SAS® Drug Development 3.4
Remote API
User's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2008. *SAS® Drug Development 3.4: Remote API User's Guide*. Cary, NC: SAS Institute Inc.

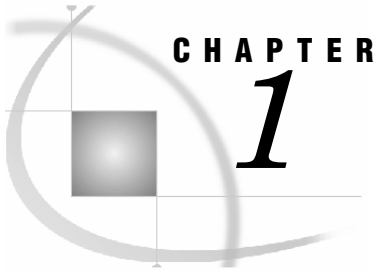
SAS® Drug Development 3.4: Remote API User's Guide

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, April 2008

SAS® Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at **support.sas.com/publishing** or call 1-800-727-3228.

Contents



CHAPTER 1

Introduction

Audience

This guide is for users who want to develop applications with the SAS Drug Development remote application programming interface (API). You must be familiar with the following:

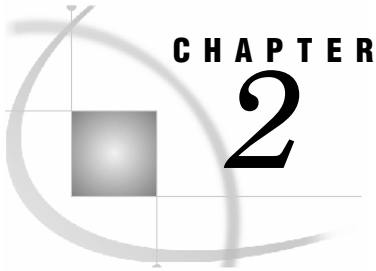
- Java programming
- SAS Drug Development functionality, such as type definitions, containers, files, and access permissions

  For reference information on SAS Drug Development functionality, see the SAS Drug Development online Help and user's guide.

Typographic Conventions Used in This Guide

Throughout this guide, you will see the following typographic conventions:

Convention	Description
`monospace` font	denotes code, such as a code example
`monospace bold` font	denotes text that you type, such as an object name
`monospace bold italics` font	denotes a value that you specify, such as your name



CHAPTER 2

Overview

Introduction

SAS Drug Development integrates with the Internet using certain technologies, such as workflow engines and encoding engines, typically used by the life sciences industry. Integration is provided by a component called a remote API.

The remote API enables developers and external applications to extend the functionality of SAS Drug Development by providing programmatic access to SAS Drug Development content and metadata. Programmatic access is implemented through a Java interface to SAS Drug Development. Java objects represent SAS Drug Development server metadata and repositories.

You can use the remote API to manage SAS Drug Development in a way that meets your company's business needs.

The Capabilities of the Remote API

The remote API consists of classes and interfaces that enable you to perform the following tasks:

- connect to the SAS Drug Development server
- read and write SAS Drug Development content and metadata
- read information about SAS Drug Development object types
- create, modify, and remove access permissions for SAS Drug Development objects
- read information about SAS Drug Development users and groups
- publish SAS code and parameter information to create a SAS Drug Development process

Benefits of the Remote API

The remote API provides an object-oriented, client-side view of the SAS Drug Development repository. The remote API facilitates development when providing extensions and integration solutions. Some of the benefits provided by the remote API are:

- The remote API is entirely remote to the installation of SAS Drug Development and requires access only to the SAS Drug Development server using the HTTPS protocol.
- The remote API uses transport protocol mechanisms that are completely abstracted and transparent. When you use the remote API, you do not need to be concerned with the low-level transport protocol mechanisms used.
- The remote API is completely validated, which means that it will continue to be the preferred method to build simple and robust integration solutions.

The Remote API Services

Overview

The remote API provides the following services:

- repository
- user
- group
- type
- sas

These services and their methods are described in this chapter. These services are the only objects in the remote API that make calls to the SAS Drug Development server.

Repository Service

Use the repository service (located in com.sas.drugdev.remote.repository) to work with nodes (containers and files) in the SAS Drug Development repository. The repository service is the service that you use most frequently when working with the remote API.

The repository service provides methods to perform the following general tasks:

- read the properties that are associated with a node
- set the values for the editable properties that are associated with a node
- enforce the validation rules as defined in the type definition for the node
- move, copy, and delete a node

In addition, the repository service provides methods to perform the following tasks with containers:

- create a container of any type while enforcing all containment rules

- move and copy a container while maintaining containment restrictions

Furthermore, the repository service provides methods to perform the following tasks with files:

- retrieve a file and its content from SAS Drug Development
- create and update a file's content
- enable file versioning
- view a file's version history
- check out and check in a file
- retrieve older versions of a file, including properties and content

Security Package

Use the security package within the repository service (located in com.sas.drugdev.remote.repository.security) to access and modify the access permissions on a SAS Drug Development node.

The security package provides methods to perform the following tasks:

- determine whether an access control entry has a specific access permission (such as Read access)
- change the access permissions of an access control entry
- determine whether an access control entry is a user or user group

User Service

Use the user service (located in com.sas.drugdev.remote.admin) to work with user accounts.

The user service provides methods to perform the following tasks:

- retrieve user account IDs from SAS Drug Development
- create and update a user account, including password, status, and properties
- add and remove system policies assigned to a user account

Group Service

Use the group service (located in com.sas.drugdev.remote.admin) to work with user groups.

The group service provides methods to perform the following tasks:

- retrieve a user group from SAS Drug Development
- create and update a user group
- add and remove members of a user group

Type Service

Use the type service (located in com.sas.drugdev.remote.type) to determine the type definition of nodes and to access the properties of nodes.

The type service provides methods to perform the following tasks:

- retrieve a list of type definitions
- retrieve the properties defined for a type, including whether each property is required, the default value for each property, and, if applicable, the list of possible values for each property

SAS Service

Use the SAS service (located in com.sas.drugdev.remote.sas) to publish and examine SAS Drug Development processes.

The SAS service provides methods to perform the following tasks:

- use process-ready SAS code and parameter information to publish a SAS Drug Development process
- examine a SAS Drug Development process by getting the parameters defined for that process

Accessing the Remote API Services

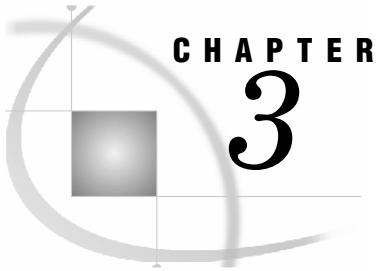
To access the remote API services, you must have a SAS Drug Development user ID and password. You must follow these general steps:

1. Connect to the SAS Drug Development server by creating a session.

   For sample code that illustrates how to connect to the SAS Drug Development server, see Chapter 4, “Remote API Reference.”

   Note: You are authenticated to the server for as long as the session is active. You have access to the remote API services throughout the session.

2. Use the service manager to look up and access the remote API services available in the session.



CHAPTER 3

Installing the Remote API

Requirements

Technical Requirements

The remote API is supported on Microsoft Windows and UNIX host environments. To install and use the remote API, your environment must include the following components:

Server:

- SAS Drug Development 3.2 or later
- The capability to deploy the remote API over the Web through the HTTPS protocol

Client:

- An operating system that runs Java 1.4.2_13 or later
- Java 2 Standard Edition Version 1.4 Software Development Kit (SDK) for the Java compiler

Required Documentation

Documentation that you need to install the remote API is included in the SASDrugDevRemoteAPI.zip file. Contact your internal SAS Drug Development project manager to get the location of this file.

As you learn about the remote API, the following additional resource might be helpful:

- the SAS Drug Development Remote API reference, which can be viewed with a Web browser
- example source files and sample programs

Before You Begin

Environment Hosted by SAS

If SAS hosts SAS Drug Development for your organization, confirm with your internal SAS Drug Development project manager that the server component of the remote API has been installed.

Environment Hosted by Customer

If your organization hosts SAS Drug Development, you must install the server component of the remote API before setting up the development environment. Perform these steps to install the server component:

1. Get the following information:
   - the location of the file sas-sdd-p21.ear
   - the URL to the WebLogic console application
   - the WebLogic system administrator user name and password
2. Create a backup copy of the file sas-sdd-p21.ear.

   Note: Do not skip this step.
3. Download the SASDrugDevRemoteApiServer.tar file and copy it to the temporary location where SAS Drug Development is deployed on the Web server.

   To get the location of this downloadable file, contact your internal SAS Drug Development project manager.
4. Extract the SASDrugDevRemoteApiServer.tar file to a folder named **`SASDrugDevRemoteApiServer`**.
5. Change folders to **`SASDrugDevRemoteApiServer`**.
6. Submit the following command:

   `sh ./install.sh` *`path_to_sas-sdd-p21`*`.ear`

   The file path must include the sas-sdd-p21.ear file.
7. Modify the access permissions of the sas-sdd-p21.ear file to match the access permissions of the user ID and group that own the domain and that run the process as follows:

   `rwx rwx --- (770)`
8. Log on to the WebLogic console application.
9. From the left pane, select **sdddomain ▶ Deployments ▶ Applications ▶ sas-sdd-p21**.
10. From the right pane, click the **Deploy** tab.
11. From the bottom of the right pane, click **Redeploy Application**.

A new Web application appears and is named **/sas-drugdev-remote.war**.

12 Select **sas-drugdev-remote.war** in the **Deployment Status for Web Application Modules** section.

Note: If you are reinstalling the remote API over an existing installation, you can skip this step.

13 Click the **Targets** tab, select the check box for the server on which the sas-sdd-p21.ear file is deployed, and then click **Apply**.

Note: If you are reinstalling the remote API over an existing installation, you can skip this step.

14 Click the **Deploy** tab, and then click **Deploy** next to **sas-drugdev-remote.war**.

Note: If you are reinstalling the remote API over an existing installation, you can skip this step.

15 Stop and restart the server.

Install the Remote API

1 Download and extract the SASDrugDevRemoteAPI.zip file.

To obtain the location of this file, contact your internal SAS Drug Development project manager.

Here is an illustration of the extracted files and folders:

2 At a command prompt, type the following command to verify that the Java compiler is in the executable path:

```
java –version
```

If the Java compiler is not in the executable path, submit one of the following commands:

Microsoft Windows:

```
set PATH=path-to-jdk\bin;%PATH%
```

UNIX:

```
PATH=path-to-jdk/bin:${PATH}
export PATH
```

3 Submit one of the following commands to verify that you have a connection to the SAS Drug Development server:

Microsoft Windows:

```
testConnection.bat
```

UNIX:

```
testConnection.sh
```

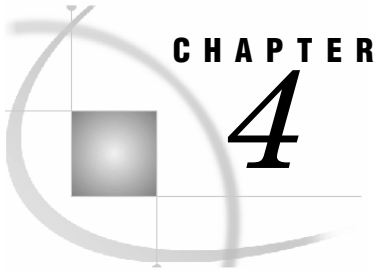
These test connection files are included in the files that you extracted from the SASDrugDevRemoteAPI.zip file.

4 When prompted for the SAS Drug Development server URL, your user name, and your password, use the URL https://*your_server*/sddremote, your SAS Drug Development user ID, and your password.

Note: If you are connecting to an instance of SAS Drug Development that is hosted by SAS, and you are connecting from within a firewall and behind proxy servers, you might need to address proxy requirements or restrictions. If HTTPS traffic is proxied, submit the following Java command to set the system properties that enable you to configure the Java run-time environment so that a specified proxy server and port is honored:

```
java -Dhttps.proxyHost=proxy_host_name
-Dhttps.proxyPort=proxy_port_number
```

If you receive a message stating that your connection was not successful, refer to Chapter 5, “Troubleshooting Connection Problems.”



CHAPTER 4

Remote API Reference

Overview

A remote API application that you create connects to the SAS Drug Development server. An application can use the services provided by the remote API to create, read, and write metadata.

After you have established a connection to the SAS Drug Development server, but before you begin to create your application, review the code in the **`Samples`** folder located in the SASDrugDevRemoteAPI.zip file. In the **`Samples`** folder, the following classes are presented:

- SASDrugDevRemoteAPISample.java enables you to test the connection to the SAS Drug Development server and to list directory contents for a specified path.
- SASDrugDevRemoteAPIRepositoryServiceSample.java provides sample methods to perform various repository actions.
- SASDrugDevRemoteAPIAdminServiceSample.java provides sample methods to perform administrative functions.

The files sample.bat (Microsoft Windows) and sample.sh (UNIX) are scripts that execute the sample code. These files contain information that helps you set the class path to execute your own code. The files show several ways to execute remote API commands, and show the results of each remote API command.

This chapter presents sample code that shows the most common tasks that you perform while developing a remote API application.

Writing and Compiling Code

Use the SAS Drug Development Remote API and this chapter to help you write your remote API application. The SAS Drug Development Remote API is located in the **doc** folder in the SASDrugDevRemoteAPI.zip file.

After you write your code, submit the following command to compile your code and to create the class file in the directory:

```
compile.bat java-source-file
```

The files compile.bat (Microsoft Windows) and compile.sh (UNIX) are scripts that specify the class path and compile the sample code.

As a final step, execute and test your code to ensure that you have set up everything correctly. Remember to test in a non-production environment if you do not want testing to affect the audit trail in your SAS Drug Development production system. If you do not have a test instance, identify and use a location in your production environment for testing that does not affect other users.

Connection to the SAS Drug Development Server

Open a Connection

The first step when developing a remote API application is to create a connection to the SAS Drug Development server. A session stores the state of the connection to the server and provides the access point to the service manager.

You create and initialize a session by calling **SessionFactory.newSession(***URL*, *credentials***)**. Then, use **isValid()** to determine whether the session is active.

Here is an example:

```
public class SASDrugDevRemoteAPIRepositoryServiceSample
{
```

```
   private Session session;
   private ServiceManager serviceMgr;
   private RepositoryService repositoryService;

   public SASDrugDevRemoteAPIRepositoryServiceSample(URL serverURL,
String userName, String password) {

      init(serverURL, userName, password);
   }

   // Initialize this SASDrugDevRemoteAPIRepositoryServiceSample

   instance private void init(URL serverURL, String userName, String
password) {

   //Create a new session

   session = createSASDrugDevSession(serverURL, userName, password);

   //Check for a valid session and initialize service manager

   if ( session.isValid()) {
      serviceMgr = session.getServiceManager();
      repositoryService = serviceMgr.getRepositoryService();
   }
}

//Create session/open connection to SDD server

public Session createSASDrugDevSession(URL serverURL, String userName,
String password) {

   // Create a session using the given user credentials
   Session session = null;
   try {
      Credentials credentials = new UsernamePasswordCredentials(userName,
password.toCharArray());
      session = SessionFactory.newSession(serverURL, credentials);

   } catch (AuthenticationException e) {
      // Handle exception
      e.printStackTrace();
   } catch (PasswordExpiredException e) {
      // Handle exception
      e.printStackTrace();
   } catch (UserInactiveException e) {
      // Handle exception
      e.printStackTrace();
   } catch (UserRetiredException e) {
      // Handle exception
      e.printStackTrace();
   } catch (RemoteException e) {
      // Handle exception
      e.printStackTrace();
   } catch (UnsupportedCredentialsException e) {
      // Handle exception
      e.printStackTrace();
   }
   return session;
   }

}
```

Close a Connection

When you are finished using the SAS Drug Development server, close the connection by first checking for a valid session and then using **`invalidate()`** to close the connection.

Here is an example:

```
// Check if session is valid before closing server session

if (session.isValid())
{
   // Invalidate session and free up server side resources held by this
session
   session.invalidate();
}
```

Access to the Remote API Services

To access the services in the remote API, retrieve the service manager from the session object. The remote API uses the service manager to access all of the services available in the session.

The following example shows how to connect to the services, use the service manager, and access the repository service:

```
// Check for a valid session and initialize service manager
if ( session.isValid()) {

   serviceMgr = session.getServiceManager();
   repositoryService = serviceMgr.getRepositoryService();
}
```

Folders and Files

Overview

The repository service provides methods to manipulate folders and files in the SAS Drug Development repository.

The type service provides methods for handling the types of objects in SAS Drug Development.

Create a File

To create a file in the SAS Drug Development repository, you must specify the path and the type of file. You can specify a predefined SAS Drug Development type or a custom type, or you can allow the type to be determined by the type of the uploaded file.

Here is an example:

```
// Create a new file of the given type in SAS Drug Development

String sddFilePath = "/SDD/Testing/foo.txt";
String typeName = "document";
String localFilePath = "c:\temp\foo.txt";
Properties properties = new Properties();
properties.put("description", "Created by sample code");

try {

   //Construct a FileBean with the SAS Drug Development path and
specified file type

   FileBean fileBean = new FileBean(pathinSDD, new TypeBean(typeName));

   //Set contents of the FileBean using the localFile to be pushed to SAS
Drug Development repository

   fileBean.setContents(new File(localFilePath));

   //Set properties if available, else will be set to default values
   fileBean.setProperties(properties);

   //Create this file on the server
   RemoteFile remoteFile = repositoryService.createFile(fileBean);

   //Create was successful, use remoteFile to perform other actions on
this file
   System.out.println("\n Created new file" + remoteFile.getPath() + "
with id = " + remoteFile.getId() + "\n");

} catch (Exception e) {

   // Handle exception

   e.printStackTrace();
}
```

Create a Folder

To create a folder in the SAS Drug Development repository, you must specify the path. You can specify the type of folder if you need to create a container other than a Folder object. You can specify a type provided by SAS Drug Development or a custom-defined type.

Here is an example:

```
// Create a new folder and protocol

String sddFolderPath = "/SDD/Testing/newFolder";
String sddProtocolPath = "/SDD/Testing/newProtocol";


try {

   //Construct a ContainerBean with the SAS Drug Development path and
specified file type

   ContainerBean folderBean = new ContainerBean(sddFolderPath);
    ContainerBean protocolBean = new ContainerBean(sddProtocolPath, new
TypeBean("protocol"));
```

```
   //Create this container on the server
   RemoteContainer remoteFolder =
repositoryService.createContainer(folderBean);
RemoteContainer remoteProtocol =
repositoryService.createContainer(protocolBean);

   //Create was successful, use remote container to perform other actions
on this file
   System.out.println("\n Created new folder" + remoteFolder.getPath() +
" with id = " + remoteFolder.getId() + "\n");
System.out.println("\n Created new protocol" + remoteProtocol.getPath() +
" with id = " + remoteProtocol.getId() + "\n");

} catch (Exception e) {

   // Handle exception

   e.printStackTrace();
}
```

Get a File

To get a file from the SAS Drug Development repository, you must specify the path to the file or the file ID, if it is available. You can also specify the version of the file that you want to get.

When you get a file, only the file's metadata (including properties, owner, and version information) is returned, not the contents of the file. To get the contents of a file, use **`getContents()`**.

Here is an example:

```
// Get a file of specified path from SAS Drug Development


String sddFilePath = "/SDD/Testing/foo.txt";
try {

   FileBean fileBean = new FileBean(sddFilePath);
   RemoteFile remoteFile = repositoryService.getFile(fileBean);
} catch (IllegalStateException e) {
      // Handle exception
      e.printStackTrace();
} catch (InvalidVersionException e) {
      // Handle exception
      e.printStackTrace();
} catch (InvalidTypeException e) {
      // Handle exception
      e.printStackTrace();
} catch (RemoteException e) {
      // Handle exception
      e.printStackTrace();
} catch (InvalidSessionException e) {
      // Handle exception
      e.printStackTrace();
}
```

Copy a Folder or File

To copy a folder or file from one location in the SAS Drug Development repository to another location in the repository, you must specify a source path and a destination path.

Here is an example:

```
// Copy a file from one location in SAS Drug Development to another

String srcPath = "/SDD/Testing/foo.txt";
String destPath = "/SDD/Testing/MyFolder/foo.txt";

// Construct FileBean for the source and destination paths

FileBean srcfileBean = new FileBean(srcPath);
FileBean destfileBean = new FileBean(destPath);

// Note: to copy container objects, you will need to use ContainerBean
instances

try {

   RemoteNode copiedNode = repositoryService.copyNode(srcFileBean,
destFileBean);
   String newPath = copiedNode.getPath();

   System.out.println("\n"+srcPath+" has been copied to "+newPath);

} catch (Exception e) {
      // Handle exception
      e.printStackTrace();
}
```

Move a Folder or File

To move a node (a folder or file) in the SAS Drug Development repository, you must specify a source node and a destination node.

If the destination node is an existing container, the source node is moved into the destination container, and the source node's name remains the same.

If the destination node does not exist, but its parent node does exist, the source node is moved into the parent node. The source node's name is changed to the value specified for the destination node because the destination is assumed to be a full path, including the node name.

Here is an example:

```
// Move a file or container from one location in SAS Drug Development to
another
String srcPath = "/SDD/Testing/TestFolder";
String destPath = "/SDD/Testing/MyFolder";
try {

   // Construct ContainerBean instances for the source and destination
paths in SDD
   ContainerBean srcContainerBean = new ContainerBean(srcPath);
   ContainerBean destContainerBean = new ContainerBean(destPath);
```

```
   // Note: to copy file objects, you will need to use FileBean instances

   RemoteNode moveNode = repositoryService.moveNode(new
NodeBean(srcPath), new NodeBean(destPath));
   String newPath = moveNode.getPath();

   System.out.println("\n" + srcPath + " has been moved to " + newPath);

} catch (Exception e) {
   // Handle exception
   e.printStackTrace();
}
```

Delete a Folder or File

To delete a folder or file in the SAS Drug Development repository, you must specify a node (a folder or file).

If the node is a folder, all of the node's children are also deleted.

If the user does not have access permission to delete the node below the specified folder, nothing is deleted and an exception is thrown.

Here is an example:

```
// Delete a file or container in SAS Drug Development
String filePath = "/SDD/Testing/foo.txt";
try {

   // Delete the given object in SAS Drug Development using the
repository service
   FileBean fileBean = new FileBean(filePath);
   repositoryService.deleteNode(fileBean);

   System.out.println("/nSuccessfully deleted "+ filePath);

} catch (IllegalStateException e) {
      // Handle exception
      e.printStackTrace();
} catch (InvalidNodeException e) {
      // Handle exception
      e.printStackTrace();
} catch (PermissionException e) {
      // Handle exception
      e.printStackTrace();
} catch (RemoteException e) {
      // Handle exception
      e.printStackTrace();
} catch (InvalidSessionException e) {
      // Handle exception
      e.printStackTrace();
}
```

List the Contents of a Folder

To list the contents of a folder, you must specify a full path (starting with `/SDD`) to the folder in the SAS Drug Development repository. The path must be to a folder.

A list of the children of the folder is returned. If the specified folder does not have any children, an empty list is returned.

Here is an example that shows how to print the names of the children of a folder:

```
// List contents of a directory

String sddContainerPath = "/SDD/Testing";

try {

   // Get children for the specified container using repository service

   List children = repositoryService.getChildren(new
ContainerBean(sddContainerPath));

   // Print the child node paths

   System.out.println("Children of "+ sddContainerPath + " are:");

   for (Iterator iter = children.iterator(); iter.hasNext();) {
      Node node = (Node) iter.next();
      System.out.println(node.getPath());
   }

} catch (IllegalStateException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidNodeException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidTypeException e) {
   // Handle exception
   e.printStackTrace();
} catch (RemoteException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // Handle exception
    e.printStackTrace();
}
```

Get the Properties of a Folder or File

To get the properties of a folder or file, you must create a file bean that represents the folder or file. Then, you must create a node that represents the bean. And then, you need to get the properties of the node.

Here is an example that shows the use of a bean and node to print the properties of a folder:

```
// Sample code to get properties
String sddPath = "/SDD/Testing";
try {
   FileBean fileBean = new FileBean(sddPath);
   RemoteNode node = repositoryService.getNode(fileBean);

   System.out.println("\nGetting properties for "+ path);
   Map properties = node.getProperties();

   // Look up properties by name

   // Iterate through the properties
   for (Iterator iter = properties.keySet().iterator(); iter.hasNext();)
   {
```

```
      String propertyName = (String) iter.next();
      String propertyValue = (String)properties.get(propertyName);

      // Do something with the property, for ex: print them out
      System.out.println(propertyName + " = " + propertyValue);
   }

} catch (RemoteException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // Handle exception
   e.printStackTrace();
}
```

Set the Properties of a Folder or File

Here is an example that shows how an individual file property can be set:

```
// Example to show how node properties can be modified individually

String filePath = "/SDD/Testing/foo.txt";
FileBean updateBean = new FileBean(filePath);
String propertyName = "description";
String propertyValue = "new description";

try {
   RemoteNode node = repositoryService.getNode(updateBean);

   // Checking write permission requires a call to the server
   if ( node.canWrite())
   {
   // Set a single property
   updateBean.setProperty(propertyName, propertyValue);
   node = repositoryService.updateNode(updateBean);
   System.out.println("Modified " + propertyName + " to " +
propertyValue.toString());
   }
   else
   {
   System.out.println("You do not have permissions to modify this node");
   }

} catch (RemoteException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidNodeException e) {
    // Handle exception
   e.printStackTrace();
} catch (InvalidNodeBeanException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // Handle exception
   e.printStackTrace();
} catch (NodeModifiedException e) {
   // Handle exception
   e.printStackTrace();
} catch (ValidationException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidTypeException e) {
   // Handle exception
   e.printStackTrace();
} catch (PermissionException e) {
```

```
   // Handle exception
   e.printStackTrace();
}
```

Enable Object Versioning

To enable object versioning, you must create a file bean that specifies the path to the file, and then use **`enableVersioning()`**.

Here is an example that shows the exceptions that are thrown by **`enableVersioning()`**:

```
// Enable versioning on a file

String sddFilePath = "/SDD/Testing/foo.txt";
try {

   // Construct FileBean class using the SAS Drug Dev file path
   FileBean SASDrugDevelopmentFileBean = new FileBean(SAS Drug
DevelopmentFilePath);

   // Turn versioning on using repository service
   RemoteFile versionedFile = repositoryService.enableVersioning(SAS Drug
DevelopmentFileBean);

   System.out.println("\n Versioning is enabled on " +
versionedFile.getPath());

} catch (InvalidTypeException e) {
   // Handle exception
   e.printStackTrace();
} catch (PermissionException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidNodeException e) {
   // Handle exception
   e.printStackTrace();
} catch (RemoteException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // Handle exception
   e.printStackTrace();
}
```

Check Out, Check In, and Get Version Numbers

You might need to check out a file, make changes to it, and then check it back in. When you check out a file, changes made to the file are made on a working version of the file. Only the user who has the file checked out can see the working version. Changes are not written back to the SAS Drug Development repository until the file is checked in. When the file is checked in, a new version of the file is created, and other users can see the changes that were made.

Here is an example that shows how to check out a file, change the contents of the file, and then check it back in. File version information is then printed:

```
// Checkin/Checkout a file and print file version information
String sddPath = "/SDD/Testing/foo.txt";
String checkinFilePath  = "c:\temp\foo.txt";
```

```
try {

   // Construct a FileBean with given path
   FileBean fileBean = new FileBean(sddpath);

   // Checkout the file using the repository service
   RemoteFile remoteFile = repositoryService.checkoutFile(fileBean);

   // Construct a filebean for the version to be checked in
   FileBean checkinFileBean =  new FileBean(sddpath);

   // Set contents of new version
   File localFile = new File(checkinFilePath);
   checkinFileBean.setContents(localFile);

   // Checkin a new version using the repository service
   RemoteFile checkedinFile =
repositoryService.checkinFile(checkinFileBean);

   // Get version information
   List versions = checkedinFile.getVersions();

   if ( versions != null)
   {
      System.out.println("\n File " + path + " has " + versions.size() +
" versions");

      for ( Iterator iter = versions.iterator(); iter.hasNext();)
      {
         String version = (String)iter.next();

         // Print version information
         System.out.println(version);
      }
   }

} catch (IllegalStateException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidNodeException e) {
   // Handle exception
   e.printStackTrace();
} catch (PermissionException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidTypeException e) {
   // Handle exception
   e.printStackTrace();
} catch (RemoteException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // Handle exception
   e.printStackTrace();
}
```

Get Available Object Types

To get the different object types that are supported in SAS Drug Development, use the type service. The type service is the entry point for information about the type definitions defined in SAS Drug Development. Each object type lists the available metadata for an object of that type.

Here is an example that lists all of the object types:

```
// List the different node/object types available in the SAS Drug
Development domain

try {

   TypeService typeService = serviceMgr.getTypeService();

   if (typeService != null)
   {
      List sddTypes = typeService.getTypes();

      if (sddTypes != null)
      {
         System.out.println("\n Node Types defined on SAS Drug Dev Server
: \n");
         for (Iterator iter = sddTypes.iterator(); iter.hasNext();) {
            Type type = (Type)iter.next();
            System.out.println("type name =" + type.getName());
         }
      }
   }

} catch (RemoteException e) {
   e.printStackTrace();
} catch (InvalidSessionException e) {
   e.printStackTrace();
}
```

Access Control Lists

An *access control list* (ACL) controls the access permissions to a node in the SAS Drug Development repository. Each node contains a single ACL. An ACL contains a set of access control entries.

An *access control entry* (ACE) contains a set of access permissions for a single user account or user group. The access permissions for a file are:

- Read
- Write
- Delete
- Manage

A container has the access permissions of a file and these additional access permissions:

- Inherit Read
- Inherit Write
- Inherit Delete
- Inherit Manage

The “inherit” access permissions determine the access permissions for a new node created within the container.

Add and Remove User Accounts and User Groups

Before you can specify the access permissions for a user account or user group, you must add the user account or user group to the ACL. To remove all access permissions for a user account or user group, you remove the user account or user group from the ACL.

Here is an example that shows how to manipulate an ACL by using ACEs. The example also shows how to add access permissions to an ACE, how to print the current access permissions, how to edit the access permissions, and then how to print the new access permissions.

```
// Sample code that demonstrates removing/adding of users/groups from/to
an access control list of a file in SDD repository

String sddFilePath = "/SDD/Testing/foo.txt";
String removeGroupName = "Group A";
String removeUserId = "user1";
String addUserId = "user2";
String addGroupName  = "Group B";

try {
   FileBean fileBean = new FileBean(sddFilePath);
   RemoteFile remoteFile = repositoryService.getFile(fileBean);


   // Get access control list for this file
   AccessControlList acl =
repositoryService.getAccessControlList(remoteFile);

   // Remove group from Access Control List
   if (! "".equals(removeGroupName)) {
      // remove entry corresponding to specified group name
      acl.removeGroupEntry(removeGroupName);
   }

   // Remove userid from Access Control List
   if (! "".equals(removeGroupName)) {
      // remove entry corresponding to userid to be removed from ACL
      acl.removeUserEntry(removeUserId);
   }

   // Construct set of permissions to be associated with the user/group
added to this file
   Set permissions = new HashSet();
   permissions.add(Permission.DELETE_PERMISSION);
   permissions.add(Permission.READ_PERMISSION);
   permissions.add(Permission.WRITE_PERMISSION);
   permissions.add(Permission.MANAGE_PERMISSION);

   // Add userid to Access Control List
   if (! "".equals(addUserId)) {

      // Create entry for user to be added
      AccessControlEntry newUserAce = acl.createUserEntry(addUserId);
      newUserAce.setPermissions(permissions);
      acl.setEntry(newUserAce);
   }

   // Add group to Access Control List
   if ( ! "".equals(addGroupName))
   {
      AccessControlEntry newGroupAce =
acl.createGroupEntry(addGroupName);
```

```
      newGroupAce.setPermissions(permissions);
      acl.setEntry(newGroupAce);
   }

   // Perform the Access Control List changes on the server
   repositoryService.updateFileAccessControlList(acl);


   // Retrieve Access Control List that was modified and print the access
control entry names
   AccessControlList updatedAcl =
repositoryService.getAccessControlList(remoteFile);
   List aceEntries = updatedAcl.getEntries();

   System.out.println("\nUpdated Access Control List for " + sddFilePath
+ ": \n");

   for (Iterator iter = aceEntries.iterator(); iter.hasNext();) {
      AccessControlEntry ace = (AccessControlEntry) iter.next();

      if (ace.isGroupEntry())
      {
         System.out.println("Group Entry ACE, name =" + ace.getName());
      }
      else
      {
         System.out.println("User Entry ACE, name =" + ace.getName());
      }
   }


} catch (Exception e) {
   // Handle exception
   e.printStackTrace();
}
```

Determine and Change Access Permissions

Here is an example that shows how to set the Delete and Write access permissions for a file:

```
// Get Access Control List of a file and grant delete and write
permissions
String sddFilePath = "/SDD/Testing/foo.txt";
try {
   FileBean fileBean = new FileBean(sddFilePath);
   RemoteFile remoteFile = repositoryService.getFile(fileBean);

   // Set write and delete permissions for current user
   AccessControlList acl =
repositoryService.getAccessControlList(remoteFile);
   List aceEntries = acl.getEntries();

   AccessControlEntry ace;
   for (Iterator iter = aceEntries.iterator(); iter.hasNext();) {
      ace = (AccessControlEntry) iter.next();
      if ( ! ace.hasPermission(Permission.DELETE_PERMISSION))
      {
         ace.addPermission(Permission.DELETE_PERMISSION);
      }
      if ( ! ace.hasPermission(Permission.WRITE_PERMISSION))
      {
         ace.addPermission(Permission.WRITE_PERMISSION);
      }
   }
```

```
   repositoryService.updateFileAccessControlList(acl);

   System.out.println("\nAccess Control List Updated, delete and write
permissions granted to all access control entries");

} catch (Exception  e) {
   // Handle exception
   e.printStackTrace();
}
```

User Accounts and User Groups

Overview

The user service provides methods for manipulating user accounts.

The group service provides methods for manipulating user groups.

Create a User Account

To create a user account, you must specify a user ID, password, first name, last name, and e-mail address.

Note: Before the user can access the remote API, the user must consent through the SAS Drug Development interface.

Here is an example that shows how to create a user account:

```
//Create a user with userId newuser
String userId = johndoe;
String password = "Password1! ";
String firstName = "John";
String lastName = "Doe";
String email = "johndoe@test.com;

try{
   //Construct a UserBean with the intended SAS Drug Development userId
   UserBean userBean = new UserBean(userId);

   //set the password
   userBean.setPassword(password.toCharArray());

   //set the first name
   userBean.setFirstName(firstName);

   //set the last name
   userBean.setLastName(lastName);

   //set email address
   userBean.setEmailAddress(email);

   //create this user on the server
   RemoteUser remoteUser = userService.create(userBean);

   System.out.prinln("\n Created new user" + remoteUser.getUserId());
} catch (Exception  e) {
   // Handle exception
```

```
   e.printStackTrace();
}
```

Get a User Account

To get a user account, you must specify a user ID. Here is an example that shows how to get a user account:

```
// Get a user of the specified userid from SAS Drug Development

String userId = "johndoe";
try {
   RemoteFile remoteUser = userService.get(userId);

   System.out.println("Got user: " + remoteUser.getUserId();
} catch (InsufficientPrivilegesException e) {
   // Handle exception
   e.printStackTrace();
} catch (RemoteException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // Handle exception
   e.printStackTrace();
}
```

Get the System Policies for a User Account

Here is an example that shows how to get the list of SAS Drug Development system policies for a user account:

```
// Get the list of SAS Drug Development policies assigned to this user

String userId = "johndoe";
try {
   RemoteFile remoteUser = userService.get(userId);
   List policies = remoteUser.getPolicies();
   for (Iterator iter = policies.iterator(); iter.hasNext();) {
      Policy policy = (Policy) iter.next();
      System.out.prinln(userId + "has policy: " + policy);
   }
} catch (InsufficientPrivilegesException e) {
   // Handle exception
   e.printStackTrace();
} catch (RemoteException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // Handle exception
   e.printStackTrace();
}
```

Update the System Policies for a User Account

To update the system policies for a user account, you must create a user bean that represents the user account. Then, you must update the bean.

Here are the methods to update system policies for a user account:

- addPolicy and addPolicies

  These methods add one or more system policies.
- removePolicy and removePolicies

  These methods remove one or more system policies.
- setPolicies

  This method overwrites all system policies for the user account with the system policies specified in the updated user bean.

Note: You can update the system policies only for a user account that exists on the SAS Drug Development server. If you attempt to add or remove a user account while updating the system policies, an exception is thrown.

```
//add policies to a user

String userId = "johndoe";
try {
   UserBean userBean = new UserBean(userId);
   ArrayList policies = new ArrayList();
   policies.add(Policy.USER_POLICY_OWNER_MANAGER);
   policies.add(Policy.USER_POLICY_UNDO_CHECKOUT);
   userBean.addPolicies(policies);
   userBean.removePolicy(Policy.USER_POLICY_SIGNER);

   //call update to make the changes on the server
   RemoteUser remoteUser = service.update(userBean);
   List policies = remoteFile.getPolicies();
   for (Iterator iter = policies.iterator(); iter.hasNext();) {
      Policy policy = (Policy) iter.next();
      System.out.prinln(userId + "has policy: " + policy);
   }
} catch(PropertyValidationException e)
   // Handle exception
   e.printStackTrace();
} catch(InvalidUserException e)
   // Handle exception
   e.printStackTrace();
} catch (InsufficientPrivilegesException e) {
   // Handle exception
   e.printStackTrace();
} catch (RemoteException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // Handle exception
   e.printStackTrace();
}
```

Get and Set the Properties of a User Account

You manipulate most of the properties of a user account using the get and set methods on a user bean.

However, there are other properties of a user account that you can get and set using methods associated with a remote user. These properties include optional user account information such as employee ID, title, phone number, and fax number. Other user account properties that you can get, but not set, are the last login date and the number of successful logins.

Note: If you attempt to set a property that cannot be set, an exception is thrown.

```
//update properties on user

String userId = "johndoe";
try {
   UserBean userBean = new UserBean(userId);

   //update the user's last name
   userBean.setLastName("Deer");

   //update the user's contact info
   userBean.setProperty(UserBean.PROPERTY_PHONE, "1-800-555-5555");
   userBean.setProperty(UserBean.PROPERTY_FAX, , "1-800-555-5556");

   //call update to make the changes on the server
   RemoteUser remoteUser = service.update(userBean);

   //check user's updated info
   System.out.prinln(userId + "has last name: " +
remoteUser.getLastName());
   System.out.prinln(userId + " phone number is: " +
updatedUser.getProperties().get(UserBean.PROPERTY_PHONE);
   System.out.prinln(userId + " fax number is: " +
updatedUser.getProperties().get(UserBean.PROPERTY_FAX);

} catch(PropertyValidationException e)
   // Handle exception
   e.printStackTrace();
} catch(InvalidUserException e)
   // Handle exception
   e.printStackTrace();
}catch (InsufficientPrivilegesException e) {
   // Handle exception
   e.printStackTrace();
} catch (RemoteException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // Handle exception
   e.printStackTrace();
}
```

Get a User Group

To get a user group, you must create a group bean with the name of a specific user group.

Here is an example that shows how to create a group bean, and then print the properties of the user group:

```
// Sample code to get group name and description
String groupName = "SDD";
try {
   GroupBean groupBean = new GroupBean(groupName);
   RemoteGroup remoteGroup = (RemoteGroup)groupService.get(groupBean);

   System.out.println("\nGetting name for " + groupName);
   String returnedName = remoteGroup.getName();
   String returnedDescription = remoteGroup.getDescription();

   // print out the name and description
   System.out.println("Got group: " + returnedName);
   System.out.println("Description: " + returnedDescription);

} catch (RemoteException e) {
```

```
   // Handle exception
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // Handle exception
   e.printStackTrace();
} catch(InvalidGroupException e) {
   // Handle exception
   e.printStackTrace();
} catch (InsufficientPrivilegesException e){
   // Handle exception
   e.printStackTrace();
}
```

Get the Members of a User Group

To get the members of a user group, you must create a group bean with the name of a specific user group. Then, you access the list of members within the user group.

Here is an example:

```
// Sample code to get group membership
String groupName = "SDD";
try {
   GroupBean groupBean = new GroupBean(groupName);
   RemoteGroup remoteGroup = (RemoteGroup)groupService.get(groupBean);

   System.out.println("\nGetting members for " + groupName);
   List users = remoteGroup.getUsers();

   // Iterate through the users
   for (Iterator iter = users.iterator(); iter.hasNext();) {
      String username = iter.next();

      // Do something with the user, for ex: print them out
      System.out.println(username);
   }

} catch (RemoteException e) {
   // Handle exception
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // Handle exception
   e.printStackTrace();
} catch(InvalidGroupException e) {
   // Handle exception
   e.printStackTrace();
} catch(InsufficientPrivilegesException e){
   // Handle exception
   e.printlnStackTrace();
}
```

Create a User Group

To create a user group, you must create a group bean with the name of the user group. The name must conform to certain rules, which are listed in the online SAS Drug Development Remote API reference. You can create a description and add members to the user group.

Here is an example:

```
// Create a new group
```

```
try{
   //construct a new group bean to represent the group to be created
   GroupBean groupBean = new GroupBean("newGroup");
   groupBean.setDescription("description");
   groupBean.addUser("admin");

   //call the group service to create the group on the server
   service.create(groupBean);

   //check to ensure the new group was created
   RemoteGroup group = (RemoteGroup)service.get(groupBean);
   System.out.println("New Group: " + group.getName());
} catch(InvalidSessionException e){
   //handle exception
   e.printStackTrace();
} catch(InsufficientPrivilegesException e){
   //handle exception
   e.printStackTrace();
} catch(InvalidGroupException e){
   //handle exception
   e.printStackTrace();
} catch(InvalidUserException e){
   //handle exception
   e.printStackTrace();
} catch(RemoteException e){
   //handle exception
   e.printStackTrace();
} catch(InvalidGroupBeanException e){
   //handle exception
   e.printStackTrace();
}
```

Update the Members of a User Group

To update the members (user accounts) of a user group, you must update the user group members on a group bean that has the name of the user group. Then, you issue **`update()`** to write the changes back to the SAS Drug Development repository.

Here are the methods to update members in a user group:

- addUser and addUsers

  These methods add one or more members.
- removeUser and removeUsers

  These methods remove one or more members.
- setUsers

  This method overwrites all members in the user group with the members specified in the updated group bean.

Note: You can specify only members that exist on the SAS Drug Development server. If you attempt to add or remove a member while updating the members of a user group, an exception is thrown.

```
try{
   //sample user update
   String groupName = "SDD";
   String user1 = "admin";
   String user2 = "user";
   String user3 = "user2";
```

```
      GroupBean groupBean = service.get(new GroupBean(groupName));
      groupBean.removeUser(user1);
      groupBean.addUser(user2);
      groupBean.addUser(user3);

      //call update on the service to make the changes on the server
      service.update(groupBean);

      //do something with the newly modified group
      RemoteGroup group = (RemoteGroup)service.get(new
GroupBean(groupName));

      List users = group.getUsers();
      for(Iterator iter = users.iterator(); iter.hasNext(); ){
         System.out.println("User: " + iter.next());
      }
   } catch(InvalidSessionException e){
      //handle exception
      e.printStackTrace();
   } catch(InsufficientPrivilegesException e){
      //handle exception
      e.printStackTrace();
   } catch(InvalidGroupException e){
      //handle exception
      e.printStackTrace();
   } catch(InvalidUserException e){
      //handle exception
      e.printStackTrace();
   } catch(RemoteException e){
      //handle exception
      e.printStackTrace();
   } catch(ObjectModifiedException e){
      //handle exception
      e.printStackTrace();
   } catch(InvalidGroupBeanException e){
      //handle exception
      e.printStackTrace();
   }
```

SAS Drug Development Processes

Publish a SAS Drug Development Process

To publish a SAS Drug Development process, you must provide the following information:

- process-ready SAS code that contains macro references that you want to appear as parameters in the SAS Drug Development process
- parameter information to help define the parameters in the SAS Drug Development process

You must create a file bean that specifies the SAS code content and the SAS Drug Development path information. Parameter information is specified in a set of ProcessParameter classes. The file bean and list of ProcessParameter classes are passed to the publishProcess method in the SAS service.

There are four types of parameters that can be published in an SAS Drug Development process with the SAS service:

- ProcessFolderParameter
- ProcessOutputFileParameter
- ProcessInputFileParameter
- ProcessIgnoredParameter

Here is an example:

```
String sddProcessPath = "/SDD/Testing/process.sas";

String typeName = "process";
String localFilePath = "c:\\temp\\sample.sas";

// Construct a FileBean with the SAS Drug Development path and process
file type
FileBean fileBean = new FileBean(sddProcessPath, new TypeBean(typeName));

// Set SAS code contents of the FileBean using the localFile
fileBean.setContents(new File(localFilePath));

// create ProcessParameter classes for parameters you want defined in the
SDD process
ProcessFolderParameter folderParm = new ProcessFolderParameter("folderA",
null, "/SDD/remoteAPI/testFolder", true, null, true, false, false);
ProcessOutputFileParameter outputFileParm = new
ProcessOutputFileParameter("outfile", null, "/SDD/remoteAPI/testFolder",
"testOutfile.txt", false);

List parms = new ArrayList();
parms.add(folderParm);
parms.add(outputFileParm);

RemoteFile remoteFile = null;
try {
   remoteFile = sasService.publishProcess(fileBean, parms);
} catch (RemoteException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (PermissionException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (InvalidProcessException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (InvalidNodeBeanException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (ValidationException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (InvalidTypeException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (ContainmentException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (InvalidParameterException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
}
```

Examine the Parameters of a SAS Drug Development Process

To examine the parameters of a SAS Drug Development process, you must call the getProcessParameters method in the SAS service. A ProcessParameter class is returned for every parameter that is defined in the process, and is one of these four types:

- ProcessFolderParameter
- ProcessOutputFileParameter
- ProcessInputFileParameter
- ProcessIgnoredParameter

If the parameter is not one of these four types, then it returns:

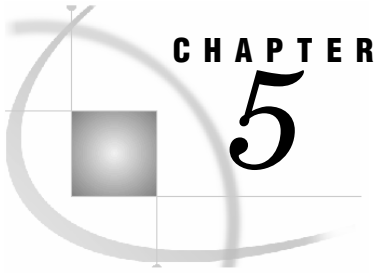
- ProcessUnknownParameter

Here is an example:

```
String sddProcessPath = "/SDD/Testing/process.sas";
List publishedParms = null;

try {
   publishedParms = sasService.getProcessParameters(new
FileBean(sddProcessPath));
} catch (RemoteException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (InvalidSessionException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (PermissionException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (InvalidNodeException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
} catch (InvalidProcessException e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
}

if (publishedParms ! = null){
   for (Iterator iter=publishedParms.iterator(); iter.hasNext();){
      ProcessParameter processParm = (ProcessParameter)iter.next();
      System.out.println("Process has parameter: " +
processParm.getName());
   }
```



CHAPTER 5

Troubleshooting Connection Problems

Overview

There are many causes of connection problems. The most common solutions are listed in this chapter. There is additional text that appears in the error message. Use this additional text to determine the root cause of the error.

Unable to Find Server

If the connection fails because the URL could not be resolved as a valid remote API connection, the following error message appears:

```
com.sas.drugdev.remote.session.RemoteException
```

To correct this problem, when you are prompted for the SAS Drug Development server URL, user name, and password, ensure that you have specified the server URL as shown here:

```
https://<your_server>/SDDremote
```

Note: You must use **`https`**.

No Trusted Certificate Found

If the connection fails because the secure sockets layer (SSL) certificate is invalid (it is out of date or for the wrong host name), or the certificate authority (CA) is not in the certification chain list, the following message appears:

```
Exception executing authentication attempt:
sun.security.validator.ValidatorException: No trusted certificate
```

To correct this problem, contact your SAS Drug Development site administrator to request that a valid certificate be used for the server, or that the required CA be installed.

Server Is Not Configured

If the connection fails because the server is not configured properly, the following message appears:

```
com.sas.drugdev.remote.RemoteException: The url is not valid. at
com.sas.drugdev.remote.client.SessionRemote.connect
```

To correct this problem, verify that the SDDDrugDevRemoteAPIServer Web application is installed on the server. Then, verify that the SDDDrugDevRemoteAPIServer Web application is deployed.

Authentication Exception Null

If the connection fails because your user ID or password is invalid, the following message appears:

```
com.sas.drugdev.remote.session.AuthenticationException
null
```

To correct this problem, provide valid credentials or verify that you have access to the SAS Drug Development user interface.

Expired Password

If the connection fails because your password has expired, the following message appears:

```
com.sas.drugdev.remote.session.PasswordExpiredException
null
```

To correct this problem, ensure that you are using the correct password. If your password has expired, ask your internal SAS Drug Development project manager to reset your password.

Inactive User ID

If the connection fails because your user ID is inactive, the following message appears:

```
com.sas.drugdev.remote.session.UserInactiveException
null
```

To correct this problem, ensure that you are using the correct user ID. If your user ID is inactive, ask your internal SAS Drug Development project manager to reactivate your password.

Retired User ID

If the connection fails because your user ID is retired, the following message appears:

```
com.sas.drugdev.remote.session.UserRetiredException
null
```

To correct this problem, ensure that you are using a correct active user ID. If your user ID is retired, ask your internal SAS Drug Development project manager to create a new user account. Once a user account is retired, it cannot be reused.

Connection Time Out

If the connection times out because the network is not allowing access to the specified URL or because there is a possible proxy issue, the following message appears:

```
Exception executing authentication attempt: Connection timed out: connect
```

To correct this problem, try one of the following solutions:

- Verify that you have Internet access.
- Verify that you have access through the SAS Drug Development user interface.
- Verify the proxy requirements and match the settings as defined in your Web browser. You might need to contact your network administrator for proxy information.

Your Turn

We welcome your feedback.

- If you have comments about this book, please send them to **`yourturn@sas.com`**. Include the full title and page numbers (if applicable).
- If you have comments about the software, please send them to **`suggest@sas.com`**.