($urlmetadataname)"/>

  #end
```

#A Look for the "Docs4U Metadata" tab
#B If there was an error looking up metadata names, print it
#C If no error, output the visible form
#D Selecting the URL attribute: Decide whether to pre-select the 'unchosen' value
#E Loop over the set of metadata attributes
#F Pre-select the included metadata attributes
#G Set up the mapping table
#H Set counter and 'seen something' flag
#I Loop over the mapping records
#J Decide on the row format based on the loop counter
#K Include an operation hidden variable for each row
#L Include a Delete button for each row
#M Increment counter
#N Note that we saw something
#O Print a special row if we had no data at all
#P Prepare to present Add button: If error find metadata names, print it
#Q If no error, list the remaining attributes
#R Include an Add button
#S Include an operation hidden variable, and the total count of mappings
#T Build selection pulldown by looping over metadata names
#U Exclude the names we've already selected
#V If this isn't the "Docs4U Metadata" tab, output hidden form variables instead

Finally, let's look at the Velocity template that renders the view page. See listing 9.7.

Listing 9.7 The SpecificationView.html Velocity template

```
<table class="displaytable">

  <tr>
    <td class="description"><nobr>Docs4U URL attribute:</nobr></td>       #A
    <td class="value">                                                    #A
      <nobr>$Encoder.bodyEscape($urlmetadataname)</nobr>                  #A
    </td>                                                                 #A
  </tr>

  <tr>
```

```
      <td class="description"><nobr>Mappings:</nobr></td>                  #B
      <td class="boxcell">
        <table class="formtable">
          <tr class="formheaderrow">
            <td class="formcolumnheader"><nobr>Source attribute</nobr></td>
            <td class="formcolumnheader"><nobr>Docs4U attribute</nobr></td>
          </tr>
#foreach($metadatarecord in $metadatarecords)                             #C
  #if($velocityCount % 2 == 0)                                             #D
          <tr class="oddformrow">                                          #D
  #else                                                                    #D
          <tr class="evenformrow">                                         #D
  #end                                                                     #D
            <td class="formcolumncell">
              <nobr>
                $Encoder.bodyEscape($metadatarecord.get('source'))
              </nobr>
            </td>
            <td class="formcolumncell">
              <nobr>
                $Encoder.bodyEscape($metadatarecord.get('target'))
              </nobr>
            </td>
          </tr>
#end
        </table>
      </td>
    </tr>

    <tr>
      <td class="description"><nobr>Docs4U security mapping:</nobr></td>   #E
      <td class="value">                                                   #E
        <nobr>$Encoder.bodyEscape($regexp) ==&gt;                          #E
          $Encoder.bodyEscape($translation)</nobr>                         #E
      </td>                                                                #E
    </tr>

  </table>
```

#A Output the URL attribute name, using description/value format
#B Begin the 'Mappings' table
#C Loop over the metadata mapping records
#D Select the row style based on the loop counter
#E Output the security mapping regular expression and translation

KEEPING THE NAME SPACES FROM COLLIDING

In the example code, you may have noticed that the output connector tab names we used included "Docs4U" in the name, and that many Javascript methods and form elements begin with the letters "oc". This is not by accident.

Tabs from the output connector and from the repository connector must coexist in the same page, the page that displays the job. But each connector may contribute Javascript methods, tab names, and named form elements independently of the other. They may well be written by a different person, and thus have little coordination between the two. How, then, should we avoid conflicts?

The simple answer is that there's no perfect way. For Javascript methods and form element names, a prefix system can be used which should guarantee that no conflicts between repository connectors and output connectors. But for tab names, it is harder, because any prefix will show up in the UI itself, and potentially confuse a user. So I've developed what I hope is a decent set of conventions that address the concerns. It is the responsibility of the output connector to properly apply the conventions to avoid conflicts. The conventions are summarized as follows:

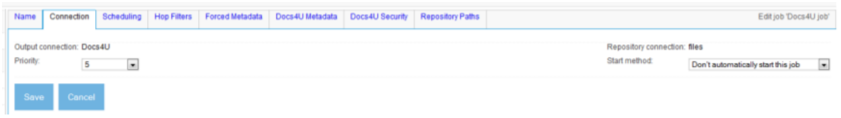
- All Javascript method names will begin with lowercase `oc`.
- All form element names will begin with lowercase `oc`.
- Tab names should include some identifying string in each one, such as the target repository type, e.g. `Solr Metadata`

Of course, this will only work if repository connectors avoid doing the same thing. Luckily, all of these rules are fairly unnatural, so it is unlikely that any repository connector would be coded this way purely by accident. And, if the repository connector writer **knows** the rules, it should never happen at all.

TESTING THE OUTPUT SPECIFICATION UI

We've completed the code for the output specification data! Now it is time to try it out. We've already created a Docs4U output connection, so the next step will be to create a job that uses the output connection, and see how that works.

If you haven't already, start ManifoldCF. For this test, you'll need to create a repository connection, or use one of the ones we created earlier. Then, when you are done, click the `List All Jobs` navigation link, and click the `Add a new job` link on that page. Fill in a job name, and click the `Connections` tab. Select the `Docs4U` output connection, and whatever repository connection you set up earlier. Then click the `Continue` button. You should see a screen looking something like figure 9.4.

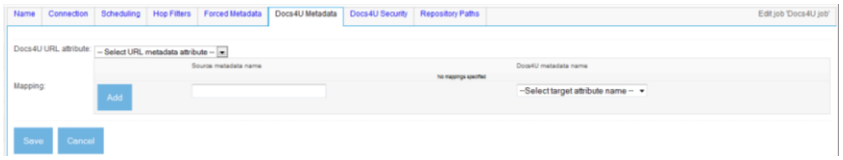


Edit job 'Docs4U job'

Output connection: Docs4U
Priority: 5
Repository connection: files
Start method: Don't automatically start this job

Figure 9.4 The Docs4U output connector tabs appear.

The two Docs4U output connector tabs have appeared! Let's see what each one looks like. Click on the `Docs4U Metadata` tab first. You should see something like figure 9.5.


Name | Connection | Scheduling | Hop Filters | Forced Metadata | Docs4U Metadata | Docs4U Security | Repository Paths — Edit job 'Docs4U job'

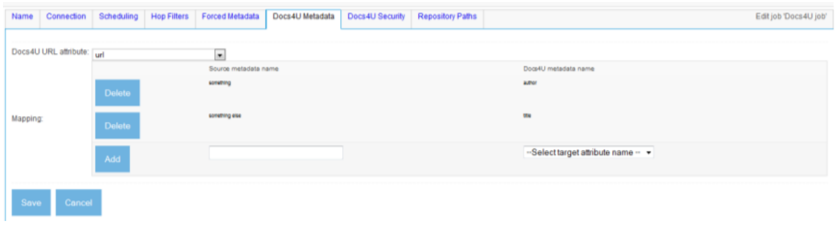
Docs4U URL attribute: -- Select URL metadata attribute --

Mapping:	Source metadata name		Docs4U metadata name
		No mappings specified	
Add			--Select target attribute name --

Save Cancel

Figure 9.5 The Docs4U Metadata tab, in its initial state. Note there are two fields – one for the URL attribute, and one for the mapping table.

Let's go ahead and choose the Docs4U repository's `url` attribute for the URL metadata attribute name selection. Let's also fill in a mapping or two. It's okay to supply non-existent source attributes. When you are done, you might see something like figure 9.6.


Name | Connection | Scheduling | Hop Filters | Forced Metadata | Docs4U Metadata | Docs4U Security | Repository Paths — Edit job 'Docs4U job'

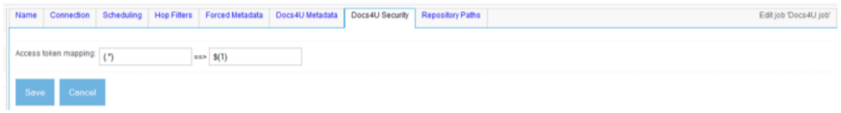
Docs4U URL attribute: url

Mapping:	Source metadata name	Docs4U metadata name
Delete	something	author
Delete	something else	title
Add		--Select target attribute name --

Save Cancel

Figure 9.6 Docs4U Metadata tab with URL attribute selected and two mappings added.

This looks good! Let's now click on the `Docs4U Security` tab. You should see something like figure 9.7.

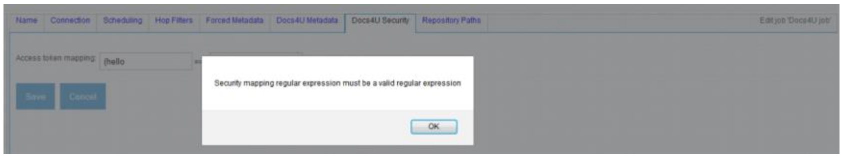

Name | Connection | Scheduling | Hop Filters | Forced Metadata | Docs4U Metadata | Docs4U Security | Repository Paths — Edit job 'Docs4U job'

Access token mapping: (.*) ==> $(1)

Save Cancel

Figure 9.7 The Docs4U Security tab. This displays only a single rules-based mapping for access tokens to user/group name.

Let's make sure that the UI enforces that what we enter as a regular expression actually must be one. Enter `(hello` in the left-hand field, and click the `Save` button. You should see an alert popup as depicted in figure 9.8.

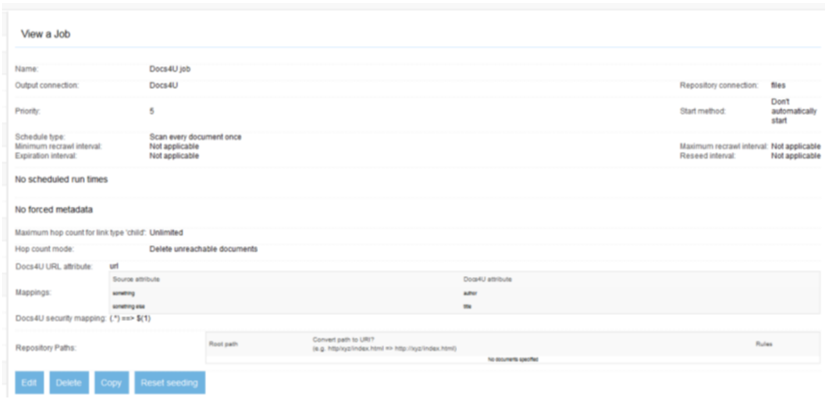

Access token mapping: (hello

Security mapping regular expression must be a valid regular expression

OK

Figure 9.8 The Javascript alert that results when an illegal regular expression is entered.

Now we'll fix the problem, and save the job. Replace the bad value with the original `(.*)`, and click `Save` once more. See figure 9.9.


View a Job

Name:	Docs4U job		
Output connection:	Docs4U	Repository connection:	files
Priority:	5	Start method:	Don't automatically start
Schedule type:	Scan every document once		
Minimum recrawl interval:	Not applicable	Maximum recrawl interval:	Not applicable
Expiration interval:	Not applicable	Reseed interval:	Not applicable

No scheduled run times

No forced metadata

Maximum hop count for link type 'child': Unlimited

Hop count mode: Delete unreachable documents

Docs4U URL attribute: url

Mappings:

Source attribute	Docs4U attribute
something	author
something else	title

Docs4U security mapping: (.*) ==> $(1)

Repository Paths:

Root path	Convert path to URI? (e.g. http/xyz/index.html => http://xyz/index.html)	Rules
	No documents specified	

Figure 9.9 View of a job that uses a Docs4U output connection, and a file system repository connection.

Congratulations! It looks like our UI is working well!

Now that we've made it through the Crawler UI parts, we can finally implement the methods that actually do the work. That's next.

9.3.3 Implementing the IOutputConnector output methods

Bert's been implementing away on his output connector, and has noticed something. "Gee," he says, "when I write a connector, I spend an awful lot of time writing UI pieces and much less time coding the functional guts of the connector."

You congratulate Bert on his astuteness, and assure him that his experience is not unique. It is true that your connectors will typically dedicate at least 50% of their code towards Crawler UI support.

Is this a bad thing? Remember that your connector is only as useful as it is usable, and pretty much every feature you add will require some UI to support it. In the case of the Docs4U output connector, the mappings that are required all unquestionably need UI support. You should not be afraid to invest time and effort to make the UI work better.

But now we begin implementing the core `IOutputConnector` methods – the ones that make the connector actually function. This consists of the methods `getOutputDescription()`, `addOrReplaceDocument()`, `removeDocument()`, and to some extent `noteJobComplete()`.

The purpose of the `getOutputDescription()` method is to create a string that, in a manner akin to the document version string you might recall from the repository connector, helps ManifoldCF keep track of whether a document needs to be reindexed or not. The `IOutputConnector` interface enforces a strict relationship between this version string and the actual output that takes place by restricting the availability of the `OutputSpecification` object to only the `getOutputDescription()` object. Thus, the connector code must convert the specification information into string form, and unpack it later, in order to function.

The purpose of the `noteJobComplete()` method is primarily to perform any kind of "commit"-like operation that is necessary to bring the newly indexed content into searchable condition. Docs4U does not have a transaction-based model, so the functionality is unneeded in this case.

The Docs4U output connector implementation of these methods, and their supporting methods, can be found in listing 9.8.

Listing 9.8 Docs4U output connector functional methods

```
public String getOutputDescription(OutputSpecification spec)
  throws ManifoldCFException, ServiceInterruption
{
  String urlMetadataName = "";
  String securityMap = "";
  List<String> metadataMappings = new ArrayList<String>();

  int i = 0;
  while (i < spec.getChildCount())
  {
    SpecificationNode sn = spec.getChild(i++);
    if (sn.getType().equals(NODE_URL_METADATA_NAME))                #1
      urlMetadataName = sn.getAttributeValue(ATTRIBUTE_VALUE);      #1
    else if (sn.getType().equals(NODE_SECURITY_MAP))                #2
      securityMap = sn.getAttributeValue(ATTRIBUTE_VALUE);          #2
    else if (sn.getType().equals(NODE_METADATA_MAP))                #3
    {                                                               #3
      String recordSource = sn.getAttributeValue(ATTRIBUTE_SOURCE); #3
      String recordTarget = sn.getAttributeValue(ATTRIBUTE_TARGET); #3
      String[] fixedList = new String[]{recordSource,recordTarget}; #3
      StringBuilder packBuffer = new StringBuilder();               #3
      packFixedList(packBuffer,fixedList,':');                      #3
      metadataMappings.add(packBuffer.toString());                  #3
    }                                                               #3
  }

  // Now, form the final string.
  StringBuilder sb = new StringBuilder();

  pack(sb,urlMetadataName,'+');                                     #4
  pack(sb,securityMap,'+');                                         #5
  packList(sb,metadataMappings,',');                                #6

  return sb.toString();
  }

  public int addOrReplaceDocument(String documentURI,
    String outputDescription,RepositoryDocument document,
    String authorityNameString, IOutputAddActivity activities)
    throws ManifoldCFException, ServiceInterruption
  {
    int index = 0;
    StringBuilder urlMetadataNameBuffer = new StringBuilder();           #7
    StringBuilder securityMapBuffer = new StringBuilder();               #7
    List<String> metadataMappings = new ArrayList<String>();             #7
    index = unpack(urlMetadataNameBuffer,outputDescription,index,'+');   #7
    index = unpack(securityMapBuffer,outputDescription,index,'+');       #7
    index = unpackList(metadataMappings,outputDescription,index,',');    #7

    String urlMetadataName = urlMetadataNameBuffer.toString();           #8
    Map<String,String> fieldMap = new HashMap<String,String>();          #8
    int j = 0;                                                           #8
    while (j < metadataMappings.size())                                  #8
    {                                                                    #8
      String metadataMapping = metadataMappings.get(j++);                #8
      String[] mappingData = new String[2];                              #8
      unpackFixedList(mappingData,metadataMapping,0,':');                #8
      fieldMap.put(mappingData[0],mappingData[1]);                       #8
    }                                                                    #8

    MatchMap securityMap = new MatchMap(securityMapBuffer.toString());   #9

    long startTime = System.currentTimeMillis();                        #10
    String resultCode = "OK";                                           #10
    String resultReason = null;                                         #10
    long byteCount = 0L;                                                #10

    userGroupLookupManager.cleanupExpiredRecords(startTime);            #11

    try
    {
      Docs4UAPI session = getSession();                                 #12
      try
      {
        D4UDocInfo docObject = D4UFactory.makeDocInfo();                #13
        try
        {
          if (document.countDirectoryACLs() > 0)                        #14
          {                                                             #14
            resultCode = "REJECTED";                                    #14
            resultReason = "Directory ACLs present";                    #14
            return DOCUMENTSTATUS_REJECTED;                             #14
          }                                                             #14

          String[] shareAcl = document.getShareACL();                   #15
          String[] shareDenyAcl = document.getShareDenyACL();           #15
          if ((shareAcl != null && shareAcl.length > 0) ||              #15
            (shareDenyAcl != null && shareDenyAcl.length > 0))          #15
          {                                                             #15
    resultCode = "REJECTED";                                    #15
    resultReason = "Share ACLs present";                        #15
    return DOCUMENTSTATUS_REJECTED;                             #15
  }                                                             #15

  String[] acl = performUserGroupMapping(document.getACL(),     #16
    securityMap,startTime);                                     #16
  String[] denyAcl = performUserGroupMapping(                   #16
    document.getDenyACL(),securityMap,startTime);               #16

  if (acl == null || denyAcl == null)                           #17
  {                                                             #17
    resultCode = "REJECTED";                                    #17
    resultReason = "Access tokens did not map";                 #17
    return DOCUMENTSTATUS_REJECTED;                             #17
  }                                                             #17

  docObject.setAllowed(acl);                                    #18
  docObject.setDisallowed(denyAcl);                             #18

  docObject.setMetadata(urlMetadataName,                        #19
    new String[]{documentURI});                                 #19

  Iterator<String> fields = document.getFields();               #20
  while (fields.hasNext())                                      #20
  {                                                             #20
    String field = fields.next();                               #20
    String mappedField = fieldMap.get(field);                   #21
    if (mappedField != null)                                    #22
    {                                                           #22
      if (Logging.ingest.isDebugEnabled())                      #22
        Logging.ingest.debug("For document '"+documentURI+      #22
          "', field '"+field+                                   #22
        "' maps to target field '"+mappedField+"'");            #22
      String[] stringValues = document.getFieldAsStrings(field);#23
      docObject.setMetadata(mappedField,stringValues);          #24
    }
    else                                                        #25
    {                                                           #25
      if (Logging.ingest.isDebugEnabled())                      #25
        Logging.ingest.debug("For document '"+documentURI+      #25
          "', field '"+field+"' discarded");                    #25
    }                                                           #25
  }

  byteCount = document.getBinaryLength();                       #26

  docObject.setData(document.getBinaryStream());                #27

  Map<String,String> lookupMap = new HashMap<String,String>();  #28
  lookupMap.put(urlMetadataName,documentURI);                   #28
  D4UDocumentIterator iter = session.findDocuments(null,null,   #28
    lookupMap);                                                 #28
  String documentID;
  if (iter.hasNext())                                           #29
  {                                                             #29
```

```
          documentID = iter.getNext();                                 #29
          session.updateDocument(documentID,docObject);                #29
        }                                                              #29
        else                                                           #29
          documentID = session.createDocument(docObject);              #29

        return DOCUMENTSTATUS_ACCEPTED;                                #30
      }
      catch (IOException e)
      {
        throw new ManifoldCFException(e.getMessage(),e);
      }
      finally                                                          #31
      {                                                                #31
        docObject.close();                                             #31
      }                                                                #31
    }
    catch (InterruptedException e)
    {
      throw new ManifoldCFException(e.getMessage(),e,
        ManifoldCFException.INTERRUPTED);
    }
    catch (D4UException e)
    {
      Logging.ingest.warn("Docs4U: Error ingesting '"+documentURI+"': "+
        e.getMessage(),e);
      throw new ManifoldCFException("Error ingesting '"+documentURI+
        "': "+e.getMessage(),e);
    }
  }
  catch (ManifoldCFException e)
  {
    if (e.getErrorCode() == ManifoldCFException.INTERRUPTED)           #32
    {                                                                  #32
      resultCode = null;                                               #32
      throw e;                                                         #32
    }                                                                  #32
    resultCode = "ERROR";                                              #33
    resultReason = e.getMessage();                                     #33
    throw e;                                                           #33
  }
  catch (ServiceInterruption e)
  {
    resultCode = "ERROR";
    resultReason = e.getMessage();
    throw e;
  }
  finally
  {
    if (resultCode != null)                                            #34
    {                                                                  #34
      activities.recordActivity(new Long(startTime),ACTIVITY_SAVE,     #34
        new Long(byteCount),documentURI,resultCode,resultReason);      #34
    }                                                                  #34
  }
  }

  protected String[] performUserGroupMapping(String[] inputACL,
    MatchMap securityMap, long currentTime)
    throws ManifoldCFException, ServiceInterruption
  {
    if (inputACL == null)                                           #35
      return new String[0];                                         #35
    String[] rval = new String[inputACL.length];
    int i = 0;
    while (i < rval.length)
    {
      String inputToken = inputACL[i];
      String mappedUserGroup = securityMap.translate(inputToken);    #36
      String userGroupID = lookupUserGroup(mappedUserGroup,currentTime);#37
      if (userGroupID == null)                                      #38
        return null;                                                #38
      rval[i++] = userGroupID;
    }
    return rval;
  }

  protected String lookupUserGroup(String userGroupName, long currentTime)
    throws ManifoldCFException, ServiceInterruption
  {
    String lockName = makeLockName(userGroupName);                  #39
    lockManager.enterWriteLock(lockName);                           #39
    try
    {
      String userGroupID =                                          #40
        userGroupLookupManager.lookupUserGroup(repositoryRoot,      #40
        userGroupName);                                             #40
      if (userGroupID != null)                                      #40
        return userGroupID;                                         #40
      try
      {
        Docs4UAPI session = getSession();
        userGroupID = session.findUserOrGroup(userGroupName);       #41
        if (userGroupID == null)                                    #41
          return null;                                              #41
        userGroupLookupManager.addUserGroup(repositoryRoot,         #42
          userGroupName,userGroupID,currentTime + CACHE_LIFETIME);  #42
        return userGroupID;
      }
      catch (InterruptedException e)
      {
        throw new ManifoldCFException(e.getMessage(),e,
          ManifoldCFException.INTERRUPTED);
      }
      catch (D4UException e)
      {
        Logging.ingest.error("Error looking up user/group: "+
          e.getMessage(),e);
        throw new ManifoldCFException("Error looking up user/group: "+
          e.getMessage(),e);
      }
    }
    finally                                                               #43
    {                                                                     #43
      lockManager.leaveWriteLock(lockName);                               #43
    }                                                                     #43
  }

  protected static String makeLockName(String userGroupName)
  {
    return "docs4u-usergroup-"+userGroupName;
  }

  public void removeDocument(String documentURI,
    String outputDescription, IOutputRemoveActivity activities)
    throws ManifoldCFException, ServiceInterruption
  {
    StringBuilder urlMetadataNameBuffer = new StringBuilder();            #44
    unpack(urlMetadataNameBuffer,outputDescription,0,'+');                #44
    String urlMetadataName = urlMetadataNameBuffer.toString();            #44

    long startTime = System.currentTimeMillis();                          #45
    String resultCode = "OK";                                             #45
    String resultReason = null;                                           #45

    try
    {
      Docs4UAPI session = getSession();
      try
      {
        Map<String,String> lookupMap = new HashMap<String,String>();      #46
        lookupMap.put(urlMetadataName,documentURI);                       #46
        D4UDocumentIterator iter = session.findDocuments(null,null,       #46
          lookupMap);                                                     #46
        if (iter.hasNext())
        {
          String documentID = iter.getNext();
          session.deleteDocument(documentID);                             #47
        }
      }
      catch (InterruptedException e)
      {
        resultCode = null;
        throw new ManifoldCFException(e.getMessage(),e,
          ManifoldCFException.INTERRUPTED);
      }
      catch (D4UException e)
      {
        resultCode = "ERROR";
        resultReason = e.getMessage();
        Logging.ingest.warn("Docs4U: Error removing '"+documentURI+"': "+
          e.getMessage(),e);
        throw new ManifoldCFException("Error removing '"+documentURI+"': "+
          e.getMessage(),e);
      }
    }
    finally                                                             #48
    {                                                                   #48
      if (resultCode != null)                                           #48
      {                                                                 #48
        activities.recordActivity(new Long(startTime),ACTIVITY_DELETE,  #48
          null,documentURI,resultCode,resultReason);                    #48
      }                                                                 #48
    }                                                                   #48
  }

  public void noteJobComplete(IOutputNotifyActivity activities)
    throws ManifoldCFException, ServiceInterruption
  {
  }
```

#1 Find the URL metadata name in output specification
#2 Find the access token mapping
#3 Process the metadata mappings
#4 Pack the URL metadata name into the string
#5 Pack the access token mapping into the string
#6 Pack the metadata mappings into the string
#7 Unpack the output description string
#8 Unpack each metadata mapping
#9 Construct a MatchMap object from the unpacked string
#10 Initialize variables for activity logging
#11 Clean up any expired user/group mapping records
#12 Get a Docs4U session
#13 Create a Docs4U document information object
#14 Check for directory ACLs, reject if found
#15 Check for share ACLs, reject if found
#16 Map the document ACLs to user/group IDs
#17 If mapping failed, reject
#18Set Docs4U document security information
#19 Set Docs4U URL metadata for the document
#20 Iterate through incoming metadata fields
#21Find the corresponding Docs4U metadata field
#22 If Docs4U field was found, log it
#23 Get the contents of the field from the incoming document
#24 Copy the field to the Docs4U document information object
#25 If field didn't map, log that
#26 Set the activity logging bytes from the document bytes
#27 Copy the document contents
#28 Look for the document, using the Docs4U metadata attribute
#29 If found, update the document, otherwise create
#30 Signal that the document was accepted
#31 Close the Docs4U document info object, as per the contract
#32 If interrupted, cancel activity logging
#33 Otherwise, record the exception information
#34 Record the activity, if there's a result code
#35 If input ACL is null, empty Docs4U ACL
#36 Use the MatchMap object to translate access token to user/group name
#37 Look up the user/group name
#38 If lookup failed, signal that document should be rejected
#39 Enter exclusive lock for named user/group

#40 Lookup user/group in database, return if found
#41 Lookup user/group in Docs4U, return if not found
#42 Write user/group info into database
#43 Exit exclusive lock
#44 Unpack URL metadata name only
#45 Initialize history logging data
#46 Look up document by its URL metadata value
#47 If found, delete the document in Docs4U
#48 Write a history record

At #1, we begin the `getOutputDescription()` method by finding the URL metadata name from the `OutputSpecification` object. We do the same at #2 for the security mapping information, and at #3 for the metadata mapping information. Once all three of these are in an appropriate form, we pack the URL metadata name into the string at #4, the security mapping at #5, and the array of metadata mappings at #6.

Next, we code the `addOrReplaceDocument()` method. At #7, we unpack the output description string's major components, and further unpack each metadata mapping record at #8. From the unpacked data, we construct a `MatchMap` object at #9, intended for access token to user/group name mapping. We're now ready to begin the actual ingestion code, so at #10 we initialize a set of variables which will be used to record history information at the end. At #11, we tell the user group lookup table manager to remove all expired records. At #12, we obtain a Docs4U session object, and create a Docs4U document information object at #13. At #14 and #15, we look for any security arrangement that is inconsistent with this output connector, and reject the document if found. At #16, we attempt to map the access tokens to Docs4U user/group identifiers, and at #17 reject the document if that cannot be done. We save the mapped information into the document information object at #18. At #19, we begin setting the metadata, starting with the URL metadata attribute. At #20, we loop through the incoming metadata fields, mapping them at #21 to Docs4U metadata fields. If mapping succeeded, we log this at #22, and copy the metadata value at #23 and #24. At #25, we also make sure to log when the mapping fails. At #26, we update the history variable that records the byte count from the incoming document's byte size, and at #27 we copy the document's contents into the document information object. It is unnecessary to close the `RepositoryDocument` input stream, because the creator of the input stream will close it when done. We're now ready to write the document into the target repository, which starts at #28 with a lookup of the document by its URL, and either a create or update operation at #29. Successful acceptance of the document is signaled at #30.

On the way out, we close the Docs4U document information object at #31, in accordance with the Docs4U API contract, and we process exceptions next. If the thread was interrupted, we avoid recording a history record at #32, otherwise we update the history information from the exception at #33. At #34, we record a history record based on the variables we initialized earlier.

In a supporting method, at #35 we deal with the possibility that the ACL from the `RepositoryDocument` object will be null. Otherwise, at #36, we use the MatchMap object we created to map each access token to a Docs4U user/group name. We lookup up the corresponding ID at #37, and if that fails we return null, signaling that the document should be rejected at #38.

The actual user/group lookup first enters an exclusive lock at #39, then looks up the user/group from the database table at #40, return the ID if found. Otherwise, at #41, it looks up the user/group from Docs4U, returning null if the ID was not found. If successfully located, the ID is written to the database at #42. The exclusive lock is exited at #43.

The `removeDocument()` method is next. At #44, we unpack only the URL metadata name from the output description string. We initialize history recording variables at #45, and look up the document by its URL metadata at #46. If found, we delete the document at #47, and write the history record at #48 with the updated history recording variables.

The security mapping functions of the code in listing 9.8 make use of the database table manager class we began coding earlier in the chapter. For the database-cached user and group lookup functionality, have a look at listing 9.9.

Listing 9.9 UserGroupLookupManager functional methods

```
public String lookupUserGroup(String repositoryRoot,
  String userGroupName)
  throws ManifoldCFException
{
  List params = new ArrayList();                                     #A
  params.add(repositoryRoot);                                        #A
  params.add(userGroupName);                                         #A
  IResultSet results = performQuery("SELECT "+userGroupIDField+      #A
    " FROM "+getTableName()+                                         #A
    " WHERE "+repositoryRootField+"=? AND "+                         #A
    userGroupNameField+"=?",params,null,null);                       #A
  if (results.getRowCount() == 0)                                    #B
    return null;                                                     #B
  IResultRow row = results.getRow(0);                                #C
  return (String)row.getValue(userGroupIDField);                     #C
}

public void addUserGroup(String repositoryRoot, String userGroupName,
  String userGroupID, long expirationTime)
  throws ManifoldCFException
{
  Map<String,Object> map = new HashMap<String,Object>();             #D
  map.put(repositoryRootField,repositoryRoot);                       #D
  map.put(userGroupNameField,userGroupName);                         #D
  map.put(userGroupIDField,userGroupID);                             #D
  map.put(expirationTimeField,new Long(expirationTime));             #D
  performInsert(map,null);                                           #D
}

public void cleanupExpiredRecords(long currentTime)
  throws ManifoldCFException
  {
    List params = new ArrayList();                                   #E
    params.add(new Long(currentTime));                               #E
    performDelete("WHERE "+expirationTimeField+"<=?",params,null);   #E
  }
```

#A Search for the matching record
#B If not found, return null
#C Return the ID value if found
#D Insert a new record into the table
#E Remove all records which have expired

Simultaneous output to the same URL

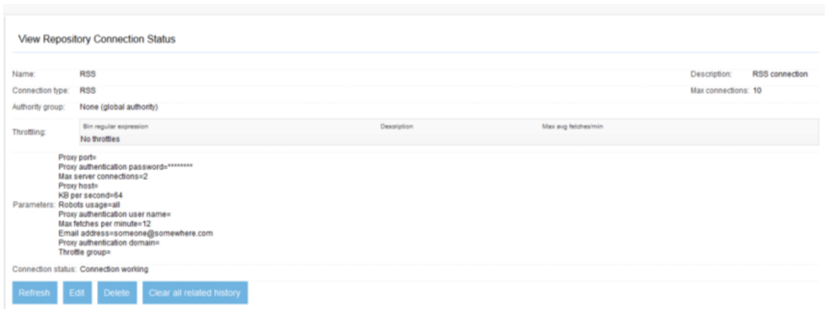
Astute readers may notice that there is a potential race condition in the way an output connector views the world. Since individual documents are uniquely described by their URLs, strange things could possibly happen if more than one ManifoldCF thread was trying to work with the same document at the same time. One thread could be adding the document, while another might be deleting it. Or, more strangely, two threads might try to "index" the document at about the same time!

But this is something that the ManifoldCF infrastructure explicitly prevents. Your output connector will never need to worry about dealing with such situations, because ManifoldCF makes sure that there is only one ongoing activity involving a single URL at the same time. Effectively, this means that ManifoldCF cannot trip itself up, although it is still possible for other clients of the same target index or repository to cause difficulties. But that situation will cause mayhem in any case, because ManifoldCF will no longer have control over the documents it is indexing – all of its attempts to manage incremental indexing will be undermined. This is not a recommended model.

RUNNING THE CONNECTOR CODE

We are finally ready to try a simple crawl using our new output connector! We're going to use our old friend the RSS connector for this purpose, since it has metadata in the form of `title` and `pubdate` fields, and since we can provide access tokens directly in this connector. We'll be targeting the `output_connector_example` Docs4U repository which has two user groups, `overlord` and `minion`, and has two users, `superman` and `clarkkent`. There is also a group called `DEAD_AUTHORITY`, which we'll need to have in order for there to be something for the negative assertion token to map to. We've already created a Docs4U output connection that points to this repository as well, earlier in the chapter.

We'll also need an RSS connection. All the default settings ought to do fine in this case, except the email, which you will need to provide. See figure 9.10.



View Repository Connection Status

Name:	RSS	Description:	RSS connection
Connection type:	RSS	Max connections:	10
Authority group:	None (global authority)		

Throttling:

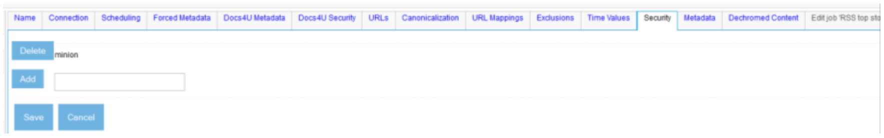
Bin regular expression	Description	Max avg fetches/min
No throttles		

Parameters:
Proxy port=
Proxy authentication password=********
Max server connections=2
Proxy host=
KB per second=64
Robots usage=all
Proxy authentication user name=
Max fetches per minute=12
Email address=someone@somewhere.com
Proxy authentication domain=
Throttle group=

Connection status: Connection working

Figure 9.10 The RSS connection we'll use to test our Docs4U output connector.

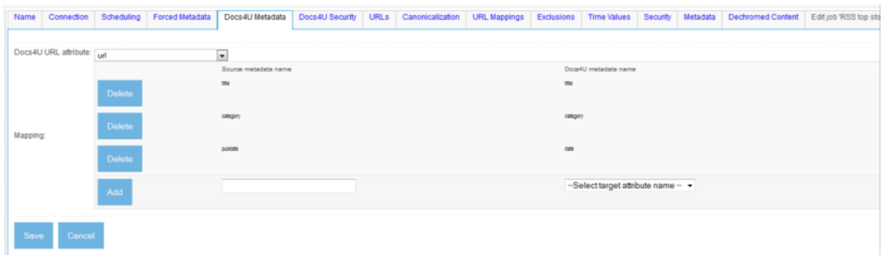
Now it's time to create the job. We'll need to include an RSS feed, of course, selected on the `URLs` tab. Pick whatever feed you like. We're going to want to set an access token for the job on the `Security` tab; let's make this visible to all minions, so enter minion in the text box and click the `Add` button. See figure 9.11.



minion

Figure 9.11 Setting up RSS access tokens for our test.

Next, we'll want to set up Docs4U metadata mapping. Click the `Docs4U Metadata` tab, and select the mappings and selections as seen in figure 9.12.



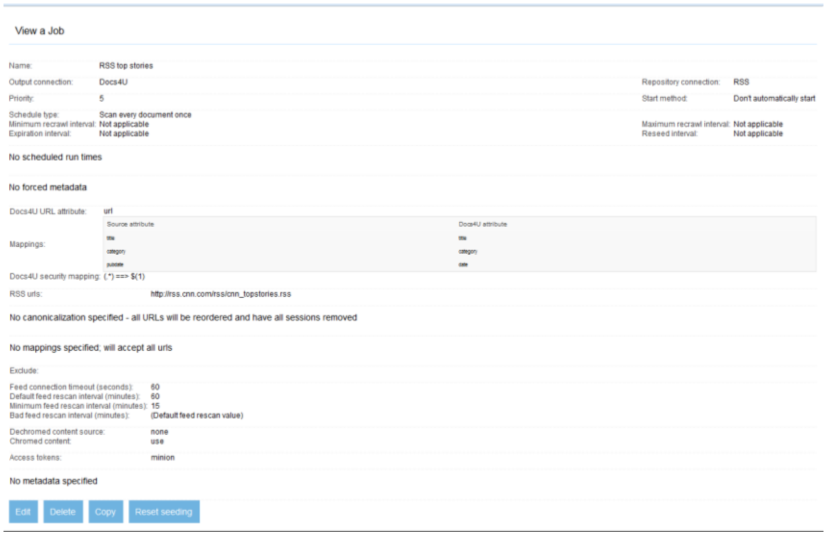
Docs4U URL attribute: url

Mapping:

Source metadata name	Docs4U metadata name
title	title
category	category
pubdate	date

Figure 9.12 Proper setup for the Docs4U Metadata tab for our test.

We won't need to modify the default Docs4U security mapping, since we chose an access token that is in fact a Docs4U group name. So we're done; go ahead and click the `Save` button. See figure 9.13.



View a Job

Name:	RSS top stories		
Output connection:	Docs4U	Repository connection:	RSS
Priority:	5	Start method:	Don't automatically start
Schedule type:	Scan every document once		
Minimum recrawl interval:	Not applicable	Maximum recrawl interval:	Not applicable
Expiration interval:	Not applicable	Reseed interval:	Not applicable

No scheduled run times

No forced metadata

Docs4U URL attribute: url

Mappings:

Source attribute	Docs4U attribute
title	title
category	category
pubdate	date

Docs4U security mapping: (.*) ==> $(1)

RSS urls: http://rss.cnn.com/rss/cnn_topstories.rss

No canonicalization specified - all URLs will be reordered and have all sessions removed

No mappings specified; will accept all urls

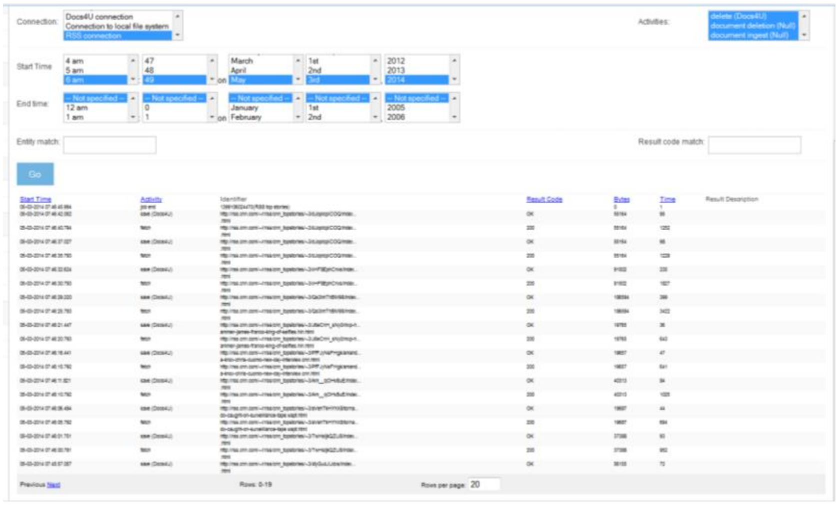
Exclude:

Feed connection timeout (seconds):	60
Default feed rescan interval (minutes):	60
Minimum feed rescan interval (minutes):	15
Bad feed rescan interval (minutes):	(Default feed rescan value)
Dechromed content source:	none
Chromed content:	use
Access tokens:	minion

No metadata specified

Figure 9.13 The completed Docs4U test job.

It's time for the moment of truth. Start the job, and let's see what happens. It should proceed until completed. When it is done, have a look at the Simple History report. See figure 9.14.



Connection: Docs4U connection / Connection to local file system / RSS connection (selected)

Activities: delete (Docs4U) / document deletion (Null) / document ingest (Null) (selected)

Start Time: 6 am : 49 on May 3rd, 2014

End time: -- Not specified -- : -- Not specified -- on -- Not specified -- -- Not specified --, -- Not specified --

Entity match:

Result code match:

Start Time	Activity	Identifier	Result Code	Bytes	Time	Result Description
[illegible]	job end	[illegible](RSS top stories)		0	1	
[illegible]	save (Docs4U)	[illegible]	OK	[illegible]	[illegible]	
[illegible]	fetch	[illegible]	200	[illegible]	1252	
[illegible]	save (Docs4U)	[illegible]	OK	[illegible]	[illegible]	
[illegible]	fetch	[illegible]	200	[illegible]	1228	
[illegible]	save (Docs4U)	[illegible]	OK	[illegible]	[illegible]	
[illegible]	fetch	[illegible]	200	[illegible]	[illegible]	
[illegible]	save (Docs4U)	[illegible]	OK	[illegible]	[illegible]	
[illegible]	fetch	[illegible]	200	[illegible]	3422	
[illegible]	save (Docs4U)	[illegible]	OK	[illegible]	36	
[illegible]	fetch	[illegible]	200	[illegible]	643	
[illegible]	save (Docs4U)	[illegible]	OK	[illegible]	47	
[illegible]	fetch	[illegible]	200	[illegible]	541	
[illegible]	save (Docs4U)	[illegible]	OK	[illegible]	94	
[illegible]	fetch	[illegible]	200	[illegible]	1025	
[illegible]	save (Docs4U)	[illegible]	OK	[illegible]	44	
[illegible]	fetch	[illegible]	200	[illegible]	694	
[illegible]	save (Docs4U)	[illegible]	OK	[illegible]	93	
[illegible]	fetch	[illegible]	200	[illegible]	952	
[illegible]	save (Docs4U)	[illegible]	OK	[illegible]	72	

Previous Next Rows: 0-19 Rows per page: 20

Figure 9.14 The Simple History report results of the test crawl. It looks like all the documents were properly saved!

Since that seemed to work, let's have a look at what the repository contains.

```
   cd output_repository_example
   ant list-repository
   list-repository:
        [java] 10   52043    (minion)               (DEAD_AUTHORITY)
[title=2         surviving         whales         released         off
Keys;url=http://rss.cnn.com/~r/rss/cnn_topstories/~3/Zicl0Y3RkUU/inde
x.html;date=1304810036000]
        [java] 11   70022    (minion)               (DEAD_AUTHORITY)
[title=Seized    videos    erase    some    of    the    bin    Laden
mystique;url=http://rss.cnn.com/~r/rss/cnn_topstories/~3/3wT0FkCOgEA/
index.html;date=1304806761000]
        [java] 12   52858    (minion)               (DEAD_AUTHORITY)
[title=Christians,            Muslims            clash            in
Cairo;url=http://rss.cnn.com/~r/rss/cnn_topstories/~3/0WkqdNqcLzk/ind
ex.html;date=1304824990000]
        [java] 5    50572    (minion)               (DEAD_AUTHORITY)
[title=Gadhafi's             forces             bomb             fuel
depots;url=http://rss.cnn.com/~r/rss/cnn_topstories/~3/thUNae_Hcjw/in
dex.html;date=1304781426000]
        [java] 6    43156    (minion)               (DEAD_AUTHORITY)
[title=Manny             Pacquiao             defeats             Shane
Mosley;url=http://rss.cnn.com/~r/rss/cnn_topstories/~3/lF5V81U3PtM/in
dex.html;date=1304820480000]
        [java] 7    52180    (minion)               (DEAD_AUTHORITY)
[title=Destroyer             named             for             fallen
SEAL;url=http://rss.cnn.com/~r/rss/cnn_topstories/~3/0R_b47iGoWM/inde
x.html;date=1304812010000]
        [java] 8    53544    (minion)               (DEAD_AUTHORITY)
[title=20-1             longshot             wins             Kentucky
Derby;url=http://rss.cnn.com/~r/rss/cnn_topstories/~3/9l-
vCzdIFF4/index.html;date=1304797389000]
        [java] 9    41589    (minion)               (DEAD_AUTHORITY)
[title=CAIR:        Two        Muslim        men        kicked        off
plane;url=http://rss.cnn.com/~r/rss/cnn_topstories/~3/7-
f7HKuhZ4g/index.html;date=1304770926000]
```

This looks great! We picked up the `title` and the `date` from the RSS metadata. The `category` attribute appears to not have been set, but that is no surprise, since we didn't attempt to set it in our job. The security looks right too. Congratulations! It looks like our output connector is working!

9.3.4 The requestInfo() method

The `requestInfo()` method exists to allow users of the ManifoldCF API the ability to perform the kinds of queries against the target repository that would permit someone to build their own UI. It turns out that, in the case of Docs4U, the only query that might be useful for UI construction is to request the list of metadata names. If this sounds familiar, it is because the Docs4U repository connector required precisely the same functionality for its document specification UI, back in Chapter 7. We can therefore copy the `requestInfo()` method from the repository connector class, and plop it down in our output connector class, and it should work, provided that the supporting methods (`getMetadataNames()` being the principal one) exist in the same form in the output connector already.

In fact, the code is truly identical! You can refer to listing 7.11 if you want to see it, or you can look at the `Docs4UOutputConnector.java` source file if you prefer. The only thing that differs a bit is the URL you would use to access it via the API, e.g:

```
curl "http://localhost:8345/mcf-api-
service/json/info/outputconnections/Docs4U/metadata"
{"metadata":[{"name":"author"},{"name":"category"},{"name":"date"},{"name":
"title"},{"name":"url"}]}
```

That's it for implementing our output connector! We're done with the hard work; now it's time to ponder how a properly written output connector can increase the overall efficiency of the whole crawler. We'll talk about that next.

9.4 System efficiency

It's easy to get stuck in the details when thinking about a complex system, and to some extent lose track of the bigger picture. You might spend weeks working through esoteric connector implementation issues, for example, and simply forget about how the system will behave when confronted with a task involving millions of documents, until you find yourself with an unwelcome problem – the problem of needing higher overall system efficiency.

System efficiency for an incremental crawler can be defined in terms of the amount of resources needed to achieve a specific goal, e.g. keeping a search index up to date with a specific repository, and the time over which those resources are needed, as measured against a set goal. The resources can be CPU cycles, disk space, network bandwidth, or whatever you care to measure, but it's convenient for a crawling task to use documents as a measure of the resources consumed. A number for system efficiency equal to or higher than 100% means that the system is meeting the desired goal.

System efficiency is related to throughput, in that throughput is the raw measure of the number of documents crawled over a given time period. But throughput by itself is a very misleading measure, as we initially discussed back in Chapter 1. It is possible to achieve significant improvements in system efficiency without improving throughput at all. That's what we're going to look at in this section.

9.4.1 Techniques for increasing throughput

The first place any competent engineer looks to improve when things are running too slowly is how to improve overall throughput of the system. In a system that has one thread of execution, this would imply changing code to improve its performance. But in ManifoldCF, the infrastructure is highly parallel, by design, so the system is quite able to make use of more CPUs. Thus, throughput tuning readily becomes an exercise in identifying and removing bottlenecks.

IDENTIFYING BOTTLENECKS

There are a number of places that bottlenecks can occur in ManifoldCF. In general, a good way to figure out if your system is hitting a bottleneck is to get a thread dump of the Agents process while it is going full tilt. How you do that will depend on whether you running Windows or Linux; on Windows, type `CTRL-BREAK` in the window that the agents process is running in, while on Linux, you find the process id and type `kill -QUIT` *process_id*. In both cases, the thread dump is sent to the process's standard output.

From the dump, it's straightforward to determine what all of ManifoldCF's worker threads are doing. These are threads labeled in the dump with a string that includes "Worker thread". We'll get into what these really are in detail in Chapter 12. If most of them are waiting, and the stack traces for most of the waiting threads are the same, you've probably identified a bottleneck.

TYPICAL BOTTLENECKS

The most typical kind of bottleneck is the maximum number of a given repository connection or output connection. We discussed these parameters in Chapter 2. The default maximum number of connections may be set by editing the connection; the default value is always `10`, which may well be unsuitable for the crawl you are trying to do. This situation shows up in a thread dump when many of the worker threads are waiting to obtain a repository or output connection.

Another typical kind of bottleneck occurs because the database cannot keep up with the threads available. In this case, most of the worker threads will be waiting to receive work to do. (Be careful, because average fetch rate throttling may yield a very similar looking thread dump.) If you conclude that the database is the bottleneck, you should consider switching to PostgreSQL. If you are already running PostgreSQL as your ManifoldCF database, then consider how your PostgreSQL maintenance is being done. PostgreSQL requires periodic maintenance in order to keep running at optimum performance. ManifoldCF is able to perform some of the maintenance – notably, periodically recalculating statistics and rebuilding indexes. But, eventually, a PostgreSQL operation called a "vacuuming" must be done, which requires periodic external intervention. All of this is discussed online at http://manifoldcf.apache.org/release/trunk/en_US/how-to-build-and-deploy.html, and http://manifoldcf.apache.org/release/trunk/en_US/performance-tuning.html.

If all of your worker threads seem to be humming away, or they seem to be waiting inside of connector code, you may want to consider increasing the number of them. This can be controlled by a `properties.xml` parameter, `org.apache.manifoldcf.crawler.threads`. Increasing this parameter also requires increasing the number of available database handles, if you are using PostgreSQL. And, as we've discussed, the PostgreSQL database is pretty much essential if you want to have decent throughput in the first place, since one of the claims to fame for this database is its high degree of parallelism.

But what if, after all of this, your efficiency is still not adequate? We'll talk about that now.

9.4.2 Techniques for increasing efficiency

When you are presented with an efficiency problem, it is very tempting to put all of your efforts into improving raw throughput numbers. But that's only half of the equation. The other half is the number of documents that you need to process in order to meet the goal. Here we're going to find ways of reducing the number of documents that need to be processed.

EXCLUDING DOCUMENTS

An output connector has the ability to provide information to the rest of ManifoldCF about what documents are suitable for inclusion in the crawl. We've discussed how the `IOutputConnector` methods `checkMimeTypeIndexable()` and `checkDocumentIndexable()` can save work, but it is important to recognize that these methods can also improve system efficiency, by reducing the number of documents that need to be crawled.

In order for these methods to be effective, not only does your output connector need to implement them, but your repository connectors must make use of them. For a repository connector implementation, they are available through the `IProcessActivity` and `IVersionActivity` objects, which are passed to the `processDocuments()` and `getDocumentVersions()` methods. As of this writing, the current connector implementations and usages of these methods across the ManifoldCF suite of connectors are currently patchy – this remains an area that is under development.

PREVENTING WASTEFUL RE-FETCHES

Another way to improve efficiency involves cutting back on the number of times a document needs to be crawled. ManifoldCF's ability to handle errors by retrying later perversely leads to situations where certain kinds of errors can lead to a document being retried many times, wreaking havoc on efficiency. It is therefore essential for your repository and output connectors to properly identify what should be a `ServiceInterruption` exception, and what should be a permanent `ManifoldCFException`. It is also valuable to consider carefully the time interval between retries, and how long retries should be attempted for, which are both parameters of the `ServiceInterruption` exception.

USING AN APPROPRIATE CRAWLING MODEL

As we discussed back in Chapter 3, the adaptive crawling model has the downside that pretty soon all of the crawling effort goes into refetching already indexed content, and very little goes into fetching new content. Picking the wrong model can thus completely wreck a system's efficiency. Considerable attention must therefore be given to selecting the right model, with the right parameters. Unfortunately, this is often easier said than done; the results of a given crawl may require days (or longer!) to obtain, so tweaking the model may well be a drawn-out affair. It is thus preferable to plan up front what you think you can get away with, and try to select a model that does the minimum, on the thought that it is usually easier to experiment by adding work to the system than it is to go the other way.

9.5 Summary

In this chapter, we've created an output connector. Not only did we write one, we chose to build one that connects to a target repository, rather than a search engine. This allowed us to explore a different sort of use-case for ManifoldCF – that of synchronizing content between repositories. We then went on to explore overall crawling system efficiency, learning that a crucial factor in such efficiency is the proper identification of the documents that should be considered, and those that should be ignored.

In the next chapter, we'll begin looking at the architecture of ManifoldCF. We'll examine the processes and the components within each process, and describe what they do.