8 Designing and Writing Authority Connectors

This chapter covers

- Designing authority connectors
- Writing authority connectors
- Authority connector performance enhancing techniques

In this chapter, we will build upon the repository connector we implemented in the last chapter, and develop an example of a working authority connector that can enforce the security model of an example repository. When that is done, we'll learn how to improve the performance of our authority connector, and explore some real-world case studies of system performance.

8.1 Designing an authority connector

Bert has now implemented a repository connector that connects to arch-nemesis Frank's new system. He can crawl content from it and get it into his nice, shiny Apache Solr-based search engine just fine. But he can't yet roll this connector out because, of course, Frank made sure that his system's security is not standard in any way at all. Therefore, Bert can't use any of the existing ManifoldCF authority connectors to do the job. It really looks like he's going to have to write his own. But he's behind in his work, and this is just another complication that he really can't afford, especially if it turns out to be as detailed as task as writing a repository connector was.

Luckily, you tell him, designing an authority connector is a very straightforward activity. The decisions that need to be made will be limited only to what the authority's access tokens

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

should look like, and what the authority's configuration information should include. There is no equivalent to the repository connector's document specification information, because an authority connector does not need to consider documents at all, and does not present any tabs in a job's Crawler UI page.

The design subtleties for an authority connector all have to do with how error cases are handled. You must make sure that your connector returns the correct access tokens for every possible error condition. Furthermore, because an authority connector controls the security of the documents ManifoldCF indexes, you have an obligation to be cautious and conservative in your design.

We will again be using the Docs4U repository for our example. As a demonstration of the kind of issues that might arise during real authority connector development, this repository is not ideal, so it will be important to consider a broader picture rather than just the code presented in this chapter.

8.1.1 Deciding on what access tokens to return, and when

Presumably, you will have chosen the form of the access token at the time you wrote the corresponding repository connector. Now it is time to decide what access tokens to return in what circumstances. The simple case, where the user exists and the access tokens corresponding to that user just need to be looked up, is only one of many. Table 8.1 describes some common circumstances and how best to think about them.

Table 8.1 Common authorization situations, and typical responses

Situation	**Response**
The repository cannot be reached, or the connector does not have the proper credentials needed to log in	The user should typically see only content that anonymous users of the repository can see. If there is a more conservative possibility, e.g. for users that exist but are disabled, then you must choose the most conservative possibility for your repository.
The repository can be reached, but the user does not exist	The user should see content that anonymous users of the repository can see.
The user exists, but is disabled	The response depends on the repository. Typically the user is allowed to see only anonymous content, but in some cases they are allowed to see no content at all.
The repository can be contacted, and the user exists	The response should be appropriate for that user, as if they were logged into the repository.

It is often necessary to do some exploration, at this point, to really understand the repository's security model, because you will not be able to determine the right logic until you know all the details. Here are some questions you might consider asking.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

- Does the repository support a notion of an anonymous user?
- Is there a concept of a document owner?
- Can users be disabled, and how does a disabled user behave?
- Is there a notion of public content, which is accessible by logged-in users, but not by anonymous users?

Once you believe that you thoroughly understand the repository's security model, you can begin to plan how to use access tokens to enforce it. You will need to have a specific answer for all of the cases described in table 8.2. Let's discuss some of the techniques that you might use to arrive at a workable solution.

ASSESSING TOKEN COUNTS

The number of access tokens associated with a user will directly affect the performance of the search, because each access token will typically generate a clause in a query expression. It therefore makes sense to choose access tokens in such a way as to minimize the number needed for every user.

This will be easier for some repositories than others. In particular, repositories that have only additive security (that is, they have no concept of restriction, only augmentation) tend to require many more access tokens per user than repositories that natively have the ability to restrict access. Documentum is one such repository. The Documentum authority connector can return 10,000 access tokens, under some conditions, for a single user.

There is little that you, as an authority connector writer, can do to reduce access token counts when the underlying repository requires that number as part of its security model. This is something you will just need to accept. You may be tempted to try to reduce the number of access tokens by clever means, but there are often costs associated with cleverness of this kind.

For example, suppose a repository supports both users and user groups, as Docs4U does. You may think it would be a good idea to "flatten" the user groups into individual user tokens at indexing time, thus requiring the authority connector to only return a single user token for every user. But this would probably be a mistake. First, the number of tokens indexed with every document could become very, very large, which might well cause problems. But, more importantly, the association between users and groups would be baked into the index. You could not change this association unless you recrawled the repository. Changes to a user's association with groups might therefore not be reflected in the search results for an extended time.

Thus, the access token count per user is going to be what it must be to implement security properly. You will need to have faith that the systems involved will be able to handle whatever comes along with sufficient performance to be workable.

Chapter author name: Wright
©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

NEGATIVE ASSERTIONS

One very common way of conditionally disabling user content is through the use of a *negative assertion* access token. This is a special token which your authority returns when it needs to restrict user access to a class of documents. The repository connector must also cooperate in that it must always add the special access token to the deny ACL for all documents in the class.

The actual token can be anything, so long as it has no chance of colliding with any other access token from the same authority. Since each repository has very strict rules as to what one of its ACLs looks like, it is usually easy to fulfill this requirement. If appropriate for your case, ManifoldCF also defines a standard negative assertion token value that you can just use. It is defined in the interface that all authority connectors must implement, and is called `GLOBAL_DENY_TOKEN`.

In typical usage, a negative assertion access token would be added to all documents which were not accessible to an anonymous user. Then, if the authority connector cannot reach the repository, or the user has been disabled, the connector would return the special token, and all non-anonymous documents would not be presented to the user.

> **Note** It is always necessary to have a negative assertion token in the case of repositories that have the explicit ability to deny access to a document or set of documents. It may not be necessary to have such a token for repositories where security is purely additive, but it does not hurt either.

Many of the connectors bundled together with ManifoldCF use the negative assertion technique. The Active Directory Authority in fact returns the standard negative assertion token, whose value is `DEAD_AUTHORITY`, whenever the active directory domain controller can't be reached. It is part of the contract for connectors that rely on the Active Directory Authority to properly index documents so as to include the `DEAD_AUTHORITY` token appropriately.

Now that we think we've figured out how to handle error cases, we're ready to move on to figuring out how to map an incoming user name to the user name that the repository will recognize.

8.1.2 Mapping the incoming user name to the repository's user name

An authority connector receives, as input, a user name. Its job is to find access tokens for this user. But there is a question we'd better ask at this point: what form will the user name actually take?

The question is not as silly as it seems at first, because ManifoldCF does not actually perform *authentication*, which is the process of insuring that a user is indeed who they claim to be. We discussed this requirement at some length in Chapter 4, and decided there that ManifoldCF did not have much to contribute towards the process of signing on a user to

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

whatever final application was written to present security-protected results. That part will depend completely on the application, and on the enterprise environment.

It's very typical in large enterprises to use Microsoft's Active Directory to perform system-wide single sign-on. Active Directory is effectively an implementation of MIT's Kerberos protocol, which has become a de-facto world-wide standard for authenticating users. ManifoldCF thus takes special care to work well with it, as we will see.

Once the authentication process is completed, we have a user name to pass to the authority service. But the user name that will result is the output of that authentication process. For Active Directory authentication, the user name will be of the form `user@domain`, where the user and domain are the Active Directory user and Active Directory domain, respectively. It is this user name that your ManifoldCF authority will see.

There is no fundamental reason why your system could not authenticate against something other than Active Directory. But if you do, bear in mind that some of the existing authority connectors included with ManifoldCF assume that the user name the Authority Service receives is in the form of an Active Directory user. The details are listed in table 8.2.

Table 8.2 Authority connectors and their relationship with Active Directory user names

Authority connector	**Relationship**
Active Directory and SharePoint/ActiveDirectory	The Active Directory authority needs to talk to the Active Directory domain controller, so it presumes the incoming user name is an Active Directory user name.
Documentum	Documentum has a built-in mapping from Active Directory user to Documentum user, so the Documentum authority presumes that the incoming user name is either an Active Directory user name or a native Documentum user name.
LiveLink	Older versions of Livelink provided no built-in mapping ability between Active Directory users and Livelink users, so the Livelink connector gives you a deprecated way to define a mapping rule. The right way is to use the Regular Expression mapper.
Meridio	Meridio has its own user management but is tightly integrated with Active Directory, and is a .NET application, so it assumes the incoming user name is an Active Directory user name.
SharePoint/Native	SharePoint internally needs users of the form "domain\username", but the SharePoint native authority automatically maps "username@domain" to the required format.

Chapter author name: Wright
©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

As you can see, each authority connector makes a somewhat different decision as to how to handle this problem. For repositories that are tightly integrated with Active Directory, it's not unreasonable to expect an Active Directory user name, because such repositories imply that Active Directory is already involved at some level. For un-integrated repositories, the trick that LiveLink authority connector uses – specifying a mapping rule – is a pretty general solution. If you have set things up so as to authenticate against one system in order to log into another, you know that such a mapping must exist. All you need to do is allow the user to specify it.

Indeed, since ManifoldCF 1.3, user name mapping functionality has been elevated to connector-level entities of their own. It is now not only possible to use a mapping rule (or multiple mapping rules) with each authority connection, but it is also possible to create a *mapping connector* which communicates with a third-party system in order to perform the appropriate user mapping work. But it is beyond the scope of this chapter to describe how to construct such a connector. Instead, we'll plan to use the regular expression mapper that comes with ManifoldCF.

For the Docs4U repository, there is no Active Directory integration whatsoever. A prudent design decision is therefore to include mapping functionality, on the chance that authentication will be against Active Directory. We'll need to be sure that we define how it will work when we specify the connector's configuration information, next.

8.1.3 Specifying the connector's configuration information

As you might expect, the configuration information needed for an authority connector designed to connect to a certain class of repository is often very similar to the configuration information needed by the corresponding repository connector. There may be small differences, however. For instance, information needed to construct a document URL may be needed for a repository connector, but not for an authority connector. The converse is sometimes true as well; sometimes the repository's service that deals with users and groups is distinct from the ones that deal with documents.

It's almost always worthwhile to start with the repository connector's configuration and make the necessary changes, rather than start from scratch. In the case of the Docs4U repository, if you recall we needed only one configuration parameter, which we called `rootdirectory`, which contained the path to the base of the Docs4U repository. If you examine the Docs4U API interface, which if you recall can be found at http://manifoldcfinaction.googlecode.com/svn/examples/edition_2_revised/docs4u/src/org/apache/manifoldcf/examples/docs4u/Docs4UAPI.java, you will note that Docs4U's user and group API is accessed using the same interface as the Docs4U document access methods. Therefore, no additional information will be required for the Docs4U authority connector. That makes our life very easy, and when implementation time comes, if we play our cards right we should be able to copy much code from the repository connector.

We will *not* the ability to specify a mapping from an Active Directory user name to a Docs4U user name, as discussed previously in this chapter. Since the mapping will be a simple string manipulation, the existing ManifoldCF regular expression mapper can be used to do the job.

That's really all there is to designing an authority connector. We're ready now to start planning the implementation!

8.2 Preparing to implement an authority connector

As we did for the repository connector in Chapter 7, we will consider both externally-facing design decisions as well as internally facing ones as part of the overall design process. The internally-facing decisions flow from the details of what we will actually need to code. So let's start by having a look at the `IAuthorityConnector` interface, which is the interface we'll be implementing.

8.2.1 The IAuthorityConnector interface

The `IAuthorityConnector` interface, whose full class name is `org.apache.manifoldcf.authorities.interfaces.IAuthorityConnector`, is derived from `IConnector`, as you have no doubt come to expect. This means that authority connectors use the very same model for connection pooling and basic configuration as all other connectors, which we described in depth in Chapter 6. The methods that are unique to `IAuthorityConnector`, and their meanings, are summarized in table 8.3.

Table 8.3 IAuthorityConnector methods and their meanings

Method	**Meaning**
`getAuthorizationResponse()`	Gets user access tokens given a user name.
`getDefaultAuthorizationResponse()`	Gets user access tokens given an exception or other difficulty retrieving access tokens via `getAuthorizationResponse()`.

Your friend Bert thinks that even without any help he should have no trouble implementing a connector that has so few methods. His expressed thought is, "even Frank could do it." You tell Bert that he's probably correct; this is not terribly hard. But there are a couple of implementation considerations he'll want to think about nonetheless. We'll start by discussing the response values from these two methods, and what they mean.

8.2.2 AuthorizationResponse objects

Both `IAuthorityConnector` methods return `AuthorizationResponse` objects. An `AuthorizationResponse` object always contains a list of access tokens, but in addition it also contains a response status. See table 8.4 for a list of the response statuses and their meanings.

Table 8.4 AuthorizationResponse response status values and their meanings

Response status	**Meaning**
`RESPONSE_OK`	The authority was reachable, the user exists, and the user is authorized to access the system.
`RESPONSE_UNREACHABLE`	The authority was unreachable.
`RESPONSE_USERNOTFOUND`	The authority was reachable, but the user did not exist.
`RESPONSE_USERUNAUTHORIZED`	The authority was reachable, the user exists, but the user is not authorized to access the system.

All of these different response codes are merely informational, and should not affect the way security is processed in any way. It is the returned access tokens themselves that matter where access restriction is concerned. But these codes represent a way of informing the user of potential reasons why the documents she is looking for might not be showing up. The Authority Service, as we have seen in Chapter 4, amalgamates each response code from each authority, and returns them independently, so that it is possible for the application to communicate any issues to the application's users.

DOCS4U RESPONSE CODES

You would think that Docs4U has no chance of ever legitimately signaling a situation where the authority is unreachable. But in fact there **is** such a situation – when the Docs4U authority connector is misconfigured so that it is not actually pointing to a Docs4U repository. Our authority connector will therefore need to take that situation into account.

The one case Docs4U actually will never need to return is the unauthorized user case. This is because users in Docs4U cannot be declared unauthorized; users either exist with full privileges, or they don't. But it should not be hard to imagine how an authority connector would deal with a repository that had this ability.

8.2.3 Connector-specific database tables

As is the case with repository connectors, authority connectors may choose to install and use database tables of their own to store persistent data. The creation of any needed database tables is expected to occur in the `IConnector` method `install()`, and their teardown in the `deinstall()` method.

I cannot furnish a current example of when such tables would naturally come in handy, because to date no authority connector I am aware of has required this capability so far. The authority connector for Docs4U example repository certainly won't either. Nevertheless, it is conceivable that some authority may eventually require caching of user access tokens on a scale that would require a disk-based solution. In that case, the ability to use connector-specific database tables would certainly come in handy.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

With the consideration of whether we might need connector-specific database tables, we're done preparing for implementation. Now it is time to start coding.

8.3 Coding the connector

As we did in Chapter 7, we're going to leverage a base class to make our coding easier, and protect it from potential future changes to the IAuthorityConnector interface. The recommended base class in this case is `org.apache.manifoldcf.authorities.authorities.BaseAuthorityConnector`. This class provides a base implementation of all of the `IConnector` methods, which we will need to override only for changes that are specific to our authority connector. There is also code in this class which is designed to map methods that were used by older authority connectors on to the current authority connector methods. These you can ignore.

Once again, we'll start by writing the `IConnector` methods first, and then proceed to write the rest as a second stage.

8.3.1 Writing the IConnector methods

Refer to table 7.3 to review the `IConnector` methods provided by the base class. In a manner very similar to the repository connector we wrote in Chapter 7, we'll need to override some of the methods of the base class in order to perform the connector's session management and to provide for the Docs4U user name mapping parameter. The methods we'll need to extend for the Docs4U authority connector are `connect()`, `poll()`, `check()`, and `disconnect()`. We'll also need implementations for `outputConfigurationHeader()`, `outputConfigurationBody()`, `processConfigurationPost()`, and `viewConfiguration()`.

IMPLEMENTATION

We're now ready to present the code implementing the `IConnector` methods. As is now the standard, we will be using Velocity templates to render the UI components. We'll get to those in a moment, but you can see the full source at http://manifoldcfinaction.googlecode.com/svn/examples/edition_2_revised/authority_connector_example/src/org/apache/manifoldcf/authorities/authorities/Docs4UAuthorityConnector.java. See listing 8.1.

Listing 8.1 IConnector method implementations for Docs4U authority connector

```
protected final static String PARAMETER_REPOSITORY_ROOT =           #1
  "rootdirectory";                                                  #1

protected final static long SESSION_EXPIRATION_MILLISECONDS =       #2
  300000L;                                                          #2

protected String rootDirectory = null;                              #3

protected ICacheManager cacheManager = null;
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

```
protected Docs4UAPI session = null;                                   #4
protected long sessionExpiration = -1L;                               #4

public void setThreadContext(IThreadContext tc)                       #5
  throws ManifoldCFException                                          #5
{                                                                     #5
  super.setThreadContext(tc);                                         #5
  cacheManager = CacheManagerFactory.make(tc);                        #5
}                                                                     #5

public void clearThreadContext()                                      #6
{                                                                     #6
  super.clearThreadContext();                                         #6
  cacheManager = null;                                                #6
}                                                                     #6

public void outputConfigurationHeader(IThreadContext threadContext,
  IHTTPOutput out, Locale locale, ConfigParams parameters,
  List<String> tabsArray)
  throws ManifoldCFException, IOException
{
  tabsArray.add("Repository");                                        #7
  Messages.outputResourceWithVelocity(out,locale,                     #8
    "ConfigurationHeader.html",null);                                 #8
}

public void outputConfigurationBody(IThreadContext threadContext,
  IHTTPOutput out, Locale locale, ConfigParams parameters,
  String tabName)
  throws ManifoldCFException, IOException
{
  Map<String,Object> velocityContext = new HashMap<String,Object>();   #9
  velocityContext.put("TabName",tabName);                             #9
  fillInRepositoryTab(velocityContext,parameters);                    #9
  Messages.outputResourceWithVelocity(out,locale,                     #9
    "Configuration_Repository.html",velocityContext);                 #9
}

public String processConfigurationPost(IThreadContext threadContext,
  IPostParameters variableContext, ConfigParams parameters)
  throws ManifoldCFException
{
  String repositoryRoot =                                            #10
    variableContext.getParameter("repositoryroot");                  #10
  if (repositoryRoot != null)                                        #11
    parameters.setParameter(PARAMETER_REPOSITORY_ROOT,               #11
      repositoryRoot);                                               #11

  return null;
}

public void viewConfiguration(IThreadContext threadContext,
  IHTTPOutput out, Locale locale, ConfigParams parameters)
  throws ManifoldCFException, IOException
{
    Map<String,Object> velocityContext = new HashMap<String,Object>();  #12
    fillInRepositoryTab(velocityContext,parameters);                    #12
    Messages.outputResourceWithVelocity(out,locale,                     #12
      "ConfigurationView.html",velocityContext);                        #12
  }

  protected static void fillInRepositoryTab(
    Map<String,Object> velocityContext, ConfigParams parameters)
  {
    String repositoryRoot =                                             #13
      parameters.getParameter(PARAMETER_REPOSITORY_ROOT);               #13
    if (repositoryRoot == null)                                         #14
      repositoryRoot = "";                                              #14
    velocityContext.put("repositoryroot",repositoryRoot);               #15
  }

  protected Docs4UAPI getSession()
    throws ManifoldCFException
  {
    if (session == null)
    {
      try
      {
        session = D4UFactory.makeAPI(rootDirectory);
      }
      catch (D4UException e)
      {
        Logging.authorityConnectors.warn(                               #16
          "Docs4U: Session setup error: "+e.getMessage(),e);            #16
        throw new ManifoldCFException("Session setup error: "+          #16
          e.getMessage(),e);                                            #16
      }
    }
    sessionExpiration = System.currentTimeMillis() +
      SESSION_EXPIRATION_MILLISECONDS;
    return session;
  }

  protected void expireSession()
  {
    session = null;
    sessionExpiration = -1L;
  }

  public void connect(ConfigParams configParameters)
  {
    super.connect(configParameters);
    rootDirectory = configParameters.getParameter(                      #17
      PARAMETER_REPOSITORY_ROOT);                                       #17
  }

  public void disconnect()
    throws ManifoldCFException
  {
    expireSession();
```

Chapter author name: Wright
©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

```
    rootDirectory = null;
    super.disconnect();
  }

  public String check()
    throws ManifoldCFException
  {
    Docs4UAPI currentSession = getSession();
    try
    {
      currentSession.sanityCheck();
    }
    catch (D4UException e)
    {
      Logging.authorityConnectors.warn(
        "Docs4U: Error checking repository: "+e.getMessage(),e);
      return "Error: "+e.getMessage();
    }
    return super.check();
  }

  public void poll()
    throws ManifoldCFException
  {
    if (session != null)
    {
      if (System.currentTimeMillis() >= sessionExpiration)
        expireSession();
    }
  }
```

#1 Declare rootdirectory parameter string
#2 Declare session expiration interval in milliseconds
#3 Root directory name
#4 Session object and time of expiration
#5 Override setThreadContext to obtain cache manager handle
#6 Override clearThreadContext to clear cache manager handle
#7 Add both tabs to the tabs array
#8 Display javascript template using Velocity
#9 Output “Repository” tab using Velocity
#10 Get the repository root parameter
#11 If it exists, set the parameter accordingly
#12 Fill in context with all tabs and output using Velocity
#13 Get the repository root parameter
#14 If null, set a default value
#15 Put the value into the Velocity context
#16 If exception, log error and throw a ManifoldCFException
#17 To connect, set up repository root parameter

At #1, we declare the name of the `repositoryroot` parameter. The session expiration interval constant is declared at #2. The root directory path variable is declared at #3. The data is completed by the Docs4U session object and expiration time variables, at #4.

At #5, we override the `setThreadContext()` method to allow us to obtain a Cache Manager handle. We’ll be using this later to help us with caching of access tokens. We also override the `clearThreadContext()` method, at #6, to make sure we don't hold onto a thread context reference when the connector class instance is returned to the pool.

The UI methods are next. For the `outputConfigurationHeader()` method, we declare one tab, `Repository`, at #7. This renders a Velocity template called `ConfigurationHeader.html` using Velocity, at #8. The main output method, `outputConfigurationBody()`, also uses Velocity to render the `Repository` tab at #9, first filling in the Velocity context using a helper method. The corresponding `processConfiguration()` method looks in the posted data for the `repositoryroot` form value at #10, and if it exists, sets the repository root parameter (at #11). In the `viewConfiguration()` method, we fill in the Velocity context with information for the tab, and use Velocity to render the `ConfigurationView.html` template, at #12.

The helper methods for filling in Velocity contexts are next. There is only one: the method that handles the `Repository` tab's data. At #13, we fetch the repository root parameter. If it null, we set it to a default value (in this case the empty string) at #14. At #15 we add the value to the Velocity context.

The session management code is very analogous to the repository connector code in Chapter 7. One of the few differences can be seen at #16, where no effort needs to be made to distinguish between transient and permanent connection issues, since there is no `ServiceInterruption` exception usage by the `IAuthorityConnector` interface. The other difference is the `connect()` method, at #17, saves the repository root string.

Note The code for session management for an authority connector differs in one major respect from comparable code for a repository connector, specifically that the authority connector does not have any concept of `ServiceInterruption` exceptions. Transient failures for authorities are not treated any differently from permanent failures. This, if you compare this code to the comparable code we developed in Chapter 6, you will see that the `getSession()` method there may throw a `ServiceInterruption`, while the code in this chapter cannot. Otherwise, the code is identical.

The Velocity templates themselves are similar in some respects to the corresponding ones from Chapter 7, but there must be additional code to support the user name mapping facility. See listing 8.2 for the Javascript template (`ConfigurationHeader.html`).

Listing 8.2 The ConfigurationHeader.html Velocity template

```
<script type="text/javascript">
<!--
function checkConfig()
{
  return true;
}

function checkConfigForSave()
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

```
{
  if (editconnection.repositoryroot.value == "")                          #A
  {
    alert("Enter a repository root");
    SelectTab("Repository");
    editconnection.repositoryroot.focus();
    return false;
  }
  return true;
}
//-->
</script>
```

#A Make sure repository root is non-null

The template for the "Repository" tab is almost identical to the corresponding one presented in Chapter 7 (see listing 7.3), so I won't include it here. The template that is rendered to view the connection is also pretty straightforward. See listing 8.3.

Listing 8.3 The ConfigurationView.html Velocity template

```
<table class="displaytable">
  <tr>
    <td class="description"><nobr>Repository root:</nobr></td>           #A
    <td class="value">$Encoder.bodyEscape($repositoryroot)</td>          #A
  </tr>
</table>
```

#A Output the repository root

BUILDING AND RUNNING THE AUTHORITY CONNECTOR

Let's try out what we have so far. We'll need to build it first, so let's check it out somewhere and do that.

```
svn co
http://manifoldcfinaction.googlecode.com/svn/examples/edition_2_revised/aut
hority_connector_example
cd authority_connector_example
ant jar
```

Next, we'll need to copy the appropriate jars into our already-built ManifoldCF Quick Start example. Copy the `build/jar/d4u-authority-connector.jar` file and the `lib/docs4u-example.jar` files from the example directory to your ManifoldCF build's `dist/example/connector-lib` directory. Finally, we'll need to edit the Quick Start's `dist/example/connectors.xml` file to tell it to register the new authority connector. Add the following line at the appropriate place:

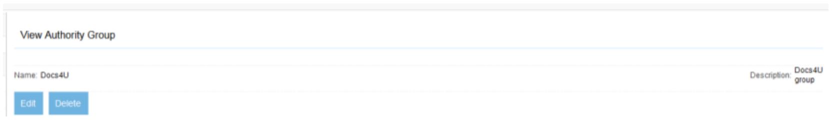
```
<authorityconnector name="Docs4U"

class="org.apache.manifoldcf.authorities.authorities.docs4u.Docs4UAuthority
Connector"/>
```

Now, if you start the Quick Start in the normal way, our new authority should be registered and be available for use. Once ManifoldCF is up and running, you should be able to open a

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

browser window and navigate to the Crawler UI. Start by clicking the `List Authority Groups` link from the navigation menu, and click the `Add a new authority group` link on the list page. Give your authority group a name, e.g. Docs4U, and save it. See figure 8.1.

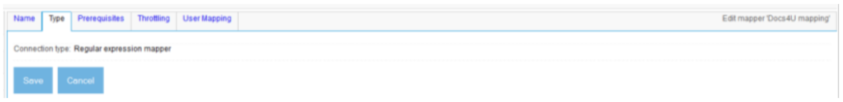


Name: Docs4U
Description: Docs4U group

Figure 8.1 The Docs4U authority group

Next, we will need to create a regular expression mapping connection, which can convert an Active Directory user into a Docs4U user. The regular expression mapper allows you to specify a regular expression, with parentheses identifying groups, and then select the groups for the output. You can output the groups in any order, using the syntax $(<group_number>) to identify the regular expression group, and add extra characters between them. You can even map group characters to lower or upper case, using a group number suffix of `l` or `u`.

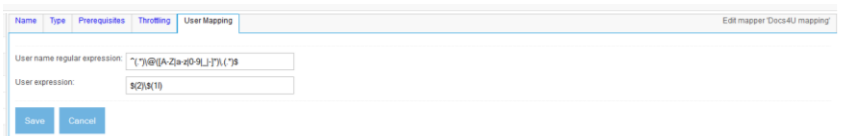
Click the `List Mapping Connections` link in the navigation menu, and click the `Add a new mapping connection` link on the list page. Give your mapping connection a name, and then click on the `Type` tab. On the `Type` tab, select `Regular expression mapper` from the pull-down, and click `Continue`. See figure 8.2.



Connection type: Regular expression mapper

Figure 8.2 Creating a regular expression user name mapper for Docs4U.

Now, click the `User Mapping` tab, so we can set the mapping we want. See figure 8.3.

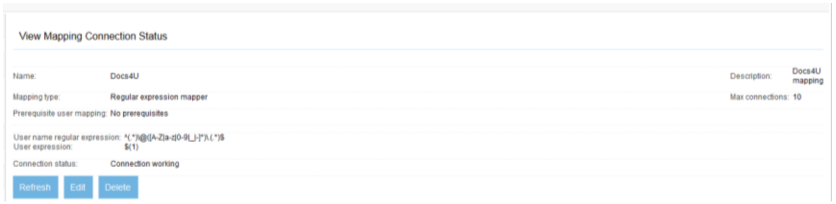


User name regular expression: ^(.*)\@([A-Z|a-z|0-9|_|-]*)\.(.*)$
User expression: $(2)\$(1l)

Figure 8.3 The User Mapping tab of the Docs4U user mapping connection.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

The default mapping for a regular expression mapping connection removes the domain from one end and adds it to the other. For our test, though, we don't want to do any of that; we just want a straight pass-through of the name, and we want to get rid of the domain entirely. We can get the mapper to do this by modifying the `User expression` field to get rid of the slash and `$(2)`. Also, since Docs4U is case sensitive as far as user names are concerned, let's change the `$(11)` to `$(1)`. Then, click the `Save` button. See Figure 8.4.

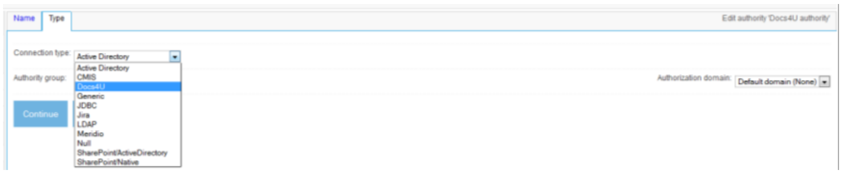


View Mapping Connection Status

Name:	Docs4U	Description:	Docs4U mapping
Mapping type:	Regular expression mapper	Max connections:	10
Prerequisite user mapping:	No prerequisites		
User name regular expression:	^(.*)\@([A-Z|a-z|0-9|_|-]*)\.(.*)$		
User expression:	$(1)		
Connection status:	Connection working		

Figure 8.4 Viewing the Docs4U user name mapper

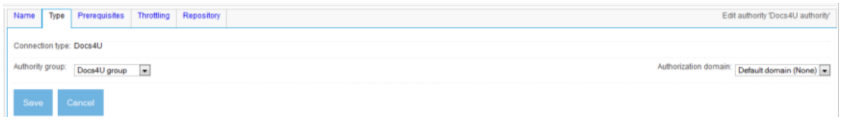
Now we are finally ready to create a Docs4U authority connection. Click the `List Authority Connections` link from the navigation menu, and then click the `Add a new connection` link on the list page. Provide an authority connection definition name, and then click the `Type` tab. You should see our Docs4U authority connector appear in the connector pulldown, as in figure 8.5.



Connection type: Active Directory
Authority group:
Authorization domain: Default domain (None)

Figure 8.5 Our Docs4U authority connector now appears in the connection type pulldown.

Select the Docs4U connection type, and the Docs4U authority group we created earlier, and click the Continue button. Our tab now appears. See figure 8.6.

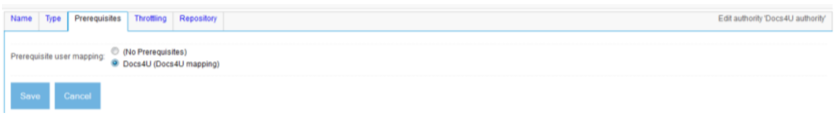


Connection type: Docs4U
Authority group: Docs4U group
Authorization domain: Default domain (None)

Figure 8.6 The Docs4U authority connector presents the Repository tab, as designed.

Next, we need to set up the regular expression mapper as a prerequisite of our authority connection, so that the user name mapping is performed in the manner desired. Click the

Prerequisites tab, and select the regular expression mapper we created earlier. See figure 8.7.

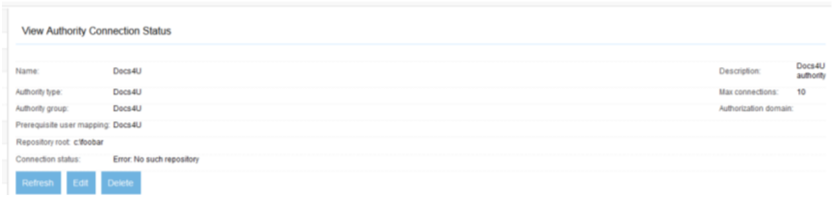

Name | Type | Prerequisites | Throttling | Repository — Edit authority 'Docs4U authority'

Prerequisite user mapping: ( ) (No Prerequisites) [✓ Docs4U (Docs4U mapping)]

Save Cancel

Figure 8.7 The Docs4U authority Prerequisites tab

Finally, click the `Repository` tab, and fill in some dummy repository path, since we haven't yet created a Docs4U repository we can connect to. Then, click `Save`. See figure 8.8.


View Authority Connection Status

Name:	Docs4U	Description:	Docs4U authority
Authority type:	Docs4U	Max connections:	10
Authority group:	Docs4U	Authorization domain:	
Prerequisite user mapping:	Docs4U		
Repository root:	c:\foobar		
Connection status:	Error: No such repository		

Refresh Edit Delete

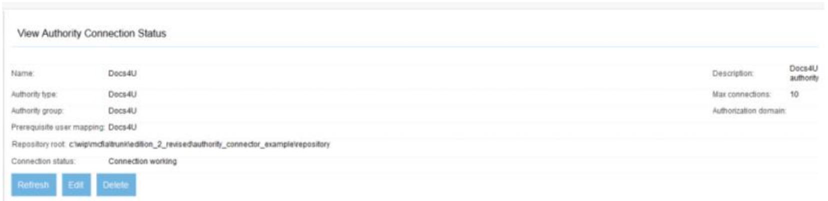
Figure 8.8 The Docs4U authority connector viewConfiguration() output, when the repository root is pointing to a non-existent path.

The next step is to actually create a Docs4U repository, so we can assure ourselves that the connection will not fail when given a proper document root. In the `authority_connector_example` directory, type the following:

```
ant sample-repository
```

This should create a repository with some documents, metadata, users, etc. The repository will be located in the `repository` subdirectory underneath `authority_connector_example`.

Now, in the Crawler-UI, click the `Edit` link for your authority connection definition. Click the `Repository` tab, and fill in the correct path to the Docs4U repository you just created. Then, click the `Save` button. See figure 8.9.


View Authority Connection Status

Name:	Docs4U	Description:	Docs4U authority
Authority type:	Docs4U	Max connections:	10
Authority group:	Docs4U	Authorization domain:	
Prerequisite user mapping:	Docs4U		
Repository root:	c:\wip\mcfia\trunk\edition_2_revised\authority_connector_example\repository		
Connection status:	Connection working		

Refresh Edit Delete

Figure 8.9 The output of viewConfiguration() when a proper repository directory has been entered as a

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

repository root.

Congratulations! It looks like everything works, so far! We're able to set up our authority connection definition, check that it is functional, and edit the user name mapping regular expression. Next, we'll write the code that actually does the work.

8.3.2 Writing the IAuthorityConnector authorization methods

The remaining methods of `IAuthorityConnector` are pretty straightforward to write. These consist only of `getAuthorizationResponse()` and `getDefaultAuthorizationResponse()`. The former is the workhorse method that does most of the work, while the latter is called only if the former throws an exception.

The only potential complication is whether or not you want to cache access tokens for a short period of time, as a way of improving performance and decreasing the load on the repository. In general, caching access tokens for a short time (on the order of a minute) is not long enough to create significant security risks, and it works wonders for those repositories that are not terribly fast in looking up access tokens.

The Docs4U authority connector uses caching in part to show how it's done. Listing 8.4 includes all the methods that are involved.

Listing 8.4 Docs4U authority connector IAuthorityConnector methods and their implementations

```
public static final String globalDenyToken = GLOBAL_DENY_TOKEN;        #1

private static final AuthorizationResponse unreachableResponse =       #2
  new AuthorizationResponse(new String[]{globalDenyToken},             #2
    AuthorizationResponse.RESPONSE_UNREACHABLE);                       #2

private static final AuthorizationResponse userNotFoundResponse =      #3
  new AuthorizationResponse(new String[]{globalDenyToken},             #3
    AuthorizationResponse.RESPONSE_USERNOTFOUND);                      #3

public AuthorizationResponse getAuthorizationResponse(String userName)
  throws ManifoldCFException
{
  if (Logging.authorityConnectors.isDebugEnabled())                    #4
    Logging.authorityConnectors.debug(                                 #4
      "Docs4U: Received request for user '"+userName+"'");             #4

  ICacheDescription objectDescription =                                #5
    new AuthorizationResponseDescription(userName,rootDirectory);      #5

  ICacheHandle ch = cacheManager.enterCache(                           #6
    new ICacheDescription[]{objectDescription},null,null);             #6
  try
  {
    ICacheCreateHandle createHandle =                                  #7
       cacheManager.enterCreateSection(ch);                            #7
    try
```

```
        {
          AuthorizationResponse response =                                 #8
            (AuthorizationResponse)cacheManager.lookupObject(              #8
              createHandle,objectDescription);                             #8

          if (response != null)                                            #9
            return response;                                               #9

          response = getAuthorizationResponseUncached(userName);          #10

          cacheManager.saveObject(createHandle,objectDescription,         #11
            response);                                                    #11
          return response;                                                #11
        }
        finally
        {
          cacheManager.leaveCreateSection(createHandle);
        }
      }
      finally
      {
        cacheManager.leaveCache(ch);
      }
    }

    protected AuthorizationResponse getAuthorizationResponseUncached(     #12
      String userName)                                                    #12
      throws ManifoldCFException                                          #12
    {
      if (Logging.authorityConnectors.isDebugEnabled())                   #13
        Logging.authorityConnectors.debug(                                #13
          "Docs4U: Calculating response access tokens for user '"+        #13
          userName+"'");                                                  #13

      String d4uUser = userName;                                          #14

      if (Logging.authorityConnectors.isDebugEnabled())                   #15
        Logging.authorityConnectors.debug("Docs4U: Mapped user name is '"+#15
          d4uUser+"'");                                                   #15

      Docs4UAPI currentSession = getSession();                            #16

      try
      {
        String userID = currentSession.findUser(d4uUser);                 #17

        if (userID == null)                                               #18
          return userNotFoundResponse;                                    #18

        String[] groupIDs = currentSession.getUserOrGroupGroups(userID);  #19

        if (groupIDs == null)                                             #20
          return userNotFoundResponse;                                    #20

        String[] tokens = new String[groupIDs.length+1];                  #21
```

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

```
      int i = 0;                                                        #21
      while (i < groupIDs.length)                                       #21
      {                                                                 #21
        tokens[i] = groupIDs[i];                                        #21
        i++;                                                            #21
      }                                                                 #21
      tokens[i] = userID;                                               #21
      return new AuthorizationResponse(tokens,                          #21
        AuthorizationResponse.RESPONSE_OK);                             #21
    }
    catch (InterruptedException e)
    {
      throw new ManifoldCFException(e.getMessage(),e,
        ManifoldCFException.INTERRUPTED);
    }
    catch (D4UException e)
    {
      Logging.authorityConnectors.error("Docs4U: Authority error: "+
        e.getMessage(),e);
      throw new ManifoldCFException("Docs4U: Authority error: "+
        e.getMessage(),e);
    }
  }

  public AuthorizationResponse getDefaultAuthorizationResponse(
    String userName)
  {
    return unreachableResponse;                                         #22
  }

  protected static long responseLifetime = 60000L;                      #23
  protected static int LRUsize = 1000;                                  #24

  protected static StringSet emptyStringSet = new StringSet();

  protected static class AuthorizationResponseDescription extends       #25
    org.apache.manifoldcf.core.cachemanager.BaseDescription             #25
  {
    protected String userName;                                          #26
    protected String repositoryRoot;                                    #26

    protected long expirationTime = -1;                                 #27

    /** Constructor. */
    public AuthorizationResponseDescription(String userName,
      String repositoryRoot)
    {
      super("Docs4UAuthority",LRUsize);                                 #28
      this.userName = userName;
      this.repositoryRoot = repositoryRoot;
    }

    public StringSet getObjectKeys()
    {
      return emptyStringSet;                                            #29
    }
    public String getCriticalSectionName()
    {
      return getClass().getName() + "-" + userName + "-" +              #30
        repositoryRoot;                                                 #30
    }

    public long getObjectExpirationTime(long currentTime)
    {
      if (expirationTime == -1)                                         #31
        expirationTime = currentTime + responseLifetime;                #31
      return expirationTime;                                            #31
    }

    public int hashCode()
    {
      return userName.hashCode() + repositoryRoot.hashCode();           #32
    }

    public boolean equals(Object o)
    {
      if (!(o instanceof AuthorizationResponseDescription))
        return false;
      AuthorizationResponseDescription ard =
        (AuthorizationResponseDescription)o;
      return ard.userName.equals(userName) &&                           #33
        ard.repositoryRoot.equals(repositoryRoot);                      #33
    }

  }
```

#1 Negative assertion token
#2 Precalculated ‘unreachable’ response
#3 Precalculated ‘user not found’ response
#4 Note the request to the log
#5 Construct a cache description object for the request
#6 Enter a caching sequence
#7 Enter a cache create section
#8 Look for a matching cached response
#9 If found, return it
#10 Build the response
#11 Save and return the response
#12 Uncached response calculation method
#13 Log the fact we got here
#14 No mapping required from user name to Docs4U name
#15 Log the Docs4U name
#16 Get a Docs4U session
#17 Look up the user
#18 If no user, return ‘user not found’ response
#19 Get the user’s groups
#20 If groups not found, ‘user not found’ response
#21 Return user + groups as access tokens
#22 Default response, in case of exception, is ‘unreachable’
#23 Response will be cached for 1 minute
#24 A maximum of 1000 Docs4U responses in the cache

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

#25 ICacheDescription class, extending BaseDescription
#26 The data used to identify the cached object
#27 The expiration time of the cached object
#28 Use LRU version of superclass constructor
#29 The cached object has no invalidation keys
#30 Build a critical section name from all data
#31 Expiration time is based on first call, which is creation
#32 Hash code is based on all data
#33 Equals code is based on all data

At #1, we declare the negative assertion token, which the repository connector and the authority connector should agree on. We pre-calculate 'unreachable authority' and 'user not found' responses involving that token at #2 and #3, respectively.

In the authorization response calculation method, we first log the fact we've seen the request, at #4. Then, we build a corresponding cache description object, at #5, and use it to enter the caching sequence at #6. We immediately enter a cache create section at #7, and once inside we look up the object at #8. If it was found, we return it at #9, otherwise we build the response at #10 (using the uncached version of the response calculation method), and save and return the response at #11.

The uncached response calculation method begins at #12. Once again, we log the fact we got here at all at #13. This will help us to distinguish between cached and non-cached requests. At #14, we simply copy the incoming user name to the Docs4U name, since as mentioned we intend to rely on a regular expression mapping connection to perform the necessary mapping from Active Directory user names to Docs4U user names. We log the user name at #15.

Next, we obtain a Docs4U session object at #16, and request the ID of the mapped user at #17. If the user did not exist, we return the 'user not found' response at #18. Otherwise, we obtain the user's groups at #19, also considering the user not to be found if that fails at #20. At #21, we fashion a response consisting of all the group IDs plus the user ID.

The default response method is called if the response calculation method returns an exception. The response, as discussed, is the 'unreachable' response, at #22.

Next, we describe the cache description object class. At #23 and #24, we provide constants describing the cached response lifetime and maximum LRU count. The class begins at #25, and extends `BaseDescription`, which insulates us from future changes to interfaces, and also provides standard support for LRU-based caches. At #26, we lay out the data that will uniquely identify the cached response. At #27, a separate variable is used to describe the expiration time of the object. When the class is constructed at #28, we make sure to use the superclass constructor that initializes LRU behavior, and provide an object class name as well as a maximum count. At #29, the object has no invalidation keys, because we have no way to explicitly invalidate it anyway. The critical section, at #30, is built from all the distinguishing data. At #31, we calculate the expiration time only once, on the first call to the `getObjectExpirationTime()` method, which corresponds to the

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

object's creation. The calculated expiration time is never updated and is returned on all subsequent calls. At #32 and #33, we implement the `hashCode()` and `equals()` methods making sure to consider all of the distinguishing data.

TRYING OUT THE CONNECTOR

To try out our finished connector, once again copy the jar to the ManifoldCF directory `dist/example/connectors-lib`, and restart ManifoldCF. We've already created an authority connection in the UI, so the next step is to see how it behaves. We'll use curl for that. Try the following command:

```
curl "http://localhost:8345/mcf-authority-
service/UserACLs?username=foo@bar"
```

The result should be:

```
USERNOTFOUND:Docs4U
TOKEN:Docs4U:DEAD_AUTHORITY
```

In this case, the only token returned is the global negative assertion token, as we desired. Next, let's try an operation that should work:

```
curl "http://localhost:8345/mcf-authority-
service/UserACLs?username=overlord@mydomain"
```

This time the answer is "authorized", with a reasonable value for the token returned.

```
AUTHORIZED:Docs4U
TOKEN:Docs4U:0
```

Congratulations! You've written your first authority connector! But it's left as an exercise for the student to try the case where the repository root points to nothing real, and confirm that caching of responses is working as expected. (Hint: turn on `authorityConnector` debug logging to see which responses are actually calculated, versus read from the cache.) As a further exercise, you may also want to try using the Docs4U connectors you have developed in conjunction with the Solr integration we described in Chapter 4.

8.4 Performance tuning authority connectors

Of all the components of ManifoldCF, the most sensitive and visible from a performance standpoint is the Authority Service. That's because this component is often involved on each request made by the end user. It therefore must be capable of meeting the demands of the aggregate end user community, in each installation.

In this section, we'll examine some case studies which demonstrate what an unforgiving task this can be. Then, we'll describe some coding solutions which may help your authority connector to perform up to specification.

8.4.1 How bad can it be?

Pretty near every large organization has more than one repository of content. Even just enumerating them all can often be an arduous task. On the other hand, when you build a search engine, often its purpose is to bridge all these disparate sources of content, and provide a central clearing-house of sorts, where content in any of the organization's repositories can be located with equal ease.

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

But this implicit consolidation creates a bottleneck, or choke-point, when security is involved. For each search, each repository must be consulted in order to insure that security is maintained. At best, this means that the entire search engine experience is limited by the slowest repository the organization possesses. This can lead to some very poor results, as we will see.

THE BRAZILIAN TANGO

One enterprise client, who was a Documentum shop, tried out our search product for almost a year in the lab before deciding to roll it out globally. But shortly after it "went live", I received a call. The whole thing, it seemed, was performing dreadfully slowly on customer searches.

After turning on debug logging for authorities, with timestamps, it became clear that one authority connection in particular was taking a very long time to respond. Indeed, it was the case that upon occasion it did not respond at all. I made inquiries into this connection, and discovered that the Documentum instance was somewhere in Brazil. The connection appeared to not only have a very high latency, but also a very low bandwidth. This meant that users with longer lists of access tokens were simply timing out while the tokens returned. Presumably, the sea turtles the tokens were riding upon were just too tired to go any faster.

The only solution we could think of was to simply remove that authority connection. This excluded all documents from that source from the search, which was unfortunate but could not be helped. ManifoldCF cannot fix basic infrastructure problems that are so drastic. One would hope that such major communication problems were rare in large organizations, but unfortunately that is far from the case.

THAT'S WAY TOO MANY TOKENS

Another case study that involves Documentum was even more interesting. The organization in question complained that everything worked pretty well unless certain people were doing the searches. Those particular people happened to be on the IT staff, so of course this was a high-priority problem.

Turning on authority connector debugging showed that the authority connections themselves were performing quite well. There was some small additional time taken to fetch the access tokens for the problem users, but it was not anywhere near enough to account for the delay the people actually saw. But when I enabled authority service debugging, the problem became obvious. This logger prints out all the access tokens for a user, and for the problem users I was not seeing just hundreds or even thousands of access tokens, but 25,000 or more of them. With numbers like that, it seemed likely that it was the search engine which was the bottleneck.

The underlying problem is that Documentum automatically provides a pre-generated ACL per folder, and the "access token" for Documentum is the ACL ID. The reason for all the tokens was the pre-configured ability of the IT personnel to see all the folders in the

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

repository. Standard users, of course, would not be affected, since they would see only small numbers of folders.

The good news was that, since Documentum's security is entirely additive, it was possible to consider just removing these access tokens from the authority connector's response. All that that could possibly change would be to eliminate the visibility of certain documents for certain users. Upon inquiry, it also transpired that this organization made absolutely no use of the automatically generated folder ACLs at all; all of these folder ACLs were a complete duplication of the standard user/group authorization structures they had built. So all I had to do was provide authority connector configuration that would allow the organization to filter out the unwanted tokens – and that cut the number of tokens down to a more manageable 2,000.

8.4.2 Performance debugging techniques

As we have seen in the case studies, by far the most valuable way of debugging performance problems with the authority service is by proper use of authority connector or authority service logging. Effective use of this technique, however, requires that time stamps be written to the log. We discussed this Chapter 7, but to refresh your memory, this is done by adding a line like this to your `logging.ini` file:

```
log4j.appender.MAIN.layout.conversionPattern = %d [%t] %-5p %c- %m%n
```

The two global loggers you can enable for the purposes of timing and displaying service information are `org.apache.manifoldcf.authorityconnectors` and `org.apache.manifoldcf.authorityservice`. As you might expect, you enable these by including the following lines in ManifoldCF's `properties.xml` file:

```
<property name="org.apache.manifoldcf.authorityconnectors" value="DEBUG"/>
<property name="org.apache.manifoldcf.authorityservice" value="DEBUG"/>
```

The `org.apache.manifoldcf.authorityconnectors` logger is the one used by all the individual authority connectors themselves. You have seen it in the example code presented in listing 8.1 and elsewhere. Use this if you are trying to figure out where all the time is going within a specific connector. The `org.apache.manifoldcf.authorityservice` logger, on the other hand, will dump all the requests to and responses from the Authority Service itself. It is therefore useful in figuring out if timing issues are the fault of the Authority Service as a whole, or something external, such as the search engine itself.

8.5 Summary

We've now worked through the planning and development of a simple authority connector, from the design stages to the implementation. We've learned how to deploy it, and how to run it. Finally, we explored some case studies and debugging techniques that I hope you will find illuminating when your authority connectors make it into the real world.

In the next chapter, we'll turn the problem around and discuss what it takes to develop an output connector in the ManifoldCF framework. We'll do this in the context of outputting content **to** a repository, instead of the other way around.

Chapter author name: Wright
©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727

©Manning Publications Co. Please post comments or corrections to the Author Online forum:
http://www.manning-sandbox.com/forum.jspa?forumID=727