IBM FileNet Content Manager 5.0: Java API Programming

(Course code F143)

Student Exercises

ERC 2.0

Trademarks

IBM® and the IBM logo are registered trademarks of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

DB2® FileNet® WebSphere®

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux® is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Other product and service names might be trademarks of IBM or other companies.

May 2011 edition

Contents

Trademarks

The reader should recognize that the following terms, which appear in the content of this training document, are official trademarks of IBM or other companies:

IBM® and the IBM logo are registered trademarks of International Business Machines Corporation.

The following are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide:

DB2® FileNet® WebSphere®

Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux® is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Windows is a trademark of Microsoft Corporation in the United States, other countries, or both.

Other product and service names might be trademarks of IBM or other companies.

Unit 1. Set Up Eclipse IDE

Unit overview

This unit contains these activities.

Activities

How the activities are organized in this course

- This course has instructions for executing code using either Eclipse IDE or a text editor such as Notepad and running the code using the Windows Command Prompt.
- Each unit in this course uses its own Java class with a method for each functionality.
- Each Java class is a stand-alone program.

Activity 1: Set up the Eclipse IDE

Procedures

Procedure 1: Create an Eclipse project

1. Start Eclipse by double-clicking the icon on the desktop.
   a. Click OK in the “Select a workspace” window to accept the default value.
2. Click File > New > Project.
   a. In the “Select a wizard” window, select Java Project.
   b. If the Java Project option is not at the root level, expand the Java folder and select Java Project.
   c. Click Next.
3. In the Create a Java Project window, complete the following steps:
   a. Enter a name for the “Project name” field. Example: `CMJavaAPI`
   b. Ensure that “Create new project in workspace” is selected in the Contents area.
   c. Select the “Use a project specific JRE” option in the JRE area and select `JavaAPIJRE` from the list.
   d. Click Next.
   e. Leave the window open for the next procedure.

Note

The `JavaAPIJRE` has been created so that you use the same JRE version that WebSphere uses. If you need to create a specific JRE use the following steps:

1. In the Create a Java Project window, click Configure JREs link in the JRE area.
2. In the Preferences (Filtered) window, click Add.

3. In the Add JRE window, do the following steps:
   a. Select Standard VM and click Next.
   b. Enter the following directory for the “JRE home” field:

      C:\Program Files\IBM\WebSphere\AppClient\java\jre

      Or, you can click Directory and browse to the folder.
   c. Enter a name in the JRE name field. Example: JavaAPIJRE
   d. Click Finish to close the Add JRE window.
4. Click OK in the Preferences (Filtered) window to close it.

Procedure 2: Add library (JAR) files to the project

1. In the Java Settings window, click the Libraries tab.
2. Click the “Add External JARs” button.
   a. In the “JAR Selection” window, browse to the folder listed in the following table.

Name of JAR	Location
Jace.jar log4j.jar	C:\CE_API\lib

   b. Select the JAR files listed in the data table, and click Open.
3. Click Finish in the Java Settings window.
4. If you are prompted with the message “Open Associated Perspective?” window, click Yes.

Procedure 3: Configure the Java compiler setting

1. If the Package Explorer window is not already open, click the “Show View as a fast view” icon at the lower left corner of the window and select Package Explorer.
2. In the Package Explorer window, right-click your project, and click Properties.

   The Properties window for the project opens.
3. Select the “Java Compiler” node in the tree view in the left pane.
4. Select the “Use default compliance settings” check box in the JDK Compliance area in the right pane.
5. Verify or select 1.6 from the “Compiler compliance level” list.
6. Click Apply.
7. Leave the Properties window open for the next procedure.

Procedure 4: Set Java Build path (Optional)

Note

If you do not select a folder, the class files are added to the default location.

1. In the Properties window, select the “Java Build Path” from the tree view in the left pane.
2. Click the Source tab in the right pane.
3. In the “Default output folder” field (at the bottom of the screen), click Browse.
4. In the Folder Selection window, click “Create New Folder” to create a new folder.
   a. Type `classes` in the Folder name box.
   b. Click OK.
   c. In the Folder Selection window, select the classes folder.
   d. Click OK to close the Folder Selection window.
5. Click OK in the Properties window.
6. Click Yes if you are prompted with the output folder information.

Procedure 5: Create a package

1. In the Package Explorer window, select your project > src node, right-click, and click New > Package.
2. In the New Java Package window, type `com.ibm.filenet.edu` in the Name field, and then click Finish.

Note

You create and save your Java source files in this package.

Procedure 6: Add supporting files to the project

1. In the Package Explorer window, right-click your project and click Import.
   a. In the Import window, expand the folder called General and select the File System option.
   b. Click Next.
   c. Click Browse next to the “From directory” field at the top, go to the folder listed in the table below, and select it.

d. Click OK.

Name of the files	Folder location
VMArgumentsforEclipse.txt Model 200.GIF Model 400.JPG annotationText.txt	C:\CMJavaAPIProg\Source

e. Select the check box next to the files listed in the data table, and then click Finish.

2. Optionally, you can add Java solution files to your project by doing the following steps:

   a. In the Package Explorer window, select your package (com.ibm.filenet.edu), right-click, and click Import.

   b. In the Import window, open the folder called General, select the File System option, and click Next.

   c. Click Browse next to the “From directory” field at the top, go to the `C:\CMJavaAPIProg\Solution` folder, and select it.

   d. Click OK.

   e. Select the check box next to the Solution folder to select all the files.

   f. Click the “Filter Types” button.

   g. In the Select Types window, select this type: `.java`

   h. Click OK.

   i. In the Import window, click Finish.

   j. Verify that all the Java files are added to your project.

Procedure 7: Create a Java file

1. Right-click the package that you just created, and click New > Class.
2. In the New Java Class window, enter the values listed in the following data table.

Field Name	Value
Source folder	Accept the default value (<Your project name>/src)
Package	Accept the default value (com.ibm.filenet.edu)
Name	Name of the Java class that you want to create **Tip:** The `.java` extension is not needed.
Modifiers	public
Superclass	Accept the default value (java.lang.Object)
Which method stubs would you like to create?	public static void main(String[] args)

3. Click Finish to close the window.

Procedure 8: Enter VM arguments and execute the code

In this procedure, you provide VM arguments in Eclipse and execute the code.

1. Open the configurations:
   a. Click Run (white arrow in green circle) > Run Configurations to run the code.

      The “Create, manage, and run configurations” window opens.

   b. Double-click the Java Application in the left pane.
   c. Eclipse keeps the Java class that was run previously in cache. You need to choose the main class that you want to run every time. Verify that Main class field in the right pane has your Java class name as the value.

      Example: `com.ibm.filenet.edu.`<Name of the Java class that you want to run>
   d. Click the Arguments tab.
2. Enter the VM arguments:
   a. In the Arguments tab, clear any entry that you find in the “VM arguments” field.
   b. Copy and paste the content from the following file into the “VM arguments” field:

      `C:\CMJavaAPIProg\Source\VMArgumentsforEclipse.txt`
   c. Click Apply to save the changes.
   d. Click Run to execute the code.

Important

The values given in the following table are for informational purposes only. To avoid any typing or formatting error, use the VMArgumentsforEclipse.txt file provided on your student system to copy and paste these values.

Values for VM Arguments (C:\CMJavaAPIProg\Source\VMArgumentsforEclipse.txt)
-Dcom.ibm.CORBA.ConfigURL="file:C:\Program Files\IBM\WebSphere\AppClient\properties\sas.client.props" -Djava.security.auth.login.config="file:C:\CE_API\config\jaas.conf.WebSphere" -Djava.naming.provider.url=iiop://ccv01135:2809 -Djava.ext.dirs="C:\Program Files\IBM\WebSphere\AppClient\java\jre\lib;C:\Program Files\IBM\WebSphere\AppClient\java\jre\lib\ext;C:\Program Files\IBM\WebSphere\AppClient\lib;C:\Program Files\IBM\WebSphere\AppClient\plugins"

Troubleshooting

If you receive the following error when you execute the code, it is most likely due to VM arguments missing for the Java class:

```
com.filenet.api.exception.EngineRuntimeException: FNRCA0032E: API_UNEXPECTED_JNDI_ERROR:
The JNDI cannot be accessed.
at com.filenet.apiimpl.util.SessionLocator.locateEJBByPath(SessionLocator.java:879)
at com.filenet.apiimpl.util.SessionLocator.findEJBSessionByPath(SessionLocator.java:814)
...
```

Follow the steps in Procedure 8 step 2 to enter the VM arguments for your Java class.

Procedure 9: Edit the sas.client.props properties file

Important

Before executing the code, verify that this file has been edited.

1. Locate the following folder:

   C:\Program Files\IBM\WebSphere\AppClient\properties

2. Open the sas.client.props file with a text editor (example: Notepad).
3. Set the values for the entries using the values listed in the following data table.

sas.client.props entries	Value
com.ibm.CORBA.sercurityServerHost	ccv01135
com.ibm.CORBA.sercurityServerPort	2809
com.ibm.CORBA.loginSource	none (type this value)

Unit 2. Communication with the Content Engine Server

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class from this unit is used in all other units for the Content Engine Connection.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Windows Command Prompt.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Windows Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference.

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.

Lesson 2.1. Get a connection to a FileNet P8 domain

Overview

Why is this lesson important to you?

Your company manages its content using the IBM FileNet Content Manager. As their programmer, you are going to write code to communicate with the Content Engine server by getting a connection to a FileNet P8 domain.

Activities

- Write code to get a connection to a FileNet P8 domain.

Prerequisites

Edit the Property File: Activity, page 2-7

Skill levels

Get a connection to a FileNet P8 domain: Challenge, page 2-9

Get a connection to a FileNet P8 domain: Walkthrough, page 2-11

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Edit the Property File: Activity

Important

This activity is required for both skill levels. You can omit this activity if you have already edited this file in the previous unit. If the information provided in the following table is not in the file, you receive login errors at run time.

1. In your client system, use a text editor to open the `C:\Program Files\IBM\WebSphere\AppClient\properties\sas.client.props` file.
2. Set the values for the entries listed in the data table.

sas.client.props entries	Value
com.ibm.CORBA.sercurityServerHost	ccv01135
com.ibm.CORBA.sercurityServerPort	2809
com.ibm.CORBA.loginSource	none (type none)

3. Save the file.

Get a connection to a FileNet P8 domain: Challenge

Challenge

Use the data in the table to create a Java class and write code to get a connection to a FileNet P8 domain. Include a line of code to display a message when the connection is successful.

Data

Item	Value

Packages to import	`java.util.Iterator`
	`javax.security.auth.Subject`
	`com.filenet.api.collection.ObjectStoreSet`
	`com.filenet.api.constants.*`
	`com.filenet.api.core.*`
	`com.filenet.api.util.UserContext`

User id	`"p8admin"`
Password	`"IBMFileNetP8"`
Universal Resource Identity (URI)	`"iiop://ccv01135:2809/FileNet/Engine"`

Verification

- Run the program using Eclipse or the Windows Command Prompt.
- Verify that the message indicating that connection is successful is displayed in the Command Prompt window or in the console of Eclipse.

Get a connection to a FileNet P8 domain: Walkthrough

Procedures

Procedure 1: Create a Java class

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the “New Java Class” window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type the class name `CEConnectionEDU` in the Name field.
5. Select `public static void main(String[] args)` for “Which method stubs would you like to create?” option.
   a. Clear other options.
6. Click Finish to add the class and close the window.
   a. Verify that the new class is listed under your package in the Project Explorer.
7. Write code inside the class that you just added.
8. Import the following packages before the declaration of the class:

```
java.util.Iterator
javax.security.auth.Subject
com.filenet.api.collection.ObjectStoreSet
com.filenet.api.core.*
com.filenet.api.util.UserContext
```

9. Inside the main method, write a `try-catch` block.
   a. Call `exception.printStackTrace()` in the catch block to display the exceptions thrown.
   b. Add a custom methods and call it from the main method as described in the following procedures.

Procedure 2: Write a method to get the connection

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getCEConnectionEDU`
   - Parameters:
     - `String` for username
     - `String` for password
   - Returns: `Connection` object
2. Write the following lines of code inside the method block.
3. Create a connection object:
   a. Call `Factory.Connection.getConnection(…)`
      - Parameter: `"iiop://ccv01135:2809/FileNet/Engine"`
   b. Assign the return value to a variable of type `Connection`.
4. Call `UserContext.createSubject(…)`
   - Parameters:
     - `Connection` object from step 3
     - `String` value for username that is passed to this method
     - `String` value for password that is passed to this method
     - `"FileNetP8"`

   a. Assign the return value to a variable of type `Subject`.
5. Get the UserContext:
   a. Call `UserContext.get()`
   b. Assign the return value to a variable of type `UserContext`.
6. Call `userContext.pushSubject(…)` on the variable from the step 4.
   - Parameter: `Subject` object from step 3.

   a. Return the Connection object.
7. Call `System.out.println(…)`to display a message “Got the connection”.
8. Close the method.

Procedure 3: Instantiate the class and call the method

1. Inside the `try` block of the main() method, create an instance of the class.

   Example: `CEConnectionEDU connectInstance = new CEConnectionEDU();`

2. Call `getCEConnectionEDU(...)`

- Parameters:
  `"p8admin"`
  `"IBMFileNetP8"`

  a. Assign the returned `Connection` object to a variable.

Procedure 4: Run the program using Eclipse

1. Before running the code, do the following:
   a. Ensure that you have edited the sas.client.props file.
   b. Ensure that Content Engine is running. If it is not, start it following the instructions in the System Check section.
   c. If you are running the code in the Command Prompt, see Procedure 5, Execute the code in the Command Prompt, page 2-14.
2. Open the configurations:
   a. Click Run (white arrow in green circle) > Run Configurations to run the code.

      The “Create, manage, and run configurations” window opens.
   b. Double-click the Java class that you want to run in the left pane.
   c. Eclipse keeps the Java class that was run previously in cache. You need to choose the main class that you want to run every time. Verify that Main class field in the right pane has your Java class name as the value. If necessary, type the value.

      Example: `com.ibm.filenet.edu.CEConnectionEDU`
   d. Click the Arguments tab.
3. Enter the VM arguments:
   a. In the Arguments tab, clear any entry that you find in the “VM arguments” field.
   b. Copy and paste the content from the following file into the “VM arguments” field:

      `C:\CMJavaAPIProg\Source\VMArgumentsforEclipse.txt`

Important

The values given in the following table are for informational purposes only. To avoid any typing or formatting error, use the VMArgumentsforEclipse.txt file provided on your student system to copy and paste these values.

Values for VM Arguments (C:\CMJavaAPIProg\Solution\VMArgumentsforEclipse.txt)
-Dcom.ibm.CORBA.ConfigURL="file:C:\Program Files\IBM\WebSphere\AppClient\properties\sas.client.props" -Djava.security.auth.login.config="file:C:\CE_API\config\jaas.conf.WebSphere" -Djava.naming.provider.url=iiop://ccv01135:2809 -Djava.ext.dirs="C:\Program Files\IBM\WebSphere\AppClient\java\jre\lib;C:\Program Files\IBM\WebSphere\AppClient\java\jre\lib\ext;C:\Program Files\IBM\WebSphere\AppClient\lib;C:\Program Files\IBM\WebSphere\AppClient\plugins"

4. Click Apply to save the changes.
5. Click Run to execute the code.
6. Verify that the message “Got the connection” is displayed in the Output window.

Troubleshooting

If you receive the following error:

```
com.filenet.api.exception.EngineRuntimeException: FNRCA0032E:
API_UNEXPECTED_JNDI_ERROR: The JNDI cannot be accessed.
```

It is most likely that the VM arguments are missing for this Java class or they are incorrect. Replace the text with the correct VMArguments and run it again.

Procedure 5: Execute the code in the Command Prompt

1. Save the file in the C:\CMJavaAPIProg\Source folder where the make.bat and run.bat files are stored.
   a. Make sure that the file name is the same as the class name (file names are case-sensitive) and that it has a .java extension.
2. Compile the code using the make.bat file:
   a. Open the Command Prompt window.
   b. At the command prompt, change the directory to C:\CMJavaAPIProg\Source.
   c. Compile the file by running the make.bat file: `make CEConnectionEDU.java`
3. Run the code using the run.bat file:
   a. At the command prompt, type the following:

      ```
      run com.ibm.filenet.edu.CEConnectionEDU
      ```


Lesson 2.2. Retrieve the domain and the object stores

Overview

Why is this lesson important to you?

You need to access the Domain object to retrieve other objects in the domain. As a programmer for the Content Manager, you are going to write code to retrieve the Domain and the ObjectStore objects.

Activities

- Write code to retrieve the Domain and the ObjectStore objects.

Skill levels

Retrieve the domain and the object stores: Challenge, page 2-17

Retrieve the domain and the object stores: Walkthrough, page 2-19

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Retrieve the domain and the object stores: Challenge

Challenge

Write code to retrieve the Domain and the ObjectStore objects using the same Java class that you created in the previous lesson.

Data

Item	Value
Domain Name	"P8Domain"

Verification

- Run the program using the Eclipse or the Command Prompt.
- Verify that the name of the domain and the names of the object stores are displayed in the Command Prompt window or in the console of Eclipse.

Sample output

```
Got the connection
Name of the Domain: P8Domain
Available Object Stores:
P8ConfigObjectStore
LoanProcess
Shared
Operations
Development
Sales
```


Retrieve the domain and the object stores: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to retrieve the domain

1. Define the method using the following signature:

- Scope: `public`
- Name: `getDomainEDU`
- Parameter: `Connection` object
- Returns: `Domain` object

2. Call `Factory.Domain.fetchInstance(…)`.

- Parameters:
  - `Connection` object that is passed to this method
  - `"P8domain"` for the domain name
  - `null`

   a. Assign the return value to a variable of type `Domain`.

3. Display the domain name by calling `System.out.println(…)`.

- Parameter: `domain.get_Name()`

4. Return the `Domain` object from step 2.
5. Save the file.

Procedure 2: Write a method to retrieve the object stores

1. Define the method using the following signature:

- Scope: `public`
- Name: `getObjectStoresEDU`
- Parameter: `Domain` object

2. `Call domain.get_ObjectStores()`on the Domain object that is passed into this method.

   a. Assign the return value to a variable of the type `ObjectStoreSet`.

3. Call `ObjectStoreSet.iterator()`.
   a. Assign the returned value to a variable of the type `Iterator`.
4. Write a `while` loop and iterate through the collection.
   `while(iterator.hasNext())`
5. Call `iterator.next()` within the while loop.
   a. Typecast the return value into an `ObjectStore` type and assign it to a variable.
   b. Display each object store name by calling `System.out.println(…)`.
      - Parameter: `objectStore.get_Name()`

Procedure 3: Call the methods inside the main() method

1. Call `getDomainEDU(...)` method inside the `try` block of the main() method.
   - Parameter: the `Connection` object variable that you got in the previous lesson
   a. Assign the return value to a variable of type `Domain`.
2. Call `getObjectStoresEDU(…)`.
   - Parameter: the `Domain` object from step 1
3. Save the file.

Procedure 4: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.
2. Verify that the name of the domain and the names of a list of object stores are displayed in the Command Prompt window or in the console of Eclipse.

Sample output

```
Got the connection
Name of the Domain: p8Domain
Available Object Stores:
P8ConfigObjectStore
LoanProcess
Shared
Operations
Development
Sales
```


Solution code for ConnectionEDU.java

```
package com.ibm.filenet.edu;

import java.util.Iterator;
import javax.security.auth.Subject;
import com.filenet.api.collection.ObjectStoreSet;
import com.filenet.api.core.*;
import com.filenet.api.util.UserContext;


public class CEConnectionEDU {

public Connection getCEConnectionEDU(String username, String password)
{
      String uri = "iiop://ccv01135:2809/FileNet/Engine";
      //"http://ccv01135:9080/wsi/FNCEWS40DIME/";
      Connection conn = Factory.Connection.getConnection(uri);
      String stanza = "FileNetP8"; //FileNetP8WSI";
      Subject subject = UserContext.createSubject(conn, username, password,
      null);
      UserContext uc = UserContext.get();
      uc.pushSubject(subject);
      System.out.println("Got the connection");
      return conn;
}

public  Domain getDomainEDU(Connection conn)
{
   String domainName = null; //"P8Domain";
   Domain domain = Factory.Domain.fetchInstance(conn, domainName, null);
   System.out.println("Name of the domain: "+ domain.get_Name());
   return domain;
}
@SuppressWarnings("unchecked")
public  void getObjectStoresEDU(Domain domain)
{
   ObjectStoreSet osSet = domain.get_ObjectStores();
   ObjectStore store;
   Iterator iterator = osSet.iterator();
   Iterator<ObjectStore> osIter = iterator;
   Iterator osIter = osSet.iterator();
   System.out.println("Available Object Stores:  ");
   while (osIter.hasNext())
```

```
   {
      store = (ObjectStore)osIter.next();
      if ((store.getAccessAllowed().intValue() &
         AccessLevel.USE_OBJECT_STORE.getValue()) > 0)
         System.out.println(store.get_Name());
   }
}

public ObjectStore getObjectStoreEDU (Domain domain, String objectStoreName)
{
   ObjectStore store = Factory.ObjectStore.fetchInstance(domain,
   objectStoreName,null);
   System.out.println("Name of the objectstore: "+ store.get_Name());
   return store;
}

public Folder getFolderEDU(ObjectStore store, String folderName){
   Folder folder= Factory.Folder.fetchInstance(store,folderName, null);
   folderName = folder.get_FolderName();
   System.out.println(folderName + "  folder is retrieved");
   return folder;
}
public static void main(String[] args)
{
   try{
      CEConnectionEDU connectInstance = new CEConnectionEDU();
      Connection conn =
      connectInstance.getCEConnectionEDU("P8admin","IBMFileNetP8");
      Domain domain = connectInstance.getDomainEDU(conn);
      connectInstance.getObjectStoresEDU(domain);
   } catch (Exception e) {
      e.printStackTrace();
   }
}
}
```


Unit 3. Folders

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class in this unit extends the CEConnectionEDU class from the “Communication with the Content Engine” unit.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference.

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 3.1. Create and delete Folder objects

Overview

Why is this lesson important to you?

The accounting department of your company wants to create a folder for each customer to store their ordering information. As their programmer, you are going to add a folder feature that allows users to create and delete folders.

Activities

- Write code to create a folder object
- Write code to delete a folder object

Skill levels

- Create and delete Folder objects: Challenge, page 3-7
- Create and delete Folder objects: Walkthrough, page 3-9

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Create and delete Folder objects: Challenge

Challenge

- Add two methods to the existing CEConnectionEDU.java class (that you created in the previous unit) to retrieve an object store and to retrieve a folder.
- Use the data in the table to create a Java class that extends CEConnectionEDU and write code to do the following:
  - Create a subfolder under the root folder.
  - Create a subfolder under the new folder that you created in the preceding step.
  - Delete the subfolder that you created in the preceding step.

Data

Item	Value

Packages to import	`java.util.Iterator`
	`com.filenet.api.collection.*`
	`com.filenet.api.constants.*`
	`com.filenet.api.core.*`

Object store name	`"Development"`

Verification

- Run the program using the Eclipse or the Command Prompt.
- Verify that the folders are created using Content Engine Enterprise Manager or Workplace.
- Run the code to delete the subfolder that you created and verify that the folder was deleted.

Sample output for creating folders

```
Name of the domain: P8Domain
Name of the objectstore: Development
JavaFolder is created.
ASubFolder is created.
```

Sample output for deleting a folder

```
Name of the domain: P8Domain
Name of the objectstore: Development
ASubFolder  folder is Deleted.
```


Create and delete Folder objects: Walkthrough

Procedures: Create Folder objects

Procedures: Delete Folder objects

Create Folder objects

Note

Add the methods described in procedures 1 and 2 to the `CEConnectionEDU.java` file that you created in the “Communication with the Content Engine Server” unit.

Procedure 1: Write a method to retrieve an object store

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Open the `CEConnectionEDU.java` file and write code inside the class block.

3. Define the method using the following signature:

- Scope: `public`
- Name: `getObjectStoreEDU`
- Parameters:
  - `Domain` object
  - `String` for the object store name
- Returns: `ObjectStore` object

4. Call `Factory.ObjectStore.fetchInstance(...)`.

- Parameters:
  - `Domain` object
  - object store name that is passed in to this method
  - `null`

    a. Assign the return value to a variable of type `ObjectStore`.

5. Display the object store name by calling `System.out.println(…)`.

- Parameter: `objectStore.Name`

6. Return the `ObjectStore` object from step 4.

Procedure 2: Write a method to retrieve a Folder

1. Define the method using the following signature:

- Scope: `public`
- Name: `getFolderEDU`
- Parameters:
  - `ObjectStore` object
  - `String` for the folder name with folder path
- Returns: `Folder` object

2. Call `Factory.Folder.fetchInstance(...)`.

- Parameters:
  - `ObjectStore` object that is passed in to this method
  - folder name with path that is passed in to this method
  - `null`

  a. Assign the return value to a variable of type `Folder`.

3. Display the folder name by calling `System.out.println(…)`.

- Parameter: `Folder.get_FolderName()`

4. Return the `Folder` object from step 2.

5. Leave the Eclipse open for the next procedure.

Procedure 3: Create a Java class that extends CEConnectionEDU

1. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
2. In the "New Java Class" window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
3. Type the class name `FoldersEDU` in the Name field.
4. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
5. Select `public static void main(String[] args)` for "Which method stubs would you like to create?" option.
   a. Clear other options.
6. Click Finish to add the class and close the window.
   a. Verify that the new class is listed under your package in the Project Explorer.
7. Write code inside the class that you just added.
8. Import the following packages before the declaration of the class:

```
java.util.Iterator
com.filenet.api.collection.*
com.filenet.api.constants.*
com.filenet.api.core.*
```

9. Inside the main method, write a `try-catch` block.
   a. Call `exception.printStackTrace()` in the catch block to display the exceptions thrown.
10. Add custom methods and call them from the main method as described in the following steps.

Procedure 4: Write a method to add a folder under the root folder

Write code inside the class block.

1. Define the method using the following signature:

- Scope: `public`
- Name: `createFolderEDU`
- Parameters:
  - `ObjectStore` object
  - `String` for the folder name
- Returns: `Folder` object

2. Call `Factory.Folder.createInstance(...)`.

- Parameters:
  - `ObjectStore` object
  - `null`

   a. Assign the return value to a variable of type `Folder`.

3. Call `folder.set_FolderName(…)` on the object from step 2.

- Parameter: folder name passed into this method

4. Call `objectStore.get_RootFolder()`.

   a. Assign the return value to a variable of type `Folder`.

5. Call `folder.set_Parent(...)` on the object from step 2.

- Parameter: root folder that you retrieved in step 4.

6. Save the changes to the folder by calling the `folder.save(...)` method.

- Parameter: `RefreshMode.REFRESH`

7. Return the `Folder` object from step 2.

Procedure 5: Write a method to add a subfolder under a folder

This procedure uses a different API method to create a folder.

1. Define the method using the following signature:

- Scope: `public`
- Name: `createSubFolderEDU`
- Parameters:
  - `Folder` object to be a parent folder
  - `String` for the folder name

2. Call `folder.createSubFolder(...)`on the parent folder that is passed into this method.

- Parameter: folder name that is passed into this method

   a. Assign the return value to a variable of type `Folder`.

3. Save the changes to the object by calling the `folder.save(...)` method.

   a. Parameter: `RefreshMode.REFRESH`

4. Return the `Folder` object from step 2.

Procedure 6: Instantiate the class and call the methods that you wrote

1. Inside the `try` block of the main() method, create an instance of the class.

   Example: `FoldersEDU folderInstance = new FoldersEDU();`

2. Call the getCEConnectionEDU(...) method of the CEConnectionEDU class:

- Parameters:
  "p8admin"
  "IBMFileNetP8"

   a. Assign the returned Connection object to a variable.

3. Call the getDomainEDU(…)method of the CEConnectionEDU class.

- Parameter: Connection object from step 2.

   a. Assign the returned Domain object to a variable.

4. Call the getObjectStoreEDU(…) method of the CEConnectionEDU class:

- Parameters:
  - Domain object from step 3.
  - Object store name from the following data table

   a. Assign the returned ObjectStore object to a variable.

Item	Value
Object store name	"Development"
Folder name	"JavaFolder"
Sub folder name	"ASubFolder"

5. Call the createFolderEDU(…) method.

- Parameters:
  - ObjectStore object from step 4
  - Folder name from the preceding table

   a. Assign the returned Folder object to a variable.

6. Call the createSubFolderEDU(…)method.

- Parameters:
  - Folder object that you got in the preceding step
  - Subfolder name from the preceding table

   a. Assign the returned Folder object to a variable.

Procedure 7: Run the program and verify the results

1. Run the program using the Eclipse or the Command Prompt as instructed in the "Communication with the Content Engine Server" unit.
2. Verify the results in the Development object store using the Workplace:
   a. Verify that the JavaFolder is created under the root folder.
   b. Verify that the ASubFolder is created under the JavaFolder.

Sample output

```
Name of the domain: P8Domain
Name of the objectstore: Development
JavaFolder is created.
ASubFolder is created.
```


Delete Folder objects

Procedure 1: Write a method to delete a folder

Use the same Java class that you created in the previous activity.

1. Define the method using the following signature:

- Scope: `public`
- Name: `deleteFolderEDU`
- Parameters:
  - `Objectstore` object
  - `String` for the folder path

2. Call `Factory.Folder.fetchInstance(...)`.

- Parameters:
  - `Objectstore` object
  - Folder path that are passed into this method
  - `null`

3. Optionally, get the folder name by calling `folder.get_FolderName()` and assign it to a variable of type `String`.
4. Call `folder.delete()`.
5. Save the changes to the object by calling the `folder.save(...)` method.

- Parameter: `RefreshMode.REFRESH`

6. Display the name of the folder that is deleted by calling the `System.out.println(...)` method.

- Parameter: `String` from step 3

Procedure 2: Call the method inside the main() method

1. Call the `deleteFolderEDU(...)`method inside the `try` block of the main() method.

- Parameters:
  - `Objectstore` variable that you got in the previous activity
  - `"/JavaFolder/ASubFolder"` (folder path for the folder that you created in the previous activity)

2. Comment out the methods that are not tested in this activity.
3. Save the file.

Procedure 3: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.
2. Use Content Engine Enterprise Manager or Workplace to verify that the ASubFolder is deleted under the root folder > JavaFolder of the Development object store.

Sample output

```
Name of the domain: P8Domain
Name of the objectstore: Development
ASubFolder  folder is Deleted.
```


Lesson 3.2. Retrieve objects from a folder

Overview

Why is this lesson important to you?

The billing section in your company wants to retrieve customer information from the customer folder. As their programmer, you are going to write code to retrieve the contents of a folder when an employee accesses the folder.

Activities

- Write code to retrieve subfolders from a folder.
- Write code to retrieve Document objects from a folder.
- Write code to retrieve all the objects from a folder.

Skill levels

- Retrieve objects from a folder: Challenge, page 3-19
- Retrieve objects from a folder: Walkthrough, page 3-21

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Retrieve objects from a folder: Challenge

Challenge

Use the same Java class that you created in the previous lesson. Use the following data table to write code to retrieve the subfolders, documents, and objects from a parent folder. Display the names of the objects retrieved.

Data

Item	Value
Object store name	"Development"
Path of the parent folder to retrieve the subfolders	"/Products"
Path of the parent folder to retrieve the documents	"/Manuals"
Path of the parent folder to retrieve the objects	"/Research"

Verification

- Run the program using the Eclipse or the Command Prompt.
- Verify that the name of the subfolders, documents, and objects are displayed in the Command prompt window or in the console of Eclipse.
- Verify that the “Luxury Models” folder is displayed as hidden.

Sample output for subfolders

```
Name of the domain: P8Domain
Name of the Object store: Development

List of sub folders under the 'Products' folder:
Basic Models
Deluxe Models
The “Luxury Models” folder is hidden.
Model 120
Model 200
Model 220
Model 300
Model 320
Model A
Model B
Model C
```

Sample output for documents

```
List of documents under the 'Manuals' folder:
User Manual for Basic Model A.doc
User Manual for Basic Model A.pdf
User Manual for Basic Model B.pdf
User Manual for Basic Model B.doc
User Manual for Basic Model C.pdf
User Manual for Basic Model C.doc
```

Sample output for objects

```
List of objects under the 'Research' folder:
-----------------
class= Customer
display Name =CustomerTest
-----------------
class= CustomObject
display Name =JavaCustomObj
-----------------
class= Document
display Name =JavaDoc
-----------------
class= Product
display Name =Research Information Basic Model A.doc
-----------------
class= Product
display Name =Research Information Basic Model B.doc
-----------------
class= Product
display Name =Research Information Basic Model C.doc
-----------------
```


Retrieve objects from a folder: Walkthrough

Procedures: Retrieve the subfolders from a folder

Procedures: Retrieve documents from a folder

Procedures: Retrieve the objects from a folder

Retrieve the subfolders from a folder

Use the same Java class that you created in the previous lesson.

Procedure 1: Write a method to retrieve the subfolders from a folder

1. Define the method using the following signature:

- Scope: `public`
- Name: `getSubFoldersEDU`
- Parameters:
  - `ObjectStore` object
  - `String` for the folder path

2. Call `Factory.Folder.fetchInstance(...)`.

- Parameters:
  - `Objectstore` object
  - Folder path that are passed into this method
  - `null`

3. Get the subfolders contained in this parent folder by calling the `folder.get_SubFolders()` method.
   a. Assign the return value to a variable of type `FolderSet`.
4. Call `folderSet.iterator()`.
   a. Assign the returned value to a variable of the type `Iterator`.
5. Write a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```
6. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into a `Folder` type and assign it to a variable.
7. Call `folder.get_FolderName()` to retrieve the name of the folder and assign the return value to a `String` variable.
8. In an if statement, check whether the given folder is a hidden container.

   ```
   if (folder.getProperties().getBooleanValue("IsHiddenContainer"))
   ```
   a. Display each folder name and the hiddenContainer status of the folder by calling `System.out.println(…)`.
      - Parameter: The `String` variable that has the folder name from step 7.
9. Close the `while` loop and the method.

Procedure 2: Call the method inside the main() method

1. Call the `getSubFoldersEDU(...)` method inside the `try` block of the main() method.

- Parameters:
  - `Objectstore` variable that you got in the previous lesson
  - `"/Products"` (path for the parent folder)

2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.
2. Verify that a list of the name of the subfolders contained in the parent folder is displayed in the Command Prompt window or in the console of Eclipse.
3. Verify that the “Luxury Models” folder is displayed as hidden.
4. Use Content Engine Enterprise Manager or Workplace to locate the subfolders contained in the parent folder.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the Object store: Development

List of sub folders under the 'Products' folder:
Basic Models
Deluxe Models
The “Luxury Models” folder is hidden.
Model 120
Model 200
Model 220
Model 300
Model 320
Model A
Model B
Model C
```

Retrieve documents from a folder

Use the same Java class that you created in the previous lesson.

Procedure 1: Write a method to retrieve documents from a folder

1. Define the method using the following signature:

- Scope: `public`
- Name: `getDocsInThisFolderEDU`
- Parameters:
  - `ObjectStore` object
  - `String` for the folder path

2. Call `Factory.Folder.fetchInstance(...)`.

- Parameters:
  - `Objectstore` object
  - Folder path that are passed into this method
  - `null`

   a. Assign the return value to a variable of type `Folder`.

3. Get the documents filed in a folder by calling `folder.get_ContainedDocuments()`.

   a. Assign the return value to a variable of type `DocumentSet`.

4. Call `documentSet.iterator()`.

   a. Assign the return value to a variable of type `Iterator`.

5. Write a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```

6. Call `iterator.next()`within the while loop.

   a. Typecast the return value into a `Document` type and assign it to a variable.

7. Display each document name by calling `System.out.println(…)`.

- Parameter: `document.get_Name()`

8. Close the method.

Procedure 2: Call the method inside the main() method

1. Call the `getDocsInThisFolderEDU(...)` method that you wrote inside the main() method.

- Parameters:
  - `Objectstore` variable that you got in the previous lesson
  - `"/Manuals"` (path for the parent folder)

2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.
2. Verify that a list of the name of the documents contained in the parent folder is displayed in the in the Command Prompt window or in the console of Eclipse.
3. Use Content Engine Enterprise Manager or Workplace to locate the documents filed in the parent folder.

Sample output

```
List of documents under the 'Manuals' folder:
User Manual for Basic Model B.pdf
User Manual for Basic Model B.doc
User Manual for Basic Model C.pdf
User Manual for Basic Model C.doc
User Manual for Basic Model A.doc
User Manual for Basic Model A.pdf
```

Retrieve the objects from a folder

Use the same Java class that you created in the previous lesson.

Procedure 1: Write a method to retrieve the objects from a folder

1. Define the method using the following signature:

- Scope: `public`
- Name: `getFolderContaineesEDU`
- Parameter: `ObjectStore` object, `String` for the folder path

2. Call `Factory.Folder.fetchInstance(...)`.

- Parameters:
  - `Objectstore` object
  - Folder path that are passed into this method
  - `null`

   a. Assign the return value to a variable of type `Folder`.

3. Get the objects filed in the folder by calling the `folder.get_Containees()` method.

   a. Assign the return value to a variable of type `ReferentialContainmentRelationshipSet`.

4. Retrieve individual `ReferentialContainmentRelationship` objects by writing a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```

5. Call `iterator.next()` within the `while` loop.

   a. Typecast the return value into a `ReferentialContainmentRelationship` type and assign it to a variable.

6. Call `referentialContainmentRelationship.get_Head()`.

   a. Assign the returned value to a variable of the type `IndependentObject`.

7. Call `independentObject.getClassName()` to get the class name.

   a. Assign the returned value to a variable of the type `String`.

8. Call `referentialContainmentRelationship.get_Name()` to get the display name.

   a. Assign the returned value to a variable of the type `String`.

9. Display each object class name and display name respectively by calling `System.out.println(…)`.

- Parameter:
  - String value from step 7 for the class name
  - String value from step 8 for the display name

Procedure 2: Call the method inside the main() method

1. Call the `getFolderContaineesEDU` method inside the main() method.

- Parameters:
  - `Objectstore` variable that you got in the previous lesson
  - `"/Research"` (path of the parent folder)

2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.
2. Verify that a list of the names of the objects contained in the parent folder is displayed in the in the Command Prompt window or in the console of Eclipse.
3. Use Content Engine Enterprise Manager or Workplace to locate the objects filed in the parent folder.

Sample output

```
   List of objects under the 'Research' folder:
   -----------------
   class= Product
   display Name =Research Information Basic Model A.doc
   -----------------
   class= Document
   display Name =JavaDoc
   -----------------
   class= CustomObject
   display Name =JavaCustObj
   -----------------
   class= Product
   display Name =Research Information Basic Model C.doc
   -----------------
   class= Product
   display Name =Research Information Basic Model B.doc
-----------------
```

Solution code for FoldersEDU.java

```
package com.ibm.filenet.edu;

import java.util.Iterator;
import com.filenet.api.core.*;
import com.filenet.api.constants.*;
import com.filenet.api.collection.*;

public class FoldersEDU extends CEConnectionEDU {
public Folder createFolderEDU(ObjectStore store,String folderName)
{
   Folder myFolder = Factory.Folder.createInstance(store, null);
   myFolder.set_FolderName(folderName);
   Folder rootFolder = store.get_RootFolder();
   myFolder.set_Parent(rootFolder);
   myFolder.save(RefreshMode.REFRESH);
   System.out.println(myFolder.get_Name() + " is created.");
   return myFolder;
}

public void createSubFolderEDU(Folder parentFolder, String folderName)
{
   Folder subFolder = parentFolder.createSubFolder(folderName);
   subFolder.save(RefreshMode.REFRESH);
   System.out.println(subFolder.get_Name() + " is created.");
}

public Folder getFolderEDU(ObjectStore store, String folderName)
{
   Folder folder= Factory.Folder.fetchInstance(store,folderName, null);
   folderName = folder.get_FolderName();
   System.out.println(folderName + "  folder is retrieved");
   return folder;
}

public void deleteFolderEDU(ObjectStore store, String folderToDelete)
{
  Folder folder= Factory.Folder.fetchInstance(store,folderToDelete, null);
  String folderName = folder.get_FolderName();
  folder.delete();
  folder.save(RefreshMode.REFRESH);
  System.out.println(folderName + "  is Deleted");
}
```



```
public void getSubFoldersEDU(ObjectStore store,String parentFolder){
try
{
   Folder folder= Factory.Folder.fetchInstance(store,parentFolder, null);
   FolderSet subFolders = folder.get_SubFolders();
   Iterator it = subFolders.iterator();
   System.out.println("List of sub folders under the 'Products' folder:");
   while(it.hasNext())
   {
      Folder retrieveFolder = (Folder)it.next();
      String name = retrieveFolder.get_FolderName();
      if (retrieveFolder.getProperties().
      getBooleanValue("IsHiddenContainer"))
         System.out.println ("Folder " + name + " is hidden");
      else
         System.out.println(name);
   }
}catch (Exception e) {
      e.printStackTrace();
}
}

public void getDocsInThisFolderEDU (ObjectStore store, String parentFolder
{
   try{
      Folder folder= Factory.Folder.fetchInstance(store,parentFolder, null);
      ocumentSet documents = folder.get_ContainedDocuments();
      Iterator it = documents.iterator();
      System.out.println("List of documents under the 'Manuals' folder:");
      while(it.hasNext()){
      Document retrieveDoc = (Document)it.next();
      String name = retrieveDoc.get_Name();
      System.out.println(name);
   }
   } catch (Exception e) {
      e.printStackTrace();
   }
}
```

```
public void getFolderContaineesEDU (ObjectStore store, String parentFolder){
try{
      Folder folder= Factory.Folder.fetchInstance(store,parentFolder, null);
      ReferentialContainmentRelationshipSet refConRelSet =
      folder.get_Containees();
      Iterator it = refConRelSet.iterator();
      System.out.println("List of objects under the 'Research' folder:");
      while(it.hasNext()){
         System.out.println("-----------------");
         ReferentialContainmentRelationship retrieveObj
         =(ReferentialContainmentRelationship)it.next();
         IndependentObject containee = retrieveObj.get_Head();
         String className = containee.getClassName();
         System.out.println("class= " + className);
         String displayName = retrieveObj.get_Name();
         System.out.println("display Name =" + displayName);
         }
  System.out.println("-----------------");
} catch (Exception e) {
      e.printStackTrace();
}
}

public static void main(String[] args) {
   try{
      FoldersEDU folderInstance = new FoldersEDU();
      Connection conn =
      folderInstance.getCEConnectionEDU("p8admin","IBMFileNetP8");
      Domain domain = folderInstance.getDomainEDU(conn);
      ObjectStore store = folderInstance.getObjectStoreEDU(domain,
      "Development");
      //Folder myFolder =
      folderInstance.createFolderEDU(store,"JavaFolder");
      //folderInstance.createSubFolderEDU(myFolder,"ASubFolder");
      //folderInstance.deleteFolderEDU(store,"/JavaFolder/ASubFolder");
      //folderInstance.getSubFoldersEDU(store,"/Products");
      //folderInstance.getDocsInThisFolderEDU(store,"/Manuals");
      folderInstance.getFolderContaineesEDU(store,"/Research");
   }catch (Exception e) {
      e.printStackTrace();
   }
}
}
```


Unit 4. Custom Objects

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class from the “Communication with the Content Engine Server” unit is used in this unit for the Content Engine Connection.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt window.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt window.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference.

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 4.1. Create custom objects

Overview

Why is this lesson important to you?

Morgan registers online to be a customer of a company. The application uses a custom object class to store the profile of the customers. As a programmer, you are going to write code to create an instance of the class each time a customer enters this information.

Activities

- Write code to create a custom object.
- Write code to set the properties to the custom object.
- Write code to file the custom object to a folder.

Skill levels

Create custom objects: Challenge, page 4-7

Create custom objects: Walkthrough, page 4-9

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Create custom objects: Challenge

Challenge

Create a Java class that extends CEConnectionEDU. Use the Data 1 and Data 2 tables to write code to create a custom object, set the properties, and then file it in a folder.

Data 1

Item	Value

Packages to import	`java.util.*`
	`java.text.*`
	`com.filenet.api.constants.*`
	`com.filenet.api.collection.*`
	`com.filenet.api.core.*`
	`com.filenet.api.exception.*`
	`com.filenet.api.property.*`

Object store name	`"Development"`
CustomObject class	`"Customer"`
Containment name for the custom object	`"APICustomObj"`
Folder path to file the custom object	`"/APIFolder"`

Data 2

Custom object property name	Value
“customer_name”	`"John Smith"`
“customer_id”	`CUS29`
“age”	29
"phone_numbers"	`"000-000-0000"` `“333-444-5555”`
"member_date"	`"20/12/2005"`

Verification

- Run the program using Eclipse or the Command Prompt.
- Use Workplace to verify that the custom object is created with the properties that you specified and filed in the `APIFolder` folder.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the object store: Development
APIFolder  folder is retrieved
Custom object John Smith is created
```


Create custom objects: Walkthrough

Procedures

Procedure 1: Create a Java class that extends CEConnectionEDU

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the “New Java Class” window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type the class name `CustomObjectsEDU` in the Name field.
5. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
6. Select `public static void main(String[] args)` for “Which method stubs would you like to create?” option.
   a. Clear other options.
7. Click Finish to add the class and close the window.
8. Verify the new class listed under your package in the Project Explorer.
9. Write code inside the class that you just added.
10. Import the following packages before the declaration of the class:

```
java.util.*
java.text.*
com.filenet.api.constants.*
com.filenet.api.collection.*
com.filenet.api.core.*
com.filenet.api.exception.*
com.filenet.api.property.*
```

11. Inside the main method, write a `try-catch` block.
    a. Call `exception.printStackTrace()` in the catch block to display exceptions.
    b. Add a custom methods and call it from the main method as described in the following procedures.

Procedure 2: Write a method to create a custom object

1. Define the method using the following signature:

- Scope: `public`
- Name: `createCustomObjectEDU`
- Parameters:
  - `ObjectStore` object
  - `Folder` object
  - `String` for the containment name

2. Call `Factory.CustomObject.createInstance(...)`.

- Parameters:
  - `ObjectStore` object
  - `"Customer"`

    a. Assign the return value to a variable of type `CustomObject` object.

Procedure 3: Set the properties to the custom object

Continue with the same method that you wrote.

1. Retrieve the properties from `CustomObject.getProperties()`.
    a. Assign the return value to a variable of type `com.filenet.api.property.Properties`.
2. Call `properties.putValue(...)`.

- Parameters:
  - `"customer_name"`
  - `"John Smith"`

3. Repeat calling `properties.putValue(...)`.

- Parameters:
  - `"customer_id"`
  - `CUS29`

4. Repeat calling `properties.putValue(...)`.

- Parameters:
  - "age"
  - 29

5. Call `Factory.StringList.createList()`.
   a. Assign the return value to a variable of type `StringList` object.
   b. Call `stringList.add(...)` to add a phone number value.
      - Parameter: `"000-000-0000"`
   c. Call `stringList.add(...)` to add a second phone number value.
      - Parameter: `"333-444-5555"`
   d. Repeat calling `properties.putValue(...)`.
      - Parameters
      - `“phone_numbers”`
      - `StringList` object from step 5
6. Create a new instance of `DateFormat` and assign it to a variable.

   ```
   DateFormat date = new SimpleDateFormat("dd/MM/yyyy");
   ```

   a. Call `dateFormat.parse(...)`to convert a date value to the desired format.
      - Parameter: `“20/12/2005”`
   b. Assign the return value to a variable of type `DateFormat`.
   c. Repeat calling `properties.putValue(...)`.
      - Parameters:
      - `“member_date”`
      - `DateFormat` object that you got in step 6b
7. Save the changes to the object by calling `customObject.save(...)` method.

- Parameter: `RefreshMode.REFRESH`

Procedure 4: File the custom object into a Folder

1. Call `folder.file(...)`.

- Parameters:
  - CustomObject object that you created in the previous activity
  - `AutoUniqueName.AUTO_UNIQUE`
  - String value for the containment name that is passed into this method
  - `DefineSecurityParentage.DO_NOT_DEFINE_SECURITY_PARENTAGE`

  a. Assign the return value to a variable of type `ReferentialContainmentRelationship`.

2. Call `referentialContainmentRelationship.save(...)`.

- Parameter: `RefreshMode.REFRESH`

Procedure 5: Instantiate the class and call the methods

1. Inside the `try` block of the main() method, create an instance of the class.

   Example: `CustomObjectsEDU CustomObjInstance = new CustomObjectsEDU();`

2. Call the `getCEConnectionEDU(...)` method of the `CEConnectionEDU` class:

- Parameters:
  - `"p8admin"`
  - `"IBMFileNetP8"`

   a. Assign the returned `Connection` object to a variable.

3. Call the `getDomainEDU(…)`method of the `CEConnectionEDU` class.

- Parameter: `Connection` object variable from step 2

   a. Assign the returned `Domain` object to a variable.

4. Call the `getObjectStoreEDU(…)` method of the `CEConnectionEDU` class:

- Parameters:
  - `Domain` object variable from step 3
  - `"Development"`

   a. Assign the returned `ObjectStore` object to a variable.

5. Call the `getFolderEDU(…)` method of the `CEConnectionEDU` class:

- Parameters:
  - `ObjectStore` object from step 4
  - `"/APIFolder"` (name of the folder to file the custom object)

   a. Assign the returned `Folder` object to a variable.

6. Call the `createCustomObjectEDU(...)` method.

- Parameters:
  - `Objectstore` object from step 4
  - `Folder` object from step 5
  - `"APICustomObj"` for the containment name

Procedure 6: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt as instructed in the "Communication with the Content Engine Server" unit.
2. Use Workplace to verify that the custom object is created with the properties that you specified and filed in the APIFolder folder of the Development object store.

   a. You see a custom object with the customer name that you specified.

Sample output

Refer to the sample output in Create custom objects: Challenge, page 4-7

Lesson 4.2. Retrieve custom object properties

Overview

Why is this lesson important to you?

The application has a custom object class that stores customer profiles. As the programmer, you are going to write code to retrieve custom object properties in order to modify the profile each time a customer updates this information.

Activities

- Write code to custom object properties.

Skill levels

Retrieve custom object properties: Challenge, page 4-15
Retrieve custom object properties: Walkthrough, page 4-17

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Retrieve custom object properties: Challenge

Challenge

Use the same Java class that you created in the previous lesson and the data in the following table to write code to retrieve custom object properties and display the values.

Data

Item	Value
Object store name	`"Development"`
Containment name and folder path of the custom object that you created in the previous lesson	`"/APIFolder/APICustomObj"`

Properties to retrieve	`"customer_name"`
	`"customer_id”`
	`"age"`
	`"member_date"`
	`"phone_numbers"`
	`PropertyNames.DATE_CREATED`

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that a list of the names of the properties that you specified and their values is displayed in the Command prompt window or in the console of Eclipse.

Sample output

```
Name of the object store: Development
APIFolder  folder is retrieved
CustomObject John Smith is retrieved
Customer Name:  John Smith
Customer age: 29
Customer ID: CUS29
Date created: Fri Mar 25 11:26:38 PDT 2011
Customer phone numbers:
   000-000-0000
   333-444-5555
Member since : Tue Dec 20 00:00:00 PST 2005
```


Retrieve custom object properties: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to retrieve a custom object

1. Define the method using the following signature:

- Scope: `public`
- Name: `getCustomObjectEDU`
- Parameters:
  - `ObjectStore` object
  - `String` for the folder path with the containment name of the custom object

2. Call `Factory.CustomObject.fetchInstance(...)`.

- Parameters:
  - `Objectstore` object
  - Folder path that is passed into this method
  - `null`

   a. Assign the return value to a variable of type `CustomObject`.

3. Retrieve and display the custom object name by calling `System.out.println(…)`.

- Parameter: `customObject.get_Name()`

4. Retrieve the properties by calling `customObject.getProperties()`.

   a. Assign the return value to a variable of type `com.filenet.api.property.Properties`.

5. Call `properties.getStringValue(...)`.

- Parameter: `"customer_name"`

   a. Assign the return value to a variable of type `String`.

6. Call `properties.getStringValue(...)`.

- Parameter: `"customer_id"`

   a. Assign the return value to a variable of type `String`.

7. Call `properties.getInteger32Value(...)`.

- Parameter: `"age"`

a. Assign the return value to a variable of type `System.Integer`.

8. Call `properties.getDateTimeValue(...)`.

- Parameter: `"member_date"`

a. Assign the return value to a variable of type `Date`.

9. Call `Properties.getStringListValue(...)`.

- Parameter: `"phone_numbers"`

a. Assign the return value to a variable of type `StringList`.

10. Call `stringList.iterator()`and assign it to an `Iterator` variable.
11. Write a `while` loop and iterate through the collection to retrieve the individual phone number.

```
while (iterator.hasNext())
```

12. Call `iterator.next()`within the while loop.
    a. Typecast the return value into a `String` type and assign it to a variable.
    b. Display the values by calling `System.out.println(…)`.
13. Call `properties.getDateTimeValue(...)`.

- Parameter: `PropertyNames.DATE_CREATED`

a. Assign the return value to a variable of type `Date`.

14. Display the values by calling `System.out.println(...)`.

Procedure 2: Call the method inside the main() method

1. Call the `getCustomObjectEDU(...)` method inside the `try` block of the main() method.

- Parameters:
  - `Objectstore` variable that you got in the previous lesson
  - `"/APIFolder/APICustomObj"` (folder path for the custom object)

2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.
2. Verify that the name and properties of the custom object are displayed in the Command Prompt window or in the console of Eclipse.

Sample output

Refer to the sample output in Retrieve custom object properties: Challenge, page 4-15

Solution code for CustomObjectsEDU.java

```
package com.ibm.filenet.edu;

import java.util.*;
import java.text.*;
import com.filenet.api.constants.*;
import com.filenet.api.collection.*;
import com.filenet.api.core.*;
import com.filenet.api.exception.*;
import com.filenet.api.property.*;
public class CustomObjectsEDU extends CEConnectionEDU {

public void createCustomObjectEDU(ObjectStore os, Folder folder, String
containmentName) throws Exception{
try{
      CustomObject myObject = Factory.CustomObject.createInstance
      (os,"Customer");
      com.filenet.api.property.Properties props = myObject.getProperties()
      props.putValue("customer_name", "John Smith");
      props.putValue("customer_id", "CUS29");
      props.putValue("age", 29);
      StringList phones = Factory.StringList.createList();
      phones.add("000-000-0000");
      phones.add("333-444-5555");
      props.putValue("phone_numbers", phones);
      DateFormat date = new SimpleDateFormat("dd/MM/yyyy");
      Date memeberDate = date.parse("20/12/2005");
      props.putValue("member_date", memeberDate);
      myObject.save(RefreshMode.REFRESH):
      System.out.println("Custom object " + myObject.get_Name() + " is
      created");
      ReferentialContainmentRelationship rel =  folder.file(myObject,
      AutoUniqueName.AUTO_UNIQUE, containmentName,
      DefineSecurityParentage.DO_NOT_DEFINE_SECURITY_PARENTAGE);
      rel.save(RefreshMode.REFRESH);
      //return myObject;
} catch (Exception e) {
      e.printStackTrace();
}
}

public void getCustomObjectEDU(ObjectStore store, String folderPath){
try{
```

```
      CustomObject myObject = Factory.CustomObject.fetchInstance(
      store,folderPath,null);
      String CustomObjName =  myObject.get_Name();
      System.out.println(CustomObjName + "  CustomObject is retrieved");
      com.filenet.api.property.Properties props = myObject.getProperties();
      String name = props.getStringValue("customer_name");
      String custID = props.getStringValue("customer_id");
      Integer age = props.getInteger32Value("age");
      Date dateCreated = props.getDateTimeValue(PropertyNames.DATE_CREATED);
      Date memeberDate = props.getDateTimeValue("member_date");
      System.out.println("Customer Name:  " + name );
      System.out.println("Customer age: " + age);
      System.out.println("Customer ID: " + custID);
      System.out.println("Date created: " + dateCreated);
      StringList phoneNumbers = props.getStringListValue("phone_numbers");
      Iterator it = phoneNumbers.iterator();
      System.out.println("Customer phone numbers: ");
      while (it.hasNext()){
         String phoneNumber = (String)it.next();
         System.out.println("   " + phoneNumber);
      }
      System.out.println("Member since : " +  memeberDate);
      }
      catch (Exception e) {
         e.printStackTrace();
      }
}
public static void main(String[] args) {
   try{
         CustomObjectsEDU myInstance = new CustomObjectsEDU();
         Connection conn =
         myInstance.getCEConnectionEDU("p8admin","IBMFileNetP8");
         Domain domain = myInstance.getDomainEDU(conn);
         ObjectStore store = myInstance.getObjectStoreEDU(domain,
         "Development");
         Folder folder = myInstance. getFolderEDU(store, "/APIFolder");
         myInstance.createCustomObjectEDU(store,folder, "APICustomObj");
         myInstance.getCustomObjectEDU(store, "/APIFolder/APICustomObj");
      } catch (Exception e) {
      e.printStackTrace();
      }
 }
}
```


Unit 5. Documents

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class in this unit extends the CEConnectionEDU class from the “Communication with the Content Engine” unit.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. (These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.)

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 5.1. Create Document objects

Overview

Why is this lesson important to you?

Your application requires a feature that creates a Document object on the Content Engine whenever someone adds content. As the programmer, you are going to write code to accomplish this.

Activities

- Write code to create a Document object.
- Write code to set content to the document and check in.
- Write code to set the properties to the document and save.
- Write code to file the document to a folder.

Skill levels

Create Document objects: Challenge, page 5-7
Create Document objects: Walkthrough, page 5-9

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Create Document objects: Challenge

Challenge

Use the data in the table to complete the following activities.

- Write a Java class that extends CEConnectionEDU and write code to create a Document object, check in the object, set properties to it, and file it in a folder.
- Optionally, write code to set the following:
  - Multiple content elements to a Document object
  - External content element to a Document object

Data

Item	Value

Packages to import	`java.io.*`
	`java.util.Iterator`
	`com.filenet.api.collection.*`
	`com.filenet.api.constants.*`
	`com.filenet.api.core.*`
	`com.filenet.api.property.*`

Object store name	`"Development"`
Document class name	`"Product"`
File name with the folder path for the content	`"C:\\CMJavaAPIProg\\SampleDocuments\\Model 200.GIF"`
Folder path to file the document	`"/APIFolder"`
Mime type	`"image/gif"`
Name of the additional file with the folder path for the multiple content	`"C:\\CMJavaAPIProg\\SampleDocuments\\Model 400.JPG"`
URL for the external content	`"http://ccvo1135:9080/FileNet/Engine"`

Document Property	Value
Document Title	`"New Model"`
model_code	`"AST765"`
product_id	`"PDT101"`
Document Title (for the multi-content document)	`"ModelMCDoc"`
Document Title (for the external document)	`"ModelExternalDoc"`

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that the Document object is created with the document title that you provided and with the properties that you set and filed in the APIFolder using Workplace.

Create Document objects: Walkthrough

Procedures

Procedure 1: Create a Java class that extends CEConnectionEDU

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the “New Java Class” window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type the class name `DocumentsEDU` in the Name field.
5. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
6. Select `public static void main(String[] args)` for “Which method stubs would you like to create?” option.
   a. Clear other options.
7. Click Finish to add the class and close the window.
8. Verify the new class listed under your package in the Project Explorer.
9. Write code inside the class that you just added.

10. Import the following packages before the declaration of the class:

```
java.io.*
java.util.Iterator
com.filenet.api.collection.*
com.filenet.api.constants.*
com.filenet.api.core.*
com.filenet.api.property.*
```

11. Inside the main method, write a `try-catch` block.
    a. Call `exception.printStackTrace()` in the catch block to display the exceptions.
    b. Add a custom method and call it from the main() method as detailed in the following steps.

Procedure 2: Write a method to create a document

1. Define the method using the following signature:

- Scope: `public`
- Name: `createDocumentEDU`
- Parameters:
  - `ObjectStore` object
  - `Folder` object

2. Call `Factory.Document.createInstance(...)`.

- Parameters:
  - `ObjectStore` object
  - `"Product"`

    a. Assign the return value to a variable of type `Document`.

Procedure 3: Set content to the Document and check in

1. Call `Factory.ContentElement.createList()`.
    a. Assign the return value to a variable of type `ContentElementList`.
2. Call `Factory.ContentTransfer.createInstance()`.
    a. Assign the return value to a variable of type `ContentTransfer`.
3. Create a new instance of `FileInputStream` using the constructor.
    a. Parameter: `"Model 200.GIF"`

Note

If you have not added this file in Eclipse project, then use the full folder path:

```
"C:\\CMJavaAPIProg\\Source\\Model 200.GIF"
```


   b. Assign the return value to a variable of type `FileInputStream`.

4. Call `ContentTransfer.setCaptureSource(...)`on the object from step 2
   - Parameter: `FileInputStream` object from the step 3
5. Call `ContentElementList.add(...)`.
   - Parameter: `ContentTransfer` object from the step 2 and 3
6. Call `document.set_ContentElements(...)`.
   - Parameter: `ContentElementList object` from the step 1 and 5
7. Call `document.checkin(...)`.
   - Parameters:
     - `AutoClassify.DO_NOT_AUTO_CLASSIFY`
     - `CheckinType.MAJOR_VERSION`

Procedure 4: Set the properties and save the Document

1. Call `document.getProperties()`.
   a. Assign the return value to a variable of type `com.filenet.api.property.Properties`.
2. Call `properties.putValue(...)` to set value to the `DocumentTitle` property.

- Parameters:
  - "DocumentTitle"
  - "New Model"

3. Repeat calling `properties.putValue(...)` to set value to the `product_id` property.

- Parameters:
  - "product_id"
  - "PDT101"

4. Repeat calling `properties.putValue(...)` to set value to the `model_code` property.

- Parameters:
  - "model_code"
  - "AST765"

5. Call `document.set_MimeType()` to set the `MimeType`.

- Parameter: "image/jpeg"

6. Save the changes to the object by calling `document.save(...)` method.

- Parameter: `RefreshMode.REFRESH`

Procedure 5: File the Document into a Folder

1. Call `folder.file(...)`.

- Parameters:
  - `Document` object that you created in the previous activity
  - `AutoUniqueName.AUTO_UNIQUE`
  - `"NewModel"` for the containment name
  - `DefineSecurityParentage.DO_NOT_DEFINE_SECURITY_PARENTAGE`

   a. Assign the return value to a variable of type `ReferentialContainmentRelationship`.

2. Call `referentialContainmentRelationship.save(...)`.

- Parameter: `RefreshMode.NO_REFRESH`

Procedure 6: Instantiate the class and call the methods that you wrote

1. Inside the `try` block of the main() method, create an instance of the class and assign the return value to a variable.

   Example: `DocumentsEDU documentsInstance = new DocumentsEDU();`

2. Call the `getCEConnectionEDU(...)` method of the CEConnectionEDU class:

- Parameters:
  `"p8admin"`
  `"IBMFileNetP8"`

   a. Assign the returned `Connection` object to a variable.

3. Call the `getDomainEDU(…)`method of the CEConnectionEDU class.

- Parameter: `Connection` object from step 2

   a. Assign the returned `Domain` object to a variable.

4. Call the `getObjectStoreEDU(…)` method of the `CEConnectionEDU` class:

- Parameters:
  - `Domain` object from step 3
  - Object store name from the following data table

   a. Assign the returned `ObjectStore` object to a variable.

Item	Value
Object store name	`"Development"`
Folder path to file the document	`“/APIFolder”`

5. Call the `getFolderEDU(...)` method of the `CEConnectionEDU` class.

- Parameters:
  - `ObjectStore` object from step 4
  - Folder path to file the document from the preceding data table

  a. Assign the return value to a variable of type `Folder`.

6. Call the `createDocumentEDU(...)` method.

- Parameters:
  - `Objectstore` object from step 4
  - `Folder` object from step 5

Procedure 7: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt as instructed in the "Communication with the Content Engine Server" unit.
2. Verify, using the Workplace, that the document with title "New Model" is created under the APIFolder folder of the Development object store.

Procedure 8: Optional - Create a Document with multiple content elements

1. Define the method using the following signature:

- Scope: `public`
- Name: `createMultipleContentsDocumentEDU`
- Parameters:
  - `Objectstore` object
  - `Folder` object

2. Reuse the code that you wrote in procedure 2 to create an instance of a `Document`.
3. Reuse the code that you wrote in procedure 3 to set content to the Document object.
4. Modify the code to add additional content elements to the ContentElementList as described in the following steps.

   a. Call `Factory.ContentTransfer.createInstance()` .

   b. Assign the return value to a variable of type `ContentTransfer`.

   c. Create a new instance of `FileInputStream` using the constructor.

   - Parameter: `"Model 200.GIF"`

Note

If you have not added this file in Eclipse project, then use the full folder path:

```
":\\CMJavaAPIProg\\SampleDocuments\\Model 200.GIF"
```

   d. Assign the return value to a variable of type `FileInputStream`.
   e. Call `ContentTransfer.setCaptureSource(...)`on the object from step 2.
      - Parameter: `FileInputStream` object from step 4d
   f. Call `contentTransfer.set_ContentType(…)`.
      - Parameter: `"image/jpeg"`
   g. Call `ContentElementList.add(...)`.
      - Parameter: `ContentTransfer` object from the previous step.
5. Repeat step 4 to add the second content element. By using the file `"Model 400.JPG"` or with complete path `"C:\\CMJavaAPIProg\\Source\\Model 400.JPG".`
   a. Call `document.set_ContentElements(…)`.
      - Parameter: `ContentElementList` object that you created
6. Reuse the code that you wrote in procedure 3 to check in the Document object.
7. Reuse the code that you wrote in procedure 4 to set the properties and save the Document object with the following modification:
   a. Use a different document title for this new document.

      Example: `"ModelMCDoc"`
8. Reuse the code that you wrote in procedure 5 to file the Document object into a folder with a different containment name, Example: `"ModelMCDoc"`.
9. Run the program and verify that the document with title `ModelMCDoc` is created under the `APIFolder` folder of the `Development` object store:
   a. If you are viewing the document using Image Viewer in Workplace, you see two pages: one page each for each content element.
   b. If you are using Content Engine Enterprise Manager, the Content Elements tab of the Document Properties window has two entries: one entry each for each content element.

Procedure 9: Optional - Create a Document object with external content

1. Use the same Java class that you created in the previous activity.
2. Define the method using the following signature:

- Scope: `public`
- Name: `createContentReferenceDocumentEDU`
- Parameters:
  - `Objectstore` object
  - `Folder` object

3. Reuse the code that you wrote in procedures 2 and 3 to create an instance of a `Document` object and `ContentElementList` object.

4. Call `Factory.ContentReference.createInstance()`.
   a. Assign the return value to a variable of type `ContentReference`.
5. Call `ContentReference.set_ContentLocation(...)`.

- Parameter: `"http://ccvo1135:9080/FileNet/Engine"`

6. Set the content type by calling `contentReference.set_ContentType(...)` property.

- Parameter: `"text/html"`

7. Call `ContentElementList.add(...)`.

- Parameter: `ContentReference` object from the previous step

8. Call `document.set_ContentElements(...)` property using the `ContentElementList` that you created in the previous steps.
9. Reuse the code that you wrote in procedure 3 to check in the Document object.
10. Reuse the code that you wrote in procedure 4 to set the properties and save the Document object with the following modification:
    a. Use a different document title for this new document.

       Example: `“ModelExternalDoc”`

11. Reuse the code that you wrote in procedure 5 to file the Document object into a folder with a different containment name.

    Example: `"ModelExternalDoc"`

12. Run the program and verify that the Document object with the title `ModelExternalDoc` is created in the `APIFolder` folder of the `Development` object store:
    a. If you are viewing the document using the Workplace, click the document to open the external link for the document.
    b. If you are using the Content Engine Enterprise Manager, the Content Elements tab of the Document Properties window has an entry for the external location of the document.

Lesson 5.2. Retrieve a document and its content

Overview

Why is this lesson important to you?

Your company employees need to access document objects to review their content. As their programmer, you are going to write code to retrieve a document object and its content when an employee accesses the document.

Activities

- Write code to retrieve a document.
- Write code to retrieve document content.

Skill levels

Retrieve a document and its content: Challenge, page 5-19
Retrieve a document and its content: Walkthrough, page 5-21

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Retrieve a document and its content: Challenge

Challenge

Use the same Java class that you created in the previous lesson.

- Write code to retrieve a document from the object store and display its name.
- Write code to retrieve a content stream for a document and output it to a file.

Data

Item	Value
Folder path for the document	`"/Research/Research Information Basic Model A.doc"`

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that the name of the document that you specified and its content information are displayed in the Command Prompt window or the console of Eclipse.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the object store: Development
Research Information Basic Model A.doc  Document is retrieved
content type = application/msword
fileName = Research Information Basic Model A.doc
content Size = 24064.0
```


Retrieve a document and its content: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures: Retrieve a document

Procedures: Retrieve the content of a document

Retrieve a document

Procedure 1: Write a method to retrieve a document

1. Define the method using the following signature:

- Scope: `public`
- Name: `getDocumentEDU`
- Parameters:
  - `ObjectStore` object
  - `String` for the folder path
- Returns: `Document` object

2. Call `Factory.Document.fetchInstance(...)`.

- Parameters:
  - `Objectstore` object
  - Folder path that is passed into this method
  - `null`

   a. Assign the return value to a variable of type `Document`.

3. Get and display the document name by calling `System.out.println(…)`.

- Parameter: `document.get_Name()`

4. Return the `Document` object from step 2.

Procedure 2: Call the method inside the main() method

1. Call the `getDocumentEDU(...)`method inside the `try` block of the main() method.

- Parameters:
  - `Objectstore` object that you got in the previous activity.
  - `"/Research/Research Information Basic Model A.doc"`

2. Comment out the methods that are not tested in this activity.
3. Save the file.

Procedure 3: Run the program and verify the results

1. Run the program using Eclipse or Command Prompt.
2. Verify that the name of the document that you specified is displayed in the Command Prompt window or the console of Eclipse.

Sample output

```
Research Information Basic Model A.doc  Document is retrieved
```


Retrieve the content of a document

Use the same Java class that you created in the previous lesson.

Procedure 1: Write a method to retrieve the content of a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getDocumentContentEDU`
   - Parameter: `Document` object
2. Call `document.get_ContentElements()`.
   a. Assign the return value to a variable of type `ContentElementList`.
3. Get the iterator by calling `contentElementList.iterator()`and assign it to an `Iterator` variable.
4. Write a `while` loop and iterate through the collection:

   ```
   while (iterator.hasNext())
   ```
5. Call `iterator.next()`within the while loop.
   a. Typecast the return value into a `ContentElement` type and assign it to a variable.
   b. Display each content type by calling `System.out.println(…)`.
      - Parameter: call `contentElement.get_ContentType()` on the object from step 5a.
6. Write a method called `getContentStreamEDU(…)` (see procedure 2) and call that method here.
   - Parameter: `ContentTransfer` (Typecast the `ContentElement` object from step 5a.)
7. Close the method.

Procedure 2: Write a method to retrieve the content stream of a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getContentStreamEDU`
   - Parameter: `ContentTransfer` object
2. Call `contentTransfer .get_RetrievalName()` on the object that is passed into this method to get the file name.
   a. Assign the return value to a variable of type `String`.
3. Call `contentTransfer .get_ContentSize()`to get the content size.
   a. Assign the return value to a variable of type `Double`.
   b. Display the file name and content size by calling `System.out.println(…)`.

Note

The solution contains a method with the following signature for writing the content stream into a file (these method calls do not use IBM FileNet P8 Content Engine Java API):

```
public void getContentStreamEDU(ContentTransfer content) throws Exception
```

Optionally, copy the lines of code of this method to write the stream into a file.

Procedure 3: Call the method inside the main() method

1. Call the `getDocumentContentEDU(...)` method that you wrote inside the main() method.

- Parameters:
  - `Document` variable that you got in the previous activity

2. Comment out the method calls that are not tested in this activity.

Procedure 4: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.
2. Verify that the name of the document that you specified and its content information are displayed in the in the Command Prompt window or the console of Eclipse.

   If you have copied and used the code from solution for the content stream, you also see that the file is written as a Word document with the file name `Research Information Basic Model A.doc` in the same folder where you run the code. (Your project folder for the Eclipse)

Sample output

```
Name of the objectstore: Development
Research Information Basic Model A.doc  Document is retrieved
content type = application/msword
fileName = Research Information Basic Model A.doc
content Size = 24064.0
```


Solution code for DocumentsEDU.java

```
package com.ibm.filenet.edu;

import java.io.*;
import java.util.Iterator;
import com.filenet.api.collection.*;
import com.filenet.api.constants.*;
import com.filenet.api.core.*;
import com.filenet.api.property.*;

public class DocumentsEDU extends CEConnectionEDU {
public void createDocumentEDU(ObjectStore store, Folder folder) throws
Exception{
try{
      Document myDoc = Factory.Document.createInstance(store,"Product");
      ContentElementList contentList = Factory.ContentElement.createList();
      ContentTransfer content = Factory.ContentTransfer.createInstance();
      // FileInputStream file = new
      FileInputStream("C:\\CMJavaAPIProg\\Source\\Model 200.GIF");
      FileInputStream file = new FileInputStream("Model 200.GIF");
      content.setCaptureSource(file);
      contentList.add(content);
      myDoc.set_ContentElements(contentList);
      myDoc.checkin(AutoClassify.DO_NOT_AUTO_CLASSIFY,
      CheckinType.MAJOR_VERSION);
      com.filenet.api.property.Properties properties =
      myDoc.getProperties();
      properties.putValue("DocumentTitle", "New Model");
      properties.putValue("product_id","PDT101");
      properties.putValue("model_code","AST765");
      myDoc.set_MimeType("image/jpeg");
      myDoc.save(RefreshMode.REFRESH);
      ReferentialContainmentRelationship rel = folder.file(myDoc,
      AutoUniqueName.AUTO_UNIQUE, "NewModel",
      DefineSecurityParentage.DO_NOT_DEFINE_SECURITY_PARENTAGE);
      rel.save(RefreshMode.NO_REFRESH);
      System.out.println("Document is created");
}
   catch (Exception e) {
      e.printStackTrace();
   }
}
```

```
public void createMultipleContentsDocumentEDU(ObjectStore store, Folder
folder) throws Exception{
try{
      Document myDoc = Factory.Document.createInstance(store,"Product");
      ContentElementList contentList = Factory.ContentElement.createList();
      ContentTransfer content1 = Factory.ContentTransfer.createInstance();
      // FileInputStream file = new
      FileInputStream("C:\\CMJavaAPIProg\\Source\\Model 200.GIF");
      FileInputStream file = new FileInputStream("Model 200.GIF");
      content1.setCaptureSource(file);
      content1.set_ContentType("image/jpeg");
      contentList.add(content1);
      ContentTransfer content2 = Factory.ContentTransfer.createInstance();
      content2.setCaptureSource(new FileInputStream("Model 400.JPG"));
      content2.set_ContentType("image/jpeg");
      contentList.add(content2);
      myDoc.set_ContentElements(contentList);
      myDoc.checkin(AutoClassify.DO_NOT_AUTO_CLASSIFY,
      CheckinType.MAJOR_VERSION);
      Properties properties = myDoc.getProperties();
      properties.putValue("DocumentTitle", "ModelMCDoc");
      properties.putValue("product_id","PDT101");
      properties.putValue("model_code","AST765");
      myDoc.set_MimeType("image/jpeg");
      myDoc.save(RefreshMode.REFRESH);
      ReferentialContainmentRelationship rel = folder.file(myDoc,
      AutoUniqueName.AUTO_UNIQUE, "ModelMCDoc",
      DefineSecurityParentage.DO_NOT_DEFINE_SECURITY_PARENTAGE);
      rel.save(RefreshMode.NO_REFRESH);
      System.out.println("document with multiple content is added");
}
catch (Exception e)
{
    e.printStackTrace();
}
}

public void createContentReferenceDocumentEDU(ObjectStore store, Folder
folder) throws Exception{
try {

      Document myDoc = Factory.Document.createInstance(store,"Document");
      ContentElementList contentList = Factory.ContentElement.createList();
      ContentReference contentRef =
```



```
      Factory.ContentReference.createInstance();
      contentRef.set_ContentLocation
      ("http://ccv01135:9080/FileNet/Engine");
      contentRef.set_ContentType("text/html");
      contentList.add(contentRef);
      myDoc.set_ContentElements(contentList);
      myDoc.checkin(AutoClassify.DO_NOT_AUTO_CLASSIFY,
      CheckinType.MAJOR_VERSION);
      Properties properties = myDoc.getProperties();
      properties.putValue("DocumentTitle", "ModelExternalDoc");
      myDoc.set_MimeType("text/html");
      myDoc.save(RefreshMode.REFRESH);
      ReferentialContainmentRelationship rel = folder.file(myDoc,
      AutoUniqueName.AUTO_UNIQUE,"ModelExternalDoc",DefineSecurityParentage
      .DO_NOT_DEFINE_SECURITY_PARENTAGE);
      rel.save(RefreshMode.NO_REFRESH);
      System.out.println("Document external content is added");
}
catch (Exception e) {
      e.printStackTrace();
}
}

public Folder getFolderEDU(ObjectStore store, String folderpath){
  Folder folder= Factory.Folder.fetchInstance(store,folderpath, null);
  String folderName = folder.get_FolderName();
  System.out.println(folderName + "  folder is retrieved");
  return folder;
}

public Document getDocumentEDU(ObjectStore store){
  Document doc = Factory.Document.fetchInstance(store,"/Research/Research
Information Basic Model A.doc", null);
  String documentName = doc.get_Name();
  System.out.println(documentName  + "  Document is retrieved");
  return doc;
}

public void getDocumentContentEDU(Document document) throws Exception
{
ContentElementList contents = document.get_ContentElements();
ContentElement content;
Iterator itContent = contents.iterator();
while (itContent.hasNext())
```

```
{
      content = (ContentElement)itContent.next();
      System.out.println("content type = " + content.get_ContentType());
      getContentStreamEDU((ContentTransfer)content);
}
}

public void getContentStreamEDU(ContentTransfer content) throws Exception
{
      String fileName = content.get_RetrievalName();
      Double contentSize = content.get_ContentSize();
      System.out.println("fileName = " + fileName);
      System.out.println("content Size = " + contentSize);
      InputStream inputStream = content.accessContentStream();
      OutputStream outputStream = new FileOutputStream(fileName);

    byte[] nextBytes = new byte[64000];
    int nBytesRead;
    while ((nBytesRead = inputStream.read(nextBytes)) != -1)
    {
      outputStream.write(nextBytes, 0, nBytesRead);
      outputStream.flush();
    }
}

  public static void main(String[] args) {
   try{
      DocumentsEDU documentsInstance = new DocumentsEDU();
      Connection conn =
      documentsInstance.getCEConnectionEDU("p8admin","IBMFileNetP8");
      Domain domain = documentsInstance.getDomainEDU(conn);
      ObjectStore store =
      documentsInstance.getObjectStoreEDU(domain,"Development");
      Folder folder = documentsInstance.getFolderEDU(store, "/APIFolder");
      //documentsInstance.createDocumentEDU(store,folder);
      //documentsInstance.createMultipleContentsDocumentEDU(store,folder);
      documentsInstance.createContentReferenceDocumentEDU(store, folder);
      //Document document = documentsInstance.getDocumentEDU(store);
      //documentsInstance.getDocumentContentEDU(document);
     } catch (Exception e) {
         e.printStackTrace();
       }
 }
}
```


Unit 6. Properties

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class in this unit extends the CEConnectionEDU class from the “Communication with the Content Engine” unit.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference.

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 6.1. Retrieve property descriptions

Overview

Why is this lesson important to you?

To manage the properties of the Content Engine objects, you need to retrieve the property descriptions. As a programmer, you are going to write code to retrieve and display the property descriptions of a given Document class.

Activities

- Write code to retrieve class descriptions.
- Write code to retrieve a document using property filter.
- Write code to retrieve property descriptions.

Skill levels

Retrieve property descriptions: Challenge, page 6-7
Retrieve property descriptions: Walkthrough, page 6-9

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Retrieve property descriptions: Challenge

Challenge

Use the data in the table to complete the following activities. Create a Java class that extends CEConnectionEDU and write code to

- Retrieve a document using property filter
- Retrieve class and property descriptions

Data

Item	Value

Namespaces to import	`com.filenet.api.meta.*;`
	`com.filenet.api.collection.*;`
	`com.filenet.api.constants.*;`
	`com.filenet.api.core.*;`
	`com.filenet.api.admin.*;`
	`com.filenet.api.property.*;`

Object store Name	`"Development"`
Folder path for the document	`"/APIFolder/APIOrderDoc"`

Verification

- Run the program using the Eclipse or the Command Prompt.
- Verify that a list of names of class descriptions for a given object store is displayed.
- Verify that a list of property descriptions with the meta data for a given document class is displayed.

Sample output for the class descriptions

```
   Got the connection
   Name of the domain: P8Domain
   Name of the objectstore: Development
   APIOrderDoc  Document has been retrieved
   List of Class Descriptions:
   Class name = Document
   Class name = Workflow Definition
   Class name = XML Property Mapping Script
   Class name = Code Module
   Class name = Entry Template
...
   Class name = Product
...
```

Sample output for the property descriptions

```
   List of Property Descriptions:
   Property name = ClassDescription
   Read only? true
   Required? true
   Settability READ_ONLY
   Cardinality 0
   System? true
   Property name = This
   Read only? true
   Required? true
   Settability READ_ONLY
   Cardinality 0
   System? true
   Property name = ReplicationGroup
   Read only? false
   Required? false
   Settability READ_WRITE
   Cardinality 0
   System? false
...
```

Note

The list of class and property descriptions that you get is much longer than what is shown here.

Retrieve property descriptions: Walkthrough

Procedures: Retrieve class descriptions

Procedures: Retrieve property descriptions

Retrieve class descriptions

Procedure 1: Create a Java class that extends CEConnectionEDU

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the “New Java Class” window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type the class name `PropertiesEDU` in the Name field.
5. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
6. Select `public static void main(String[] args)` for “Which method stubs would you like to create?” option.
   a. Clear other options.
7. Click Finish to add the class and close the window.
8. Verify the new class listed under your package in the Project Explorer.
9. Write code inside the class that you just added.

10. Import the following packages before the declaration of the class:

```
java.util.Iterator
com.filenet.api.meta.*
com.filenet.api.core.*
com.filenet.api.collection. *
com.filenet.api.constants.*
com.filenet.api.admin.*
com.filenet.api.property.*
```

11. Inside the main method, write a `try-catch` block.
    a. Call `exception.printStackTrace()` in the catch block to display the exceptions thrown.
    b. Add a custom methods and call it from the main method as detailed in the following steps.

Procedure 2: Write a method to retrieve class descriptions

1. Define the method using the following signature:

- Scope: `public`
- Name: `getClassesEDU`
- Parameter: `ObjectStore` object

2. Call `objectstore.get_ClassDescriptions()`
    a. Assign the return value to a variable of type `ClassDescriptionSet`.
3. Get the iterator by calling `ClassDescriptionSet.iterator()`and assign it to an `Iterator` variable.
4. Write a `while` loop and iterate through the collection:
5. `while (iterator.hasNext())`
6. Call `iterator.next()`within the while loop.
    a. Typecast the return value into a `ClassDescription` type and assign it to a variable.
    b. Display each class description by calling `System.out.println(…).`
    - Parameter: `classDescription.get_DisplayName()`

Procedure 3: Instantiate the class and call the methods

1. Inside the `try` block of the main() method, create an instance of the class.

    Example: `PropertiesEDU propertiesInstance = new PropertiesEDU();`

2. Call the getCEConnectionEDU(...) method of the CEConnectionEDU class:
   - Parameters:
     "p8admin"
     "IBMFileNetP8"
     a. Assign the returned Connection object to a variable.
3. Call the getDomainEDU(…)method of the CEConnectionEDU class
   - Parameter: Connection object from step 2.
     a. Assign the returned Domain object to a variable.
4. Call the getObjectStoreEDU(…) method of the CEConnectionEDU class:
   - Parameters:
     - Domain object from step 3.
     - "Development"
     a. Assign the returned ObjectStore object to a variable.
5. Call getClassesEDU(...)
   - Parameter: Objectstore object from step 4.

Procedure 4: Run the program and verify the results

1. Run the program using the Eclipse or Command Prompt as instructed in the "Communication with the Content Engine Server" unit.
2. Verify that a list of names of the class descriptions in the specified object store is displayed in the Command Prompt window or the console of the Eclipse.

Sample output

Refer to the sample output in Retrieve property descriptions: Challenge, page 6-7

Retrieve property descriptions

You might have already written a method to get a document in the “Documents” unit. The following procedure gets a document by using a property filter.

Procedure 1: Write a method to retrieve a document

1. Define the method using the following signature:

- Scope: `public`
- Name: `getDocumentEDU`
- Parameters: `ObjectStore` object, `String` for the folder path
- Returns: `Document` Object

2. Create a new instance a `PropertyFilter` object.

   a. Assign the return value to a variable of type `PropertyFilter` object.

3. Call `PropertyFilter.addIncludeType(...)`.

- Parameters:

```
0
null
null
FilteredPropertyType.ANY
1
```

4. Call `Factory.Document.fetchInstance(...)`.

- Parameters:
  - `Objectstore` object
  - Folder path that is passed into this method
  - `PropertyFilter` object from step 2.

   a. Assign the return value to a variable of type `Document` object.

5. Get and display the document name by calling `System.out.println(…)`.

- Parameter: `Document.get_Name()`

6. Return the `Document` object from step 4.

Procedure 2: Write a method to retrieve property descriptions

1. Define the method using the following signature:

   - Scope: `public`
   - Name: `getPropertyDescriptionsEDU`
   - Parameter: `Engine` object (`Document` object is used)

2. Call `document.get_ClassDescription ()`

   a. Assign the return value to a variable of type `ClassDescription.`

3. Retrieve the property descriptions for the given document class by calling `classDescription.get_PropertyDescriptions()`
   a. Assign the return value to a variable of type `PropertyDescriptionList.`
4. Call `propertyDescriptionList.iterator()`
   a. Assign the returned value to a variable of the type `Iterator.`
5. Write a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```

6. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into a `PropertyDescription` type and assign it to a variable.
7. Retrieve and display the names of each property description by calling `System.out.println(…)`

   •`Parameter: propertyDescription.get_SymbolicName()`
8. Repeat step 7 to retrieve and display the metadata for the individual property:

- Use each of the following parameters:

  ```
  propertyDescription.get_IsReadOnly().toString()
  propertyDescription.get_IsValueRequired().toString()
  propertyDescription.get_Settability().toString()
  propertyDescription.get_Cardinality().getValue()
  propertyDescription.get_IsSystemGenerated().toString()
  ```

Procedure 3: Call the method inside the main() method

1. Call the `getDocumentEDU(...)` method inside the `try` block of the main() method.

- Parameters:
  - `Objectstore` variable that you got in the previous activity.
  - `“/APIFolder/APIOrderDoc"`

  a. Assign the return value to a variable of type `Document`.
2. Call the `getPropertyDescriptionsEDU(...)`method

- Parameter: `Document` variable from step1

3. Comment out the method calls that are not tested in this activity.

Procedure 4: Run the program and verify the results

1. Run the program using Eclipse or Command Prompt.
2. Verify that a list of property descriptions with the metadata for a given document class is displayed in the Output window.

Sample output

Refer to the sample output in Retrieve property descriptions: Challenge, page 6-7

Lesson 6.2. Retrieve a choice list

Overview

Why is this lesson important to you?

The custom application provides its customers with payment options when ordering a item. A choice list for the payment options has been created on the Content Engine using FileNet Enterprise Manager. As the programmer, you are going to write code to retrieve and display the choice list of payment options.

Activities

- Write code to retrieve choice list properties.

Skill levels

Retrieve a choice list: Challenge, page 6-17
Retrieve a choice list: Walkthrough, page 6-19

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Retrieve a choice list: Challenge

Challenge

Use the same Java class that you created in the previous lesson and the data in the table to write code to

- Retrieve a choice list for a given document class.
- Display the individual choice values for the choice list.

Data

Item	Value
Object store name	`"Development"`
Folder path for the document	`"/APIFolder/APIOrderDoc"`

Verification

- Run the program using the Eclipse or the Command Prompt.
- Verify that a list of choice values of the choice list for a given document class is displayed in the Command Prompt window or the console of the Eclipse.

Sample output

```
Choice list name = style
Has Hierarchy? = false
List of choices and their display names
     Basic-Basic
     Deluxe-Deluxe
     Luxury-Luxury

Choice list name = payment_type
Has Hierarchy? = true
List of choices and their display names
   CHK-Check
   CASH-Cash
   MO-Money Order
   Credit Card
   List of choices for the nested choice list
      0-Visa
      1-MasterCard
      2-AmericanExpress
      3-EnRoute
      4-Diner's Club
```


Retrieve a choice list: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to retrieve a choice list

1. Define a method using the following signature:

- Scope: `public`
- Name: `getChoiceListEDU`
- Parameter: `Document` object

2. Retrieve the class description for the given document by calling `document.get_ClassDescription()`
   a. Assign the return value to a variable of type `ClassDescription`.
3. Retrieve the property descriptions for the given document class by calling `classDescription.get_PropertyDescriptions()`
   a. Assign the return value to a variable of type `PropertyDescriptionList`.
4. Call `propertyDescriptionList.iterator()`
   a. Assign the returned value to a variable of the type `Iterator.`
5. Write a `while` loop and iterate through the collection.

```
while (iterator.hasNext())
```

6. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into a `PropertyDescription` type and assign it to a variable.
7. Retrieve the choice list by calling `propertyDescription.get_ChoiceList()`
   a. Assign the return value to a variable of type `com.filenet.api.admin.ChoiceList`.
8. Write an `if` statement to see if there is a choice list in the given property description and write the code inside the `if` block
9. Retrieve and display the name of the choice list by calling `System.out.println(…)`

•`Parameter: propertyDescription.get_SymbolicName()`

10. Retrieve the choice values collection by calling `choiceList.get_ChoiceValues(…)` on the object from step 7.
    a. Assign the return value to a variable of type `com.filenet.api.collection.ChoiceList`.
11. Retrieve and display the hierarchy information for the given choice list by calling `System.out.println(…)`.
- Parameter: `choiceList.get_HasHierarchy().toString()`on the object from step 7.
12. Retrieve and display the value and name of each choices by calling the method `displayChoiceList(…)`that you are going to write in the following procedure.
- Parameter: `ChoiceList` variable that you got in the step 10.
13. Close the `if` block and close the `while` loop.

Procedure 2: Write a method to display choice values

This method is recursive it displays individual choice values for the choice list.

1. Define a method using the following signature:
- Scope: `public`
- Name: `displayChoiceList`
- Parameter: `com.filenet.api.collection.ChoiceList` object
2. Call `choiceList.iterator()`on the object passed into this method.
    a. Assign the returned value to a variable of the type `Iterator.`
3. Retrieve the choice values by write a `while` loop and iterate through the collection.

```
while (iterator.hasNext())
```

4. Call `iterator.next()`within the `while` loop.
    a. Typecast the return value into a `Choice` type and assign it to a variable.
5. Inside the `while` loop, write an `if - else if- else` statement to see if the choice value is an Integer, a String, or a nested choice list.
6. If the choice value is an Integer `choice.get_ChoiceType().equals(ChoiceType.INTEGER)`, do the following:
    a. Call `choice.get_ChoiceIntegerValue().toString()`to get the value of the choice.
    b. Display the value and name by calling `System.out.println(…).`
7. If the choice value is a String `choice.get_ChoiceType().equals(ChoiceType.STRING)`, do the following:
    a. Call `choice.get_ChoiceStringValue()` to get the value of the choice.
    b. Display the value and name by calling `System.out.println(…)`.

8. If the choice value is neither String nor Integer, do the following:
   a. Call `choice.get_ChoiceValues()` to get the nested choice list.
   b. Assign the return value to a variable of type `com.filenet.api.collection.ChoiceList`.
   c. Display the value and name of each choices by calling this same method (recursive) `displayChoiceList(…)`.

- Parameter: `ChoiceList` object that you got in the previous step

Procedure 3: Call the method inside the main() method

1. Call the `getChoiceListEDU(...)` method inside the `try` block of the main() method.

- Parameter: `Document` variable that you got in the previous lesson

2. Comment out the method calls that are not tested in this activity.

Procedure 4: Run the program and verify the results

1. Run the program using Eclipse or Command Prompt.
2. Verify that a list of choice values of the choice list for a given document class is displayed in the Command Prompt window or the console of the Eclipse.

Sample output

Refer to the sample output in Retrieve a choice list: Challenge, page 6-17

Lesson 6.3. Retrieve object properties

Overview

Why is this lesson important to you?

To manage the properties of the Content Engine objects, you need to retrieve the properties for a given object. As a programmer, you are going to write code to retrieve and display the values of properties of a given Document class.

Activities

- Write code to retrieve object properties.

Skill levels

Retrieve object properties: Challenge, page 6-25
Retrieve a object properties: Walkthrough, page 6-27

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Retrieve object properties: Challenge

Challenge

Use the same Java class that you created in the previous lesson. Write code to retrieve object properties using the data in the table.

Data type constants	Method call for single value	Method call for multi-values
TypeID.DATE_AS_INT	property.getDateTimeValue().toString()	property.getDateTimeListValue()
TypeID.STRING_AS_INT	property.getStringValue()	property.getStringListValue()
TypeID.BOOLEAN_AS_INT	property.getBooleanValue().toString()	property.getBooleanListValue()
TypeID.DOUBLE_AS_INT	property.getFloat64Value().toString()	property.getFloat64ListValue()
TypeID.LONG_AS_INT	property.getInteger32Value().toString()	property.getInteger32ListValue()
TypeID.GUID_AS_INT	property.getIdValue().toString()	property.getIdListValue()

Verification

- Run the program using the Eclipse or Command Prompt.
- Verify that a list of names and values of the properties for a given document is displayed in the Command Prompt window or the console of the Eclipse.

Sample output

```
Name of the objectstore: Development
APIOrderDoc  Document has been retrieved
-----------------------------------
Property name = IsCurrentVersion
Property value = true
-----------------------------------
Property name = Id
Property value = {C2641447-A0EF-425A-8795-D83BC2E98101}
-----------------------------------
Property name = DateCreated
Property value = Thu Feb 19 10:16:46 PST 2009
```

Note

The list of properties that you get is much longer than what is shown here.

Retrieve a object properties: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to retrieve object properties

1. Define a method using the following signature:

- Scope: `public`
- Name: `displayPropertiesEDU`
- Parameter: `Document` object

2. Retrieve the properties collection for the given document by calling `document.getProperties()`.
   a. Assign the return value to a variable of type `com.filenet.api.property.Properties`.
3. Call `properties.iterator()`on the object from the step 2.
   a. Assign the return value to a variable of type `Iterator`.
4. Write a `while` loop and iterate through the collection.

```
while (iterator.hasNext())
```

5. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into a `Property` type and assign it to a variable.
6. Inside the `while` loop, retrieve and display the name and value of each property by calling the method `displayPropertyEDU(…)`that you are going to write in the following procedure.

- Parameters:
  - `Document` object that was passed into the method
  - `Property` object that you got

7. Close the `while` loop.

Procedure 2: Write a method to display property name and value

1. Define a method using the following signature:
   - Scope: `public`
   - Name: `displayPropertyEDU`
   - Parameters: `Document` object, `Property` object
2. Call `property.getPropertyName()` on the object that is passed into this method.
   a. Assign the return value to a variable of type `String`.

Note

Steps 3 and 7 below are the same as those in the Retrieve property descriptions: Walkthrough, page 6-9 activity. You might want to copy the code from the `getPropertyDescriptionEDU()` method.

3. Call `document.get_ClassDescription ()`
   a. Assign the return value to a variable of type `ClassDescription.`
4. Retrieve the property descriptions for the given document class by calling `classDescription.get_PropertyDescriptions()`
   a. Assign the return value to a variable of type `PropertyDescriptionList.`
5. Call `propertyDescriptionList.iterator()`
   a. Assign the returned value to a variable of the type `Iterator.`
6. Write a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```

7. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into a `PropertyDescription` type and assign it to a variable.
8. Call `propertyDescription.get_SymbolicName()`
   a. Assign the returned value to a variable of the type `String.`
9. Write an `if` statement to see if the property name from step 2 matches the name of the property description from step 8.
   a. Write the following code inside the `if` block.
10. Display the symbolic name of the PropertyDescription by calling `System.out.println(…)`
    - Parameter: String from step 8

11. Write a `switch case` statement to retrieve the value of the property.
    `switch(propertyDescription.get_DataType().getValue())`
12. Create a case for each of the Data type constant in the following table. Use the following data to perform steps 12 and 14.

Data type constants	Method call for single value	Method call for multi-values
`TypeID.DATE_AS_INT`	`property.getDateTimeValue().toString()`	`property.getDateTimeListValue()`
`TypeID.STRING_AS_INT`	`property.getStringValue()`	`property.getStringListValue()`
`TypeID.BOOLEAN_AS_INT`	`property.getBooleanValue().toString()`	`property.getBooleanListValue()`
`TypeID.DOUBLE_AS_INT`	`property.getFloat64Value().toString()`	`property.getFloat64ListValue()`
`TypeID.LONG_AS_INT`	`property.getInteger32Value().toString()`	`property.getInteger32ListValue()`
`TypeID.GUID_AS_INT`	`property.getIdValue().toString()`	`property.getIdListValue()`

13. Do the following steps for the case `TypeID.DATE_AS_INT`:
    a. Call `propertyDescription.get_Cardinality().getValue()`
    b. Assign the returned value to a variable of the type `int`.
    c. Verify that the cardinality value is a single (`Cardinality.SINGLE_AS_INT`)and the value is not `null` by using `if` statements:
    d. Retrieve the value of the property by calling `property.getDateTimeValue().toString()`
    e. Display the value of the property by calling `System.out.println(…)`
    f. Otherwise, if it is multivalued, verify that the value is not `null`.
    g. Write a method called `displayListValuesEDU(...)`in the following procedure 3 and call it inside this method to retrieve individual values from the list.

- Parameter: `property.getDateTimeListValue()`

14. Repeat step 13 to retrieve property values for other data types. Use the data table for the data type constants and methods to call.
15. End the `Switch` statement and the `if` statement.
16. Close the method.

Procedure 3: Write a method to retrieve individual property value

1. Define a method using the following signature:

- Scope: `public`
- Name: `displayListValuesEDU`
- Parameter: `List` object

2. Call `list.iterator()`on the object that is passed into this method.
    a. Assign the return value to a variable of type `Iterator.`
3. Write a `while` loop and iterate through the collection.

    ```
    while (iterator.hasNext())
    ```

4. Call `iterator.next()`within the `while` loop.
    a. Typecast the return value into an `Object` type and assign it to a variable.
    b. Retrieve and display the each value for the property by calling `System.out.println(…)`
    - Parameter:`object.toString()`
5. Close the `while` loop and close the method.

Procedure 4: Call the method inside the main() method

1. Call the `displayPropertiesEDU(...)` method inside the `try` block of the main() method.

- Parameter:
    - `Document` variable that you got in the previous lesson

2. Comment out the method calls that are not tested in this activity.

Procedure 5: Run the program and verify the results

1. Run the program using Eclipse or Command Prompt.
2. Verify that a list of names and values of the properties for a given document is displayed in the Command Prompt window or the console of the Eclipse.

Sample output

Refer to the sample output in Retrieve object properties: Challenge, page 6-25

Solution code for PropertiesEDU.java

```
package com.ibm.filenet.edu;

import com.filenet.api.meta.*;
import com.filenet.api.collection.*;
import com.filenet.api.constants.*;
import com.filenet.api.core.*;
import com.filenet.api.admin.*;
import com.filenet.api.property.*;

import java.util.*;
@SuppressWarnings("unchecked")
public class PropertiesEDU extends CEConnectionEDU
{
/* getClassDescriptionsEDU */
public void getClassesEDU(ObjectStore os)
{
   ClassDescriptionSet classDescriptions = os.get_ClassDescriptions();
   ClassDescription classDesc;
   Iterator it = classDescriptions.iterator();
   while (it.hasNext())
   {
      classDesc = (ClassDescription)it.next();
      System.out.println("    Class name = " + classDesc.get_DisplayName());
   } // while
}


/* getPropertyDescriptionsEDU */
public void getPropertyDescriptionsEDU(Document document)
{
   ClassDescription classDescription = document.get_ClassDescription();
   PropertyDescriptionList propDescs =
   classDescription.get_PropertyDescriptions();
   propertyDescription propDesc;
   Iterator propIt = propDescs.iterator();
   while (propIt.hasNext())
   {
      propDesc = (PropertyDescription)propIt.next();
      System.out.println("    Property name = " +
      propDesc.get_SymbolicName());
      System.out.println("    Read only? " +
      propDesc.get_IsReadOnly().toString());
      System.out.println("    Required? " +
```

```
      propDesc.get_IsValueRequired().toString());
      System.out.println("    Settability " +
      propDesc.get_Settability().toString());
      System.out.println("    Cardinality " +
      propDesc.get_Cardinality().getValue());
      System.out.println("    System? " +
      propDesc.get_IsSystemGenerated().toString());
   }
}


/* getChoiceListEDU */
public void getChoiceListEDU(Document document)
{
   ClassDescription classDescription = document.get_ClassDescription();
   PropertyDescriptionList propDescs =
   classDescription.get_PropertyDescriptions();
   PropertyDescription propDesc;
   Iterator propIt = propDescs.iterator();
   while (propIt.hasNext())
      {
      propDesc = (PropertyDescription)propIt.next();
      com.filenet.api.admin.ChoiceList choiceList =
      propDesc.get_ChoiceList();
      if (choiceList !=null)
      {
         System.out.println("  ");
         System.out.println("Choice list name = " +
         propDesc.get_SymbolicName());
         com.filenet.api.collection.ChoiceList choices
         =choiceList.get_ChoiceValues();
         System.out.println("Has Hierarchy? = " +
         choiceList.get_HasHierarchy().toString());
         System.out.println("List of choices and their display names");
         displayChoiceList(choices);
      }//if
      } // while
} // getChoiceListEDU

public void displayChoiceList( com.filenet.api.collection.ChoiceList
choices)
{
   Choice choice;
   Iterator choiceIt = choices.iterator();
```



```
   while (choiceIt.hasNext())
   {
      choice = (Choice)choiceIt.next();
      if (choice.get_ChoiceType().equals(ChoiceType.INTEGER)){
         System.out.println("     " +
         choice.get_ChoiceIntegerValue().toString()+ "-" +
         choice.get_DisplayName());
      }
      else if(choice.get_ChoiceType().equals(ChoiceType.STRING)){
         System.out.println("     " + choice.get_ChoiceStringValue()+ "-" +
         choice.get_DisplayName());
      }
      else {
         com.filenet.api.collection.ChoiceList subChoices;
         subChoices = choice.get_ChoiceValues();
         System.out.println("    ---" + choice.get_DisplayName());
         System.out.println("    List of choices for the nested choice
         list");
         displayChoiceList(subChoices);
         }
   }//while choiceIt
}// displayChoiceList

/*displayChoiceList */
public void displayPropertiesEDU(Document document)
{
   com.filenet.api.property.Properties properties =
   document.getProperties();
   Property property;
   Iterator it = properties.iterator();
   while (it.hasNext())
   {
      property = (Property)it.next();
      displayPropertyEDU(document, property);
      System.out.println("-----------------------------------");
   }
}
public void displayPropertyEDU(Document document, Property property)
{
   String propertyName = property.getPropertyName();
   ClassDescription classDescription = document.get_ClassDescription();
   PropertyDescriptionList propDescs =
   classDescription.get_PropertyDescriptions();
   Iterator propIt = propDescs.iterator();
```

```
PropertyDescription propertyDescription = null;
while (propIt.hasNext())
   {
   propertyDescription = (PropertyDescription)propIt.next();
   String symbolicName = propertyDescription.get_SymbolicName();
   if (symbolicName.equalsIgnoreCase(propertyName)){
   System.out.println("Property name = " + symbolicName);
   System.out.print("Property value = ");
   switch(propertyDescription.get_DataType().getValue()){
      case TypeID.DATE_AS_INT:
      if (propertyDescription.get_Cardinality().getValue() ==
      Cardinality.SINGLE_AS_INT)
      {
      if (property.getDateTimeValue() != null)
      System.out.println(property.getDateTimeValue().toString());
      }
      else
      if (property.getDateTimeListValue() != null)
      displayListValuesEDU(property.getDateTimeListValue());
      break;
      case TypeID.STRING_AS_INT:
      if (propertyDescription.get_Cardinality().getValue() ==
      Cardinality.SINGLE_AS_INT)
      System.out.println(property.getStringValue());
      else
      displayListValuesEDU(property.getStringListValue());
      break;
      case TypeID.BOOLEAN_AS_INT:
      if (propertyDescription.get_Cardinality().getValue() ==
      Cardinality.SINGLE_AS_INT){
      if (property.getBooleanValue() != null)
      System.out.println(property.getBooleanValue().toString());
      }
      else
      if (property.getBooleanListValue() != null)
      displayListValuesEDU(property.getBooleanListValue());
      break;
      case TypeID.DOUBLE_AS_INT:
      if (propertyDescription.get_Cardinality().getValue() ==
      Cardinality.SINGLE_AS_INT){
      if (property.getFloat64Value() != null)
      System.out.println(property.getFloat64Value().toString());
      }
      else
```



```
        displayListValuesEDU(property.getFloat64ListValue());
        break;
        case TypeID.LONG_AS_INT:
        if (propertyDescription.get_Cardinality().getValue() ==
        Cardinality.SINGLE_AS_INT)
        {
        if (property.getInteger32Value() != null)
        System.out.println(property.getInteger32Value().toString());
        }
        else
        displayListValuesEDU(property.getInteger32ListValue());
        break;
        case TypeID.GUID_AS_INT:
        if (propertyDescription.get_Cardinality().getValue() ==
        Cardinality.SINGLE_AS_INT)
        {
        if (property.getIdValue() != null)
        System.out.println(property.getIdValue().toString());
        }
        else
        displayListValuesEDU(property.getIdListValue());
        break;
        default:
        system.out.println("");
        break;
     }//Switch
     }//if
   }//While

}

public void displayListValuesEDU(List list)
{
Iterator listIt = list.iterator();
Object value;
while (listIt.hasNext())
   {
      Value = (Object)listIt.next();
      System.out.println(value.toString());
   }
}

public Document getDocumentEDU(ObjectStore store){
```

```
   PropertyFilter pf = new PropertyFilter();
   pf.addIncludeType(0, null, null, FilteredPropertyType.ANY,1);
   Document document =
   Factory.Document.fetchInstance(store,"/APIFolder/APIOrderDoc", pf);
   String documentName = document.get_Name();
   System.out.println(documentName + "  Document has been retrieved");
   return document;
}

public static void main(String[] args) {
   try
   {
      PropertiesEDU myInstance = new PropertiesEDU();
      Connection conn =
      myInstance.getCEConnectionEDU("p8admin","IBMFileNetP8");
      Domain domain = myInstance.getDomainEDU(conn);
      ObjectStore store =
      myInstance.getObjectStoreEDU(domain,"Development");
      Document document = myInstance.getDocumentEDU(store);
      //  System.out.println(" List of Class Descriptions: ");
      //myInstance.getClassesEDU(store);
      // System.out.println(" List of Property Descriptions: ");
      // myInstance.getPropertyDescriptionsEDU(document);
      //  myInstance.getChoiceListEDU(document);
      myInstance.displayPropertiesEDU(document);
   }
   catch (Exception e){
      System.out.println(e.getMessage());
      e.printStackTrace();
      }
   }
}
```


Unit 7. Searches

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class in this unit extends the CEConnectionEDU class from the “Communication with the Content Engine” unit.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. (These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.)

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 7.1. Search for objects

Overview

Why is this lesson important to you?

Your application requires a feature that searches for Document objects on the given Content Engine. You are going to write code to accomplish this.

Activities

Search for objects: Challenge, page 7-7
Search for objects: Walkthrough, page 7-9

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Search for objects: Challenge

Challenge

Create a Java class that extends CEConnectionEDU. Use the data in the table to write code to search for the following:

- Documents based on the properties
- Folders (Optional)

Display the names of the documents and folders retrieved.

Data

Item	Value

Packages to import	`java.util.Iterator`
	`com.filenet.api.collection.*`
	`com.filenet.api.core.*`
	`com.filenet.api.query.*`
	`com.filenet.api.constants.*`
	`com.filenet.api.property.*`

Object store name	`"Development"`
SQL statement for searching the documents	`"Select * from Document"`
SQL statement for searching the documents in a given folder	`"Select D.DocumentTitle from Product D INNER JOIN ReferentialContainmentRelationship r ON D.This = r.Head WHERE r.Tail = OBJECT('/Manuals')"`
SQL statement for searching the folders	`"Select * from Folder WHERE Parent=OBJECT('/Products')"`

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that a list of names of the documents or folders based on the search criteria is displayed in the Command prompt window or in the console of Eclipse.

Sample output (documents)

```
Name of the domain: P8Domain
Name of the objectstore: Development
document=In Planning Luxury Model 400.doc
document=User Manual for Basic Model A.doc
document=User Manual for Basic Model A.pdf
```

Sample output (folders)

```
Name of the domain: P8Domain
Name of the objectstore: Development
Folder= Model 320
Folder= Model 300
Folder= Luxury Models
Folder= Model C
Folder= Model 200
Folder= Model A
Folder= Basic Models
Folder= Deluxe Models
...
```

Note

The list of documents or folders that you get can be longer than what is shown here.

Search for objects: Walkthrough

Procedures

Procedure 1: Create a Java class that extends CEConnectionEDU

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the "New Java Class" window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type the class name `SearchEDU` in the Name field.
5. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
6. Select `public static void main(String[] args)` for "Which method stubs would you like to create?" option.
   a. Clear other options.
7. Click Finish to add the class and close the window.
8. Verify that the new class is listed under your package in the Project Explorer.
9. Write code inside the class that you just added.
10. Import the following packages before the declaration of the class:

```
java.util.Iterator
com.filenet.api.core.*
com.filenet.api.collection.*
com.filenet.api.constants.*
com.filenet.api.query.*
com.filenet.api.property.*
```

11. Inside the main method, write a `try-catch` block.
    a. Call `exception.printStackTrace()` in the catch block to display the exceptions thrown.

b. Add a custom methods and call it from the main method as detailed in the following steps.

Procedure 2: Write a method to search for a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `searchDocumentsEDU`
   - Parameter: `ObjectStore` object
2. Use the following SQL statement for searching the documents and assign it to a `String` variable:

   `"Select * from Document"`
3. Create an instance of SearchScope class.

   `new SearchScope(...)`
   - Parameter: `ObjectStore` object

   a. Assign the return value to a variable of type `SearchScope`.
4. Create an instance of SearchSQL class.

   `new SearchSQL(...)`
   - Parameter: `String` variable for SQL statement from step 2.

   a. Assign the return value to a variable of type `SearchSQL`.
5. Call `SearchScope.fetchObjects(...)`to retrieve the documents.
   - Parameters:
     - `SearchSQL` object from step 4
     - `Integer.valueOf("50")`
     - `null`
     - `Boolean.valueOf(true)`

   a. Assign the return value to a variable of type `DocumentSet`.
6. Call `documentSet.iterator()`

   a. Assign the returned value to a variable of the type `Iterator`.
7. Write a `while` loop and iterate through the collection.

   `while (iterator.hasNext())`
8. Call `iterator.next()`within the `while` loop.

   a. Typecast the return value into a `Document` type and assign it to a variable.

   b. Display the name of each document by calling `System.out.println(…)`.
      - Parameter: `doc.get_Name()`
9. Close the `while` loop.

10. Close the method.
11. Optional: Use the following additional SQL statements for step 2 and repeat the steps in procedure 2 to search Content Engine documents or folders

Objects Searched	SQL statements
Documents in a given folder	`"Select D.DocumentTitle from Product D INNER JOIN ReferentialContainmentRelationship r ON D.This = r.Head WHERE r.Tail = OBJECT('/Manuals')"`
Folders	`"Select * from Folder WHERE Parent = OBJECT('/Products')"`

12. Optional: If you use the SQL statement to retrieve the folders collection, complete the following steps:
    a. Repeat steps 3, 4, and 5 in procedure 2.
    b. Then complete steps 13 through 17 to retrieve individual folders and display their names.
13. Assign the return value from step 5 to a variable of type `FolderSet`.
14. Call `folderSet.iterator()`.
    a. Assign the returned value to a variable of the type `Iterator`.
15. Write a `while` loop and iterate through the collection.

    ```
    while (iterator.hasNext())
    ```

16. Call `iterator.next()`within the `while` loop.
    a. Typecast the return value into a `Folder` type and assign it to a variable.
    b. Display each folder name by calling `System.out.println(…)`.
       - Parameter: `folder.get_FolderName()`
17. Close the `while` loop.
18. Close the method.

Procedure 3: Instantiate the class and call the methods

1. Inside the `try` block of the main() method, create an instance of the class.

   Example: `SearchEDU searchInstance = new SearchEDU();`

2. Call the `getCEConnectionEDU(...)` method of the `CEConnectionEDU` class:
   - Parameters:
     - `"p8admin"`
     - `"IBMFileNetP8"`

   a. Assign the returned `Connection` object to a variable.
3. Call the `getDomainEDU(…)`method of the `CEConnectionEDU` class.
   - Parameter: `Connection` object variable from step 2

   a. Assign the returned `Domain` object to a variable.
4. Call the `getObjectStoreEDU(…)` method of the `CEConnectionEDU` class:
   - Parameters:
     - `Domain` object variable from step 3
     - `"Development"`

   a. Assign the returned `ObjectStore` object to a variable.
5. Call the `searchDocumentsEDU(...)` method.
   - Parameter: `ObjectStore` variable from step 4

Procedure 4: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt as instructed in the "Communication with the Content Engine Server" unit.
2. Verify that a list of names of the documents (folders, for the optional activity) for a given object store is displayed in the Command prompt window or in the console of Eclipse.

Sample output

Refer to the sample output in Search for objects: Challenge, page 7-7

Lesson 7.2. Search for objects with paging

Overview

Why is this lesson important to you?

Your application requires a feature that searches for Document objects on the given Content Engine and returns documents as sets per page. You are going to write code to accomplish this.

Activities

Search for objects with paging: Challenge, page 7-15
Search for objects with paging: Walkthrough, page 7-17

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Search for objects with paging: Challenge

Challenge

- Use the same Java class that you created in the previous lesson.
- Write code to search for the documents in the given object store.
- Display the document titles with paging.

Data

Item	Value
Object store name	`"Development"`
SQL statement for searching the documents	`"Select * from Product where Creator='P8Admin'"`

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that a list of names of the documents based on the search criteria is displayed with paging in the Command prompt window or in the console of Eclipse.

Sample output

```
Name of the domain: P8Domain
Name of the objectstore: Development
query = Select * from Product where Creator='P8Admin'

Page: 1 Element count: 5
page size=5
document=User Manual for Basic Model A.pdf
document=User Manual for Basic Model B.pdf
document=User Manual for Basic Model C.pdf
document= Model 200.JPG
document=Research Information Basic Model B.doc

Page: 2 Element count: 5
page size=5
document= Approved Basic Model A.doc
document=APIOrderDoc
document=Research Information Basic Model A.doc
document=User Manual for Basic Model A.doc
document= Approved Deluxe Model 120.doc

Page: 3 Element count: 5
page size=5

...
```

Note

The list of documents that you get can be longer than what is shown here.

Search for objects with paging: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to search for a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `searchPagedDocumentsEDU`
   - Parameter: `ObjectStore` object
2. Use the following SQL statement for searching the documents created by a given user and assign it to a `String` variable:

   ```
   "select * from Product where Creator='P8Admin'"
   ```
3. Create an instance of SearchScope class.

   ```
   new SearchScope(...)
   ```
   - Parameter: `ObjectStore` object

   a. Assign the return value to a variable of type `SearchScope`.
4. Create an instance of SearchSQL class.

   ```
   new SearchSQL(...)
   ```
   - Parameter: `String` variable for SQL statement from step 2.

   a. Assign the return value to a variable of type `SearchSQL`.
5. Call `SearchScope.fetchObjects(...)`to retrieve the documents.
   - Parameters:
     - `SearchSQL` object from step 4
     - `Integer.valueOf("50")`
     - `null`
     - `Boolean.valueOf(true)`

   a. Assign the return value to a variable of type `DocumentSet`.
6. Call `documentSet.pageIterator()`.

   a. Assign the returned value to a variable of the type `PageIterator`.

7. Call pageIterator.getPageSize() on the object from step 6.
   a. Display the page size by calling System.out.println(...).
8. Call pageIterator.setPageSize(5) on the object from step 6.
9. Write a while loop and iterate through the collection.

```
int pageCount = 0;
while (pageIter.nextPage() == true)
```

10. Add a counter: pageCount++;

```
pageIter.getElementCount();
```

11. Call pageIterator.getElementCount() on the object from step 6.
    a. Assign the return value to an int type variable.
    b. Display the page number and element count by calling System.out.println(…).
       - Parameters: page count from step 10 and element count from step 11 as a String
12. Get the current page by calling pageIterator.getCurrentPage().
    a. Assign the return value to an Object[] type variable.
13. Call pageIterator.getPageSize() on the object from step 6.
    a. Display the page size by calling System.out.println(...).
14. Write a for loop with the variable from step 12 (in the following lines of code, the pageObjects variable name is used):

```
for (int index = 0; index < pageObjects.length; index++)
   {
      Document document = (Document)pageObjects[index];
      System.out.println("document = "+ document.get_Name());
   }
```

15. Close the for loop.
16. Close the while loop.
17. Close the method.

Procedure 2: Call the method inside the main() method

1. Call the searchPagedDocumentsEDU(...) method that you wrote inside the main() method.
   - Parameters:
     - Objectstore variable that you got in the previous lesson
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.
2. Verify that a list of the names of the documents for the given condition is displayed with paging in the in the Command Prompt window or in the console of Eclipse.

Sample output

Refer to the sample output in Search for objects with paging: Challenge, page 7-15

Lesson 7.3. Search for objects across object stores

Overview

Why is this lesson important to you?

Your application requires a feature that searches for Document objects across multiple object stores on the given Content Engine. You are going to write code to accomplish this.

Activities

Search for objects across object stores: Challenge, page 7-23
Search for objects across object stores: Walkthrough, page 7-25

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Search for objects across object stores: Challenge

Challenge

- Use the same Java class that you created in the previous lesson.
- Write code to search for the documents in the given object stores.
- Display the document titles.

Data

Item	Value
Object store names	`"Development", “Sales”`
SQL statement for searching the documents	`"Select * from Product d where Creator='P8Admin' AND NOT (IsClass(d, CodeModule))"`

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that a list of names of the documents based on the search criteria is displayed with paging in the Command prompt window or in the console of Eclipse.

Sample output (multiple object store search)

```
Name of the domain: P8Domain
Name of the objectstore: Development
Name of the objectstore: Sales
----------------------
Object Store Name: Development
document-1= In Planning Luxury Model 400.doc
----------------------
Object Store Name: Sales
document-2= Approved Basic Model A.doc
----------------------
Object Store Name: Development
document-3= User Manual for Basic Model A.doc
----------------------
Object Store Name: Development
document-4= In Planning Luxury Model 420.doc
```

Note

The list of documents that you get can be longer than what is shown here.

Search for objects across object stores: Walkthrough

Use the same Java class that you created in the previous activity.

Procedure 1: Write a method to search across the object stores

1. Define the method using the following signature:

- Scope: `public`
- Name: `multipleObjectStoresSearchEDU`
- Parameters:
  - `ObjectStore` object for the first object store to search
  - `ObjectStore` object for the second object store to search

2. Declare an `ObjectStore` type array and assign the two object stores as the values.
3. Create an instance of SearchScope class.

- Parameters:
  - `ObjectStore` array from step 2
  - `MergeMode.UNION`

   a. Assign the return value to a variable of type SearchScope.

4. Use the following SQL statement for retrieving documents and assign it to a `String` variable.

   ```
   "SELECT * FROM Document d WHERE Creator='CEAdmin' AND NOT (IsClass(d,
   CodeModule))"
   ```

5. Create an instance of SearchSQL class.

- Parameter: `String` variable for SQL statement from step 4

   a. Assign the return value to a variable of type `SearchSQL`.

6. Call `SearchScope.fetchObjects(...)`

- Parameters:
  - `SearchSQL` object from step 5
  - `Integer.getInteger("50")`
  - `null`
  - `Boolean.valueOf(true)`

   a. Assign the return value to a variable of type `DocumentSet`.

7. Call `documentSet.iterator()`

   a. Assign the returned value to a variable of the type `Iterator`.

8. Write a `while` loop and iterate through the collection to retrieve each document.

   a. Call `iterator.next()`within the `while` loop.

   b. Typecast the return value into a `Document` type and assign it to a variable.

c. Retrieve the object store from where each document is stored by calling `document.getObjectStore()`

d. Assign the return value to a variable of type `ObjectStore.`

e. Call `objectStore.refresh()`

f. Display the name of the object store by calling `System.out.println(…)`

- Parameter: `objectStore.get_Name()`

g. Display the name of each document by calling `System.out.println(…)`

- Parameter: `doc.get_Name()`

h. Close the `while` loop.

i. Optionally, add a counter to track the number of documents retrieved.

Procedure 2: Call the method in the form

1. Call the `getObjectStoreEDU(…)` method of the `CEConnectionEDU` class:

- Parameters:
  - `Domain` object variablethat you got in the previous lesson
  - `"Development"`

a. Assign the returned `ObjectStore` object to a variable.

b. You might have got this object store already in the previous activity, in which case skip step 1.

2. Repeat step 1 to get the `"Sales"` object store.
3. Call the `multipleObjectStoresSearchEDU(...)` method.

- Parameters:
  - `Objectstore` variable from step 1
  - `Objectstore` variable from step 2

4. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program and verify that a list of the names of the documents for the given object stores is displayed in the Command Prompt window or in the console of Eclipse.

Sample output

Refer to the sample output in Search for objects across object stores: Challenge, page 7-23

Lesson 7.4. Build SQL statements

Overview

Why is this lesson important to you?

Your application requires a feature that searches for Document objects on the given Content Engine. You want to use the IBM FileNet P8 capabilities for constructing SQL statements to run the search. As the programmer, you are going to write code to accomplish this.

Activities

Build SQL statements: Challenge, page 7-29
Build SQL statements: Walkthrough, page 7-31

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Build SQL statements: Challenge

Challenge

- Use the same Java class that you created in the previous lesson.
- Write code to construct SQL statement using Java API helper methods in the table and search for the documents in the given object store: `Development`
- Display the document titles.

Data

Helper method	Parameter value
`SearchSQL.setSelectList(...)`	`"Name, DocumentTitle"`
`SearchSQL.setFromClauseInitialValue(...)`	`"Product","d", True`
`SearchSQL.setWhereClause(...)`	`"Creator='P8Admin' AND NOT (IsClass(d, CodeModule))"`

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that a list of names of the documents for the given conditions is displayed in the Command prompt window or in the console of Eclipse.

Sample output

```
Name of the domain: P8Domain
Name of the objectstore: Development
Document title = User Manual for Basic Model A.pdf
Document title = User Manual for Basic Model B.pdf
Document title = User Manual for Basic Model C.pdf
...
Document title = In Planning Luxury Model 400.doc
```

Note

The list of documents that you get can be longer than what is shown here.

Build SQL statements: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to construct SQL statement and query an object store

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `searchSQLEDU`
   - Parameter: `ObjectStore` object
2. Create an instance of SearchScope class.

   `new SearchScope(...)`
   - Parameter: `ObjectStore` object

   a. Assign the return value to a variable of type `SearchScope`.
3. Create an instance of SearchSQL class.

   a. Assign the return value to a variable of type `SearchSQL`.
4. Call `searchSQL.setSelectList(...)`.
   - Parameter: `"Name, DocumentTitle"`
5. Call `searchSQL.setFromClauseInitialValue(...)`.
   - Parameters:
     - `"Product"`
     - `"d"`
     - `True`
6. Call `searchSQL.setWhereClause(...)`.
   - Parameter: `"Creator='P8Admin' AND NOT (IsClass(d, CodeModule))"`

7. Call SearchScope.fetchObjects(...).
   - Parameters:
     - SearchSQL object from steps 4 through 6
     - Integer.valueOf("50")
     - null
     - Boolean.valueOf(true)

   a. Assign the return value to a variable of type DocumentSet.
8. Call documentSet.iterator().
   a. Assign the returned value to a variable of the type Iterator.
9. Write a while loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```
10. Call iterator.next()within the while loop.
    a. Typecast the return value into a Document type and assign it to a variable.
11. Retrieve and display the document title of each document by calling System.out.println(…).
    - Parameter: "Document title = " + doc.getProperties().getStringValue("DocumentTitle")
12. Close the while loop and the method.

Procedure 2: Call the method inside the main() method

1. Call the searchSQLEDU(...) method that you wrote inside the main() method.
   - Parameters:
     - Objectstore variable that you got in the previous lesson
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.
2. Verify that a list of the name of the documents for the given condition is displayed in the in the Command Prompt window or in the console of Eclipse.

Sample output

Refer to the sample output in Build SQL statements: Challenge, page 7-29

Lesson 7.5. Content-based retrieval

Overview

Why is this lesson important to you?

Your application requires a feature that searches for Document objects on the given Content Engine based on the content. As the programmer, you are going to write code to accomplish this.

Activities

Search for documents based on content: Challenge, page 7-35
Search for documents based on content: Walkthrough, page 7-39

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Search for documents based on content: Challenge

Challenge

Use the same Java class that you created in the previous lesson and the data from the data table below to write code to do the following:

- Search for documents based on their content.
- Search for repository rows of properties.
- Display the document titles of the documents and the Rank property.

Data

Item	Value
Object store name	`"Development"`
SQL statement for searching the documents	`"SELECT d.This, DocumentTitle, Name FROM Product d INNER JOIN VerityContentSearch v ON v.QueriedObject = d.This WHERE d.IsCurrentVersion = TRUE AND CONTAINS(d.*,'" + keyWord + "')"`
SQL statement for searching the repository rows of properties	`"SELECT DocumentTitle, Rank FROM Product d INNER JOIN VerityContentSearch v ON v.QueriedObject = d.This WHERE d.IsCurrentVersion = TRUE AND CONTAINS(d.*,'" + keyWord + "')"`
Keyword	`“model”`

Optional: Use these additional SQL statements for search

1. Search for documents that contain the two keywords (*model* and *deluxe*) in a sentence (use with <SENTENCE>).

   ```
   "SELECT d.This, DocumentTitle, Name FROM Product d INNER JOIN
   VerityContentSearch v ON v.QueriedObject = d.This WHERE
   d.IsCurrentVersion = TRUE AND CONTAINS(d.*,'model <SENTENCE> Deluxe')"
   ```

2. Search for documents that contain the two keywords (*tested* and *passed*) located close to each other (use with <NEAR>).

   ```
   "SELECT d.This, DocumentTitle, Name FROM Product d INNER JOIN
   VerityContentSearch v ON v.QueriedObject = d.This WHERE
   d.IsCurrentVersion = TRUE AND CONTAINS(d.*,'tested <NEAR/25> passed')"
   ```

SQL statements for the IBM Content Search Services

1. Proximity Search similar to NEAR with Autonomy. Search for documents that contain the two keywords (*loan* and *payments*) located close to each other:

```
SELECT This, DocumentTitle, Name FROM Loan D INNER JOIN ContentSearch CS
ON CS.QueriedObject = D.This WHERE CONTAINS(D.*, '\"loan payments\" ~3')
```

2. Search for XML documents that contain the keyword for an element.

```
SELECT This, DocumentTitle, Name FROM Loan D INNER  JOIN ContentSearch CS
ON CS.QueriedObject=D.This WHERE CONTAINS(D.*, '@xmlxp:'
'/Loan/customer_name [.contains(\"Jane\")]' ' ')
```

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that a list of document titles of the documents containing the given keyword for a given object store is displayed in the Command prompt window or in the console of Eclipse.

Sample output (documents)

```
Name of the objectstore: Development
document=Deluxe Models.ppt
document=User Manual for Basic Model B.doc
document=User Manual for Basic Model C.doc
document=Research Information Basic Model C.doc
document=Research Information Basic Model A.doc
document=Research Information Basic Model B.doc
document=User Manual for Basic Model C.pdf
```

Note

The list of documents that you get can be longer than what is shown here.

Sample output (repository row of properties)

```
DocumentTitle = Deluxe Models.ppt
Rank = 0.8658
DocumentTitle = User Manual for Basic Model B.doc
Rank = 0.8351
DocumentTitle = User Manual for Basic Model C.doc
Rank = 0.8351
DocumentTitle = Research Information Basic Model C.doc
Rank = 0.8351
DocumentTitle = User Manual for Basic Model A.doc
Rank = 0.8351
DocumentTitle = Research Information Basic Model A.doc
Rank = 0.8351
```

Note

The list of documents that you get can be longer than what is shown here.

Search for documents based on content: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to search for documents

1. Define a method using the following signature:
   - Scope: `public`
   - Name: `contentSearchEDU`
   - Parameters:
     - `ObjectStore` object
     - `String` for keyword
2. Create an instance of SearchScope class: new `SearchScope(...)`
   - Parameter: `ObjectStore` object

   a. Assign the return value to a variable of type `SearchScope`.
3. Use the following SQL statement for retrieving the documents containing the key word and assign it to a `String` variable:

   ```
   "SELECT d.This, DocumentTitle, Name FROM Product d INNER JOIN
   VerityContentSearch v ON v.QueriedObject = d.This WHERE
   d.IsCurrentVersion = TRUE AND CONTAINS(d.*,'" + keyWord + "')"
   ```
4. Create an instance of SearchSQL class: new `SearchSQL(...)`
   - Parameter: `String` variable for SQL statement from step 3

   a. Assign the return value to a variable of type `SearchSQL`.
5. Call `searchScope.fetchObjects(...)` to retrieve the documents.
   - Parameters:
     - `SearchSQL` object from step 4.
       - `Integer.valueOf("50")`
       - `null`
       - `Boolean.valueOf(true)`

   a. Assign the return value to a variable of type `DocumentSet`.

6. Call documentSet.iterator().
   a. Assign the returned value to a variable of the type Iterator.
7. Write a while loop and iterate through the collection.
8. Call iterator.next()within the while loop.
   a. Typecast the return value into Document type and assign it to a variable.
   b. Display the name of each document by calling System.out.println(…).
      - Parameter: doc.get_Name()
9. Close the while loop.

Procedure 2: Retrieve repository rows and display properties

Write code inside the same method that you created in procedure 1.

1. Use the following SQL statement for retrieving the repository rows of properties and assign it to a String variable:

   ```
   "SELECT DocumentTitle, Rank FROM Product d INNER JOIN VerityContentSearch
   v ON v.QueriedObject = d.This WHERE d.IsCurrentVersion = TRUE AND
   CONTAINS(d.*,'" + keyWord + "')"
   ```

2. Create an instance of SearchSQL class.

   ```
   new SearchSQL(...)
   ```

   - Parameter: String variable for SQL statement from step 1

   a. Assign the return value to a variable of type SearchSQL.
3. Call searchScope.fetchRows(...)to retrieve the repository rows of properties.
   - Parameters:
     - SearchSQL object from step 2
     - Integer.valueOf("50")
     - null
     - Boolean.valueOf(true)

   a. Assign the return value to a variable of type RepositoryRowSet.
4. Call repositoryRowSet.iterator().
   a. Assign the returned value to a variable of the type Iterator.
5. Write a while loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```

   a. Call iterator.next()within the while loop.
   b. Typecast the return value into RepositoryRow type and assign it to a variable.
   c. Call repositoryRow.getProperties().

d. Assign the return value to a variable of type `Properties`.

e. Call `properties.iterator()`.

f. Assign the returned value to a variable of the type `Iterator`.

6. Write a second `while` loop and iterate through the collection.

   a. Call `iterator.next()`within the `while` loop.

   b. Typecast the return value into `com.filenet.api.property.Property` type and assign it to a variable.

7. Write an `if` statement to verify the name of the property is "Rank"

   ```
   if (property.getPropertyName().equalsIgnoreCase("Rank"))
   ```

   a. Inside the if statement retrieve the property name by calling `property.getPropertyName()`.

   b. Retrieve the property value by calling `property.getFloat64Value().toString()`

   c. Display the name and value of the property.

8. Inside the `else` condition, retrieve the property name by calling `property.getPropertyName()`.

   a. Retrieve the property value by calling `property.getStringValue()`.

   b. Display the name and value of the property.

9. Complete the method:

   a. Close the second `while` loop.

   b. Close the first `while` loop.

   c. Close the method.

Procedure 3: Additional SQL statements for search

Use these additional SQL statements for search in procedure 1 step 3.

1. Search for documents that contain the two keywords (*model* and *deluxe*) in a sentence (use with <SENTENCE>).

   ```
   "SELECT d.This, DocumentTitle, Name FROM Product d INNER JOIN
   VerityContentSearch v ON v.QueriedObject = d.This WHERE
   d.IsCurrentVersion = TRUE AND CONTAINS(d.*,'model <SENTENCE> Deluxe')"
   ```

2. Search for documents that contain the two keywords (*tested* and *passed*) located close to each other (use with <NEAR>).

   ```
   "SELECT d.This, DocumentTitle, Name FROM Product d INNER JOIN
   VerityContentSearch v ON v.QueriedObject = d.This WHERE
   d.IsCurrentVersion = TRUE AND CONTAINS(d.*,'tested <NEAR/25> passed')"
   ```

3. SQL statements for the IBM Content Search Services - Proximity Search similar to NEAR with Autonomy:

   Search for documents that contain the two keywords (*loan* and *payments*) located close to each other:

   ```
   SELECT This, DocumentTitle, Name FROM Loan D INNER JOIN ContentSearch CS
   ON CS.QueriedObject = D.This WHERE CONTAINS(D.*, '\"loan payments\" ~3')
   ```

4. SQL statements for the IBM Content Search Services - Search for XML documents that contain the keyword for an element.

   ```
   SELECT This, DocumentTitle, Name FROM Loan D INNER  JOIN ContentSearch CS
   ON CS.QueriedObject=D.This WHERE CONTAINS(D.*, '@xmlxp:'
   '/Loan/customer_name [.contains(\"Jane\")]' ' ')
   ```

Procedure 4: Call the method inside the main() method

1. Call the `contentSearchEDU(...)` method inside the `try` block of the main() method.
   - Parameters:
     - `Objectstore` variable that you got in the previous lesson
     - `"model"` (key word for the search)
2. Comment out the method calls that are not tested in this activity.

Procedure 5: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.
2. Verify that document titles of the documents containing the given keyword for a given object store is displayed in the Command Prompt window or in the console of Eclipse.

Sample output

Refer to the sample output in Search for documents based on content: Challenge, page 7-35

Solution code for SearchEDU.java

```
package com.ibm.filenet.edu;

import java.util.Iterator;
import com.filenet.api.collection.*;
import com.filenet.api.core.*;
import com.filenet.api.query.*;
import com.filenet.api.constants.*;
import com.filenet.api.property.*;

public class SearchEDU extends CEConnectionEDU{

public void searchDocumentsEDU(ObjectStore os)
   {
   SearchScope search = new SearchScope(os);
   String sql1 = "select * from Document";
   SearchSQL searchSQL = new SearchSQL(sql1);
   //String sql2 = "Select D.DocumentTitle from Product D INNER JOIN
   ReferentialContainmentRelationship r ON D.This = r.Head WHERE r.Tail =
   OBJECT('/Manuals')";
   //SearchSQL searchSQL = new SearchSQL(sql2);
   DocumentSet documents = (DocumentSet)search.fetchObjects
   (searchSQL,Integer.valueOf("50"), null, Boolean.valueOf(true));
   Document doc;
   Iterator DocIt = documents.iterator();
   while (DocIt.hasNext())
   {
      doc = (Document)DocIt.next();
      System.out.println("document="+ doc.get_Name());
   }
   String sqlFolder = "select * from Folder WHERE
   Parent=OBJECT('/Products')";
   SearchSQL searchSQL = new SearchSQL(sqlFolder);
   FolderSet folders;
   folders = (FolderSet)search.fetchObjects(searchSQL,
   Integer.valueOf("50"), null, Boolean.valueOf(true));
   Folder folder;
   Iterator folderIt = folders.iterator();
   while (folderIt.hasNext()){
      folder = (Folder)folderIt.next();
      System.out.println("Folder= "+ folder.get_FolderName());
   }
}
```

```
void searchPagedDocumentsEDU(ObjectStore os)
{
   SearchScope search = new SearchScope(os);
   //SearchSQL sql = new SearchSQL("select * from Product where
   Creator='p8admin'");
   SearchSQL sql = new SearchSQL("select * from Document");
   System.out.println("query= " + sql);
   DocumentSet documents =
   (DocumentSet)search.fetchObjects(sql,Integer.valueOf("5"), null,
   Boolean.valueOf(true));
   PageIterator pageIter = documents.pageIterator();
   System.out.println("Page size=" + pageIter.getPageSize());
   pageIter.setPageSize(5);
   int pageCount = 0;
   while (pageIter.nextPage() == true)
      {
      // display number of objects on this page
      pageCount++;
      int elementCount = pageIter.getElementCount();
      System.out.println("Page: "
      + pageCount + "  Element count: " + elementCount);
      Object[] pageObjects = pageIter.getCurrentPage();
      System.out.println("Page size=" + pageIter.getPageSize());
      for (int index = 0; index < pageObjects.length; index++)
         {
         Document document = (Document)pageObjects[index];
         System.out.println("document = "+ document.get_Name());
         }
      }
}

public void multipleObjectStoresSearchEDU(ObjectStore os1, ObjectStore os2)
{
   ObjectStore objectStores[] = new ObjectStore[2];
   objectStores[0] = os1;
   objectStores[1] = os2;

   PropertyFilter pf = new PropertyFilter();
   pf.addIncludeProperty(new FilterElement(Integer.getInteger("2"), null,
   Boolean.valueOf (true), "Name", Integer.getInteger("1")));
   SearchScope search = new SearchScope(objectStores, MergeMode.UNION);
   SearchSQL sql = new SearchSQL("select * from Product d where
   Creator='P8Admin' AND NOT (IsClass(d, CodeModule))");
   DocumentSet documents =
```



```
   (DocumentSet)search.fetchObjects(sql,Integer.getInteger("50"),pf,
   Boolean.valueOf(true));
   Document doc;
   ObjectStore objectStore;
   Iterator it = documents.iterator();
   Integer count = 1;
   while (it.hasNext())
   {
      doc = (Document)it.next();
      objectStore = doc.getObjectStore();
      objectStore.refresh();
      System.out.println("----------------------");
      System.out.println("Object Store Name: " + objectStore.get_Name());
      System.out.println("document-" + count.toString() + "= " +
      doc.get_Name());
      count = count + 1;
   }
}

public void searchSQLEDU(ObjectStore os)
{
   SearchScope search = new SearchScope(os);
   SearchSQL searchSQL = new SearchSQL();
   //searchSQL.setSelectList("*");
   searchSQL.setSelectList("Name, DocumentTitle");
   //searchSQL.setFromClauseInitialValue("Document", "d", true);
   searchSQL.setFromClauseInitialValue("Product", "d", true);
   //searchSQL.setWhereClause("Creator='P8Admin' AND NOT (IsClass(d,
   CodeModule))");
   //searchSQL.setWhereClause("Creator='CEadmin' AND NOT (IsClass(d,
   CodeModule))");
   DocumentSet documents =(DocumentSet)search.fetchObjects
   (searchSQL,Integer.valueOf("50"),null, Boolean.valueOf(true));
   Document doc;
   Iterator it = documents.iterator();
   while (it.hasNext()){
      doc = (Document)it.next();
      System.out.println("Document title = " +
      doc.getProperties().getStringValue("DocumentTitle"));
      }
}

public void contentSearchEDU(ObjectStore os, String keyWord)
{
```

```
   SearchScope search = new SearchScope(os);

   String mySQLString = "SELECT d.This, DocumentTitle, Name FROM Product d";
   mySQLString = mySQLString + "INNER JOIN VerityContentSearch v ON
   v.QueriedObject = d.This ";
   mySQLString = mySQLString + "WHERE d.IsCurrentVersion = TRUE AND
   CONTAINS(d.*,'" + keyWord + "')";

   /*
   String mySQLString2 = "SELECT d.This, DocumentTitle, Name FROM Product d
   ";
   mySQLString2 = mySQLString2 + "INNER JOIN VerityContentSearch v ON
   v.QueriedObject = d.This ";
   mySQLString2 = mySQLString2 + "WHERE d.IsCurrentVersion = TRUE AND
   CONTAINS(d.*,'Model <NEAR/25> Deluxe')";
   */

   /* Search for documents that contain the two keywords (tested and passed)
   located close to each other (use with <NEAR>) */
   /*
   String mySQLString3 = "SELECT d.This, DocumentTitle, Name FROM Product d
   INNER JOIN VerityContentSearch v ON v.QueriedObject = d.This WHERE
   d.IsCurrentVersion = TRUE AND CONTAINS(d.*,'tested <NEAR/25> passed')";
   */
/*For IBM Content Search Services - Similar to the NEAR with Autonomy*/
   String mySQLString4 = "SELECT This, DocumentTitle, Name FROM Loan D INNER
   JOIN ContentSearch CS ON CS.QueriedObject = D.This WHERE CONTAINS(D.*,
   '\"loan payments\" ~3')";
/*For IBM Content Search Services - XML Documents Search */
   String mySQLString5 = "SELECT This, DocumentTitle, Name FROM Loan D INNER
   JOIN ContentSearch CS ON CS.QueriedObject=D.This WHERE CONTAINS(D.*,
   '@xmlxp:' '/Loan/customer_name [.contains(\"Jane\")]' ' ');
   SearchSQL sql = new SearchSQL(mySQLString);
   // SearchSQL sql = new SearchSQL(mySQLString2);
   DocumentSet documents =
   (DocumentSet)search.fetchObjects(sql,Integer.valueOf("50"),null,
   Boolean.valueOf(true));
   com.filenet.api.core.Document doc;
   Iterator it = documents.iterator();
   while (it.hasNext()){
      doc = (Document)it.next();
      System.out.println("document="+ doc.get_Name());
   }
   System.out.println("======================");
```



```
String rowSQLString = "SELECT DocumentTitle, Rank FROM Product d ";
rowSQLString = rowSQLString + "INNER JOIN VerityContentSearch v ON
v.QueriedObject = d.This ";
rowSQLString = rowSQLString + "WHERE d.IsCurrentVersion = TRUE AND
CONTAINS(d.*,'" + keyWord + "')";

/*
String rowSQLString2 = "SELECT DocumentTitle, Rank FROM Product d ";
rowSQLString2 = rowSQLString2 + "INNER JOIN VerityContentSearch v ON
v.QueriedObject = d.This ";
rowSQLString2 = rowSQLString2 + "WHERE d.IsCurrentVersion = TRUE AND
CONTAINS(d.*,'Model <SENTENCE> Deluxe')";

String rowSQLString3 = "SELECT DocumentTitle, Rank FROM Product d INNER
JOIN VerityContentSearch v ON v.QueriedObject = d.This WHERE
d.IsCurrentVersion = TRUE AND CONTAINS(d.*,'tested <NEAR/25> passed')";
*/
SearchSQL sqlRow = new SearchSQL(rowSQLString);

System.out.println(sqlRow.toString());
RepositoryRowSet rowSet = search.fetchRows(sqlRow, Integer.valueOf("50"),
null, Boolean.valueOf(true));
Iterator itRow = rowSet.iterator();
com.filenet.api.property.Property prop;
com.filenet.api.property.Properties props;
RepositoryRow row;
while (itRow.hasNext()) {
   row = (RepositoryRow)itRow.next();
   props = row.getProperties();
   Iterator itProp = props.iterator();
   while (itProp.hasNext()){
      prop = (com.filenet.api.property.Property)itProp.next();
      if (prop.getPropertyName().equalsIgnoreCase("Rank")){
         System.out.println(prop.getPropertyName() + " = " +
         prop.getFloat64Value().toString());
      }
      else{
         System.out.println(prop.getPropertyName()+ " = " +
         prop.getStringValue());
      }
   }
}
System.out.println("--------------------------");
```

```
}


public void showTables(ObjectStore store)
{
   TableDefinitionSet tables = store.get_TableDefinitions();
   Iterator it = tables.iterator();
   while (it.hasNext())
   {
      TableDefinition table = (TableDefinition)it.next();
      System.out.println("Table = " + table.get_Name() + "; table name=" +
      table.get_TableName());
   }
}

public static void main(String[] args) {
   try
      {

         SearchEDU myInstance = new SearchEDU();
         Connection conn =
         myInstance.getCEConnectionEDU("p8admin","IBMFileNetP8");
         Domain domain = myInstance.getDomainEDU(conn);
         ObjectStore store = myInstance.getObjectStoreEDU(domain,
         "Development");
         // myInstance.showTables(store);
         // myInstance.searchDocumentsEDU(store);
         myInstance.searchPagedDocumentsEDU(store);
         // myInstance.searchSQLEDU(store);
         //  myInstance.contentSearchEDU(store, "model");
         System.out.println("Done");
      }
      catch (Exception e) {
            e.printStackTrace();
      }
   }
}
```


Unit 8. Document Versions

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class in this unit extends the CEConnectionEDU class from the “Communication with the Content Engine” unit.
- Concepts and some parts of the code from Documents unit is used.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt window.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference.

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 8.1. Retrieve versionable objects

Overview

Why is this lesson important to you?

Amy is a member of the Product Development team. Her documents go through many steps from draft, to revised, to released. All of these documents are stored on the content Engine as different versions. Amy needs to work with previous versions of some documents. As the team programmer, you are going to write code to retrieve prior versions of a document.

Prerequisites

- Activity , Create a document and document versions, page 8-7

Activities

- Retrieve versionable objects: Challenge, page 8-9
- Retrieve versionable objects: Walkthrough, page 8-11

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Create a document and document versions

Important

This activity is required for both skill levels.

1. Sign in to Workplace as p8admin and navigate to Development > APIFolder.
2. Create a new document using the specifications in the following data table.

Item	Value
Document class	`Document`
Document title	`APIVersions.doc`
Add as major version	`Yes`
Location for the content for the document	`C:\CMJavaAPIProg\SampleDocuments\APIVersions.doc`

3. Check out the document and then check in the document as a major version.
   - These actions create a new version of the document.
   - You can repeat the check out and check in to create more versions.
   - Use this document for the following activities.

Retrieve versionable objects: Challenge

Challenge

Use the data in the table to create a Java class that extends CEConnectionEDU and write code to retrieve the following:

- A document with version related properties
- Versions for a document
- VersionSeries utility object for a document

Data

Item	Value

Packages to import	`java.util.*`
	`java.io.*`
	`com.filenet.api.core.*`
	`com.filenet.api.constants.*`
	`com.filenet.api.collection.*`
	`com.filenet.api.property.*`

Object store name	`"Development"`
Folder path for the document	`"/APIFolder/APIVersions.doc"`

Document properties to retrieve	`PropertyNames.NAME`
	`PropertyNames.MINOR_VERSION_NUMBER`
	`PropertyNames.MAJOR_VERSION_NUMBER`
	`PropertyNames.VERSION_SERIES`
	`PropertyNames.IS_CURRENT_VERSION`
	`PropertyNames.IS_RESERVED`
	`PropertyNames.RESERVATION`
	`PropertyNames.VERSIONS`
	`PropertyNames.VERSION_STATUS`
	`PropertyNames.CURRENT_VERSION`

Verification

Run the program using Eclipse or the Command Prompt and verify that the following are displayed:

- A list of versions with the version number and the version status for a given document
- Version numbers of the released version and the current version of a given document

Sample output (Document versions)

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIVersions.doc  Document is retrieved
Version =3.0,  Version status =RELEASED
Version =2.1,  Version status =SUPERSEDED
Version =2.0,  Version status =SUPERSEDED
Version =1.0,  Version status =SUPERSEDED
```

Sample output (VersionSeries)

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIVersions.doc  Document is retrieved
Released major version =3
Released minor version =0
Current major version =3
Current minor version =0
```


Retrieve versionable objects: Walkthrough

Procedures: Retrieve versions for a document

Procedures: Retrieve a VersionSeries for a document

Retrieve versions for a document

Procedure 1: Create a Java class that extends CEConnectionEDU

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the “New Java Class” window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type the class name `VersionsEDU` in the Name field.
5. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
6. Select `public static void main(String[] args)` for “Which method stubs would you like to create?” option.
   g. Clear other options.
7. Click Finish to add the class and close the window.
   a. Verify that the new class is listed under your package in the Project Explorer.

8. Write code inside the class that you just added.
9. Import the following packages before the declaration of the class:

```
java.util.*
java.io.*
com.filenet.api.core.*
com.filenet.api.constants.*
com.filenet.api.collection.*
com.filenet.api.property.*
```

Procedure 2: Write a method to retrieve a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getDocumentEDU`
   - Parameters:
     - `ObjectStore` object
     - `String` for the folder path
   - Returns: `Document` Object
2. Create a new instance a `PropertyFilter` object.
   a. Assign the return value to a variable of type `PropertyFilter` object.
3. Call `PropertyFilter.addIncludeProperty(...)`.
   - Parameters:
     - `0`
     - `null`
     - `null`
     - `PropertyNames.NAME`
     - `1`
4. Repeat step 3 to include the following properties in the property filter:

```
PropertyNames.MINOR_VERSION_NUMBER
PropertyNames.MAJOR_VERSION_NUMBER
PropertyNames.VERSION_SERIES
PropertyNames.IS_CURRENT_VERSION
PropertyNames.IS_RESERVED
PropertyNames.RESERVATION
PropertyNames.VERSIONS
PropertyNames.VERSION_STATUS
PropertyNames.CURRENT_VERSION
```


5. Call `Factory.Document.fetchInstance(...)`.
   - Parameters:
     - `Objectstore` object that is passed into this method
     - Folder path that is passed into this method
     - `PropertyFilter` object that you got in step 2
6. Get and display the document name by calling `System.out.println(…)`.
   - Parameter: `Document.get_Name()`
7. Return the `Document` object.

Procedure 3: Write a method to get versions for a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: showVersionsEDU
   - Parameter: `Document` object
2. Retrieve the versions by calling `Document.get_Versions()`.
   a. Assign the return value to a variable of type `VersionableSet`.
3. Call `versionableSet.iterator()`.
   a. Assign the returned value to a variable of the type `Iterator`.
4. Write a `while` loop to retrieve each versionable object from the collection.

   ```
   while (iterator.hasNext())
   ```

5. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into a `Versionable` type and assign it to a variable.
6. Call `versionable.get_MajorVersionNumber()` to retrieve the major version number for each versionable object.
   a. Assign the returned value to a variable of the type `Integer`.
7. Call `versionable.get_MinorVersionNumber()` to retrieve the minor version number.
   a. Assign the returned value to a variable of the type `Integer`.
8. Call `versionable.get_VersionStatus().toString()` to retrieve the version status.
   a. Assign the returned value to a variable of the type `String`.
9. Display the major and minor version numbers and version status by calling `System.out.println(…)` and using each of the following parameters:
   - `Integer` from step 6 for major version number
   - `Integer` from step 7 for major version number
   - `String` from step 8 for version status
10. Close the `while` loop and the method.

Procedure 4: Instantiate the class and call the methods

1. Inside the `try` block of the main() method, create an instance of the class.

   Example: `VersionsEDU versionsInstance = new VersionsEDU();`

2. Call the `getCEConnectionEDU()` method of the `CEConnectionEDU` class:
   - Parameters:
     - `"p8admin"`
     - `"IBMFileNetP8"`

   a. Assign the returned `Connection` object to a variable.

3. Call the `getDomainEDU(…)`method of the `CEConnectionEDU` class.
   - Parameter: `Connection` object from step 2

   a. Assign the returned `Domain` object to a variable.

4. Call the `getObjectStoreEDU(…)` method of the `CEConnectionEDU` class:
   - Parameters:
     - `Domain` object from step 3
     - `“Development”`

   a. Assign the returned `ObjectStore` object to a variable.

5. Call `getDocumentEDU(...)`.
   - Parameters:
     - `ObjectStore` variable that you got in step 4
     - `"/APIFolder/APIVersions.doc"`

   a. Assign the return value to a variable of type `Document`.

6. Call `showVersionsEDU(...)`.
   - Parameter: `Document` object from step 5

Procedure 5: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt as instructed in the “Communication with the Content Engine Server” unit.
2. Verify that a list of versions with the version numbers and the version status for a given document is displayed in the Command prompt window or in the console of Eclipse.

Sample output (Document versions)

Refer to the sample output in Retrieve versionable objects: Challenge, page 8-9

Retrieve a VersionSeries for a document

Use the same Java class that you created in the previous lesson. In this activity, you are going to retrieve the VersionSeries utility object for a given Document and display the information.

Procedure 1: Write a method to get version series for a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `showVersionSeriesEDU`
   - Parameter: `Document` object
2. Retrieve the VersionSeries object by calling the `document.get_VersionSeries()` method.
   a. Assign the return value to a variable of type `VersionSeries`.
3. Retrieve the released version of the document by calling the `VersionSeries.get_ReleasedVersion()`.
   a. Type cast the returned value to a `Document` object type and assign it to a variable.
4. Call `document.get_MajorVersionNumber()` on the object from step 3 to retrieve the major version number.
   a. Display the version number by calling `System.out.println(...)`.
5. Repeat step 4 to retrieve and display the minor version number by calling `document.get_MinorVersionNumber()`.
6. Retrieve the current version of the document by calling `versionSeries.get_CurrentVersion` ().
   a. Type cast the returned value to a `Document` object type and assign it to a variable.
7. Call `document.get_MajorVersionNumber()` on the object from step 6 to retrieve the major version number.
   a. Display the version number by calling `System.out.println(...)`.
8. Repeat step 7 to retrieve and display the minor version number by calling `document.get_MinorVersionNumber()`.

Procedure 2: Call the method inside the main() method

1. Call the `showVersionSeriesEDU(...)` method inside the `try` block of the main() method.
   - Parameter: `Document` variable from the previous activity
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.
2. Verify that version numbers for the released version and for the current version of a given document are displayed in the output window.

Sample output (VersionSeries)

Refer to the sample output in Retrieve versionable objects: Challenge, page 8-9

Lesson 8.2. Check out and check in a document

Overview

Why is this lesson important to you?

- Allen and Amanda are two product development authors. Allen writes a draft for a new document and adds it to the Content Engine. Amanda needs to check out Allen's draft, review and edit it, and check it back in. As their programmer, you are going to write code to check out and check in a document.
- Amanda has checked out Allen's draft. Allen wants to add some more content before Amanda reviews it. Amanda is going to release the document to Allen by canceling her checkout. As their programmer, you are going to write code to cancel the checkout of a document.

Activities

- Check out, check in, and cancel checkout of a document: Challenge, page 8-19
- Check out, check in, and cancel checkout of a document: Walkthrough, page 8-23

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Check out, check in, and cancel checkout of a document: Challenge

Challenge

Use the same Java class that you created in the previous lesson. Use the data from the table and write code to

- Check out the document that you retrieved in the previous lesson.
- Check in the document that you checked out.
- Check out the document again to use it for the next step.
- Cancel the checkout for the document that you checked out in the previous step.
- Retrieve and display the version information after each action on the document. Reuse the code from the previous lesson.

Data

Item	Value
Object store name	`"Development"`
Folder path for the document	`"/APIFolder/APIVersions.doc"`
Content location for checking in	`"C:\\CMJavaAPIProg\\SampleDocuments\\Model 200.GIF"`

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that, after each action, the correct version of the document is displayed in the Command Prompt window or in the console of Eclipse.
- Optionally, use Content Engine Enterprise Manager or Workplace to verify the current version of the document.

Sample output (Check out)

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIVersions.doc  Document is retrieved
The document is checked out
Version =3.1,  Version status =RESERVATION
Version =3.0,  Version status =RELEASED
Version =2.1,  Version status =SUPERSEDED
Version =2.0,  Version status =SUPERSEDED
Version =1.0,  Version status =SUPERSEDED
```

Sample output (Check-in)

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIVersions.doc  Document is retrieved
The document is checked In
Version = 4.0,  Version status = RELEASED
Version = 3.0,  Version status = SUPERSEDED
Version = 2.1,  Version status = SUPERSEDED
Version = 2.0,  Version status = SUPERSEDED
Version = 1.0,  Version status = SUPERSEDED
```

Sample output (Second-time check out to use the document for cancel checkout)

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIVersions.doc  Document is retrieved
The document is checked out
Version = 4.1,  Version status = RESERVATION
Version = 4.0,  Version status = RELEASED
Version = 3.0,  Version status = SUPERSEDED
Version = 2.1,  Version status = SUPERSEDED
Version = 2.0,  Version status = SUPERSEDED
Version = 1.0,  Version status = SUPERSEDED
```


Sample output (Cancel checkout)

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIVersions.doc  Document is retrieved
Checked out is cancelled
Version = 4.0,  Version status = RELEASED
Version = 3.0,  Version status = SUPERSEDED
Version = 2.1,  Version status = SUPERSEDED
Version = 2.0,  Version status = SUPERSEDED
Version = 1.0,  Version status = SUPERSEDED
```


Check out, check in, and cancel checkout of a document: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures: Check out a document

Procedures: Check in a document

Procedures: Cancel a checkout for a document

Check out a document

Procedure 1: Write a method to check out a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: checkout`EDU`
   - Parameter: `Document` object
2. Write an `if` statement to check whether the given document is the current version by calling `document.get_IsCurrentVersion().booleanValue()`.
3. If the document is not the current version, get the current version from the document by calling `document.get_CurrentVersion()`.
   a. Typecast the return value into a `Document` type and assign it to a variable.

4. Write an `if` statement to check that the given document is not reserved using the `document.get_IsReserved().booleanValue()`.
   a. Inside the `if` statement, call `document.checkout(...)`.
      - Parameters:
        - `ReservationType.EXCLUSIVE`
        - `null`
        - `null`
        - `null`
   b. Call `document.save(...)`.
      - Parameter: `RefreshMode.REFRESH`
   c. Display a message that the checkout was successful by calling `System.out.println(...)`.
   d. Optionally, call the `showVersionsEDU(Document)` method that you wrote in the previous lesson to display the versions.
5. If the document is already reserved, display a message that the document is already checked out and cannot be checked out.

Procedure 2: Call the method inside the main() method

1. Call the `checkoutEDU(...)` method inside the `try` block of the main() method.
   - Parameter: `Document` variable that you got in the previous lesson
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program and verify that, after the checkout, the correct version of the document is displayed in output window.
2. Optionally, use Content Engine Enterprise Manager or Workplace to verify that the document is checked out.

Sample output (Check out)

Refer to the sample output in Check out, check in, and cancel checkout of a document: Challenge, page 8-19.

Check in a document

Use the same Java class that you created in the previous lesson.

Procedure 1: Write a method to check in a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `checkinEDU`
   - Parameter: `Document` object
2. Write an `if` statement to check whether the given document is the current version using the `document.get_IsCurrentVersion().boolieanValue()` method.
   a. If the document is not the current version, get the current version by calling `document.get_CurrentVersion()`.
   b. Typecast the return value to a `Document` type and assign it to a variable.
3. Write an `if` statement to ensure the following:
   - The current version document is reserved using the `document.get_IsReserved().booleanValue()` method.
   - The version status of the current version is `VersionStatus.RESERVATION_AS_INT` using the `document.get_VersionStatus().getValue()` method.
4. If the document is not in reservation, get the reservation object by calling `document.get_Reservation()` method.
   a. Typecast the return value to a `Document` type and assign it to a variable.
   b. Inside the `if` statement, write the following code, as described in steps 4 through 12 to check in the document with new content.
   c. You can also copy this part of code from “Create Document objects” lesson of the Documents unit.
5. Call `Factory.ContentElement.createList()`.
   a. Assign the return value to a variable of type `ContentElementList`.
6. Call `Factory.ContentTransfer.createInstance()`.
   a. Assign the return value to a variable of type `ContentTransfer`.
7. Create a new instance of `FileInputStream` using the constructor.
   a. Parameter: “`C:\\CMJavaAPIProg\\SampleDocument\\Model 200.GIF`”
8. Assign the return value to a variable of type `FileInputStream`.
9. Call `ContentTransfer.setCaptureSource(...)` on the object from step 6.
   - Parameter: `FileInputStream` object that you got in the previous step

10. Call `ContentTransfer.setContentType(...)`.
    - Parameter: `"image/gif"`
11. Call `ContentElementList.add(...)`.
    - Parameter: `ContentTransfer` object that you got in the previous step
12. Call `document.set_ContentElements(...)`.
    - Parameter: `ContentElementList` that you created in the previous steps
13. Call `document.checkin(...)`.
    - Parameters:
      - `AutoClassify.DO_NOT_AUTO_CLASSIFY`
      - `CheckinType.MAJOR_VERSION`
14. Save the changes to the object by calling the `document.save(...)` method.
    - Parameter: `RefreshMode.REFRESH`
15. Optionally, call the `showVersionsEDU(Document)` method that you wrote in the previous lesson to display the versions.

Procedure 2: Call the method inside the main() method

1. Call the `checkinEDU(...)` method inside the `try` block of the main() method.
   - Parameter: `Document` variable that you got in the previous lesson
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program and verify that, after the checkin, the correct version of the document is displayed in the Command Prompt window or in the console of Eclipse.
2. Optionally, use Content Engine Enterprise Manager or Workplace to verify that the document is checked in.

Sample output (Check in)

Refer to the sample output in Check out, check in, and cancel checkout of a document: Challenge, page 8-19

Cancel a checkout for a document

Use the same Java class that you created in the previous lesson.

Procedure 1: Check out a document

1. Sign in to Workplace as p8admin and navigate to Development > APIFolder.
2. Check out the document specified in the following data table and use the checked out document in the following procedure.

Item	Value
Document name	`APIVersions.doc`
Folder where the document is filed	`APIFolder`

Procedure 2: Write a method to cancel a checkout of a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `cancelCheckoutEDU`
   - Parameter: `Document` object
2. Write an `if` statement to check whether the given document is the current version using the `document.get_IsCurrentVersion().booleanValue()` method.
   a. If the document is not the current version, get the current version by calling `document.get_CurrentVersion()`.
   b. Typecast the return value to a `Document` type and assign it to a variable.
3. Write an `if` statement to ensure that the current version document is reserved using the `document.get_IsReserved().booleanValue()` method.
   a. Inside the `if` statement, call `document.cancelCheckout()`.
   b. Typecast the return value to a `Document` type and assign it to a variable.
4. Save the changes to the object in step 3 by calling the `document.save(...)` method.
   - Parameter: `RefreshMode.REFRESH`
5. Call the `document.refresh(...)` method on the object from step 2.
6. Optionally, call the `showVersionsEDU(IDocument)` method that you wrote in the previous lesson to display the versions.

Procedure 3: Call the method inside the main() method

1. Call the `cancelCheckoutEDU(...)` method inside the `try` block of the main() method.
   - Parameter: `Document` variable that you got in the previous lesson
2. Comment out the method calls that are not tested in this activity.

Procedure 4: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt window.
2. Verify that the correct version of the document after canceling the checkout is displayed in the Command Prompt window or in the console of Eclipse.
3. Optionally, use Content Engine Enterprise Manager or Workplace to verify that the check out was canceled.

Sample output (Cancel checkout)

Refer to the sample output in Check out, check in, and cancel checkout of a document: Challenge, page 8-19

Lesson 8.3. Promote and demote a document

Overview

Why is this lesson important to you?

Tom reviews the documents drafted by the team and releases them to the general audience if they are satisfactory. One of the released documents has an incorrect entry for the product price. He needs to demote that document to draft version status. As their programmer, you are going to write code to promote and demote a document.

Activities

- Promote and demote a document: Challenge, page 8-31
- Promote and demote a document: Walkthrough, page 8-33

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Promote and demote a document: Challenge

Challenge

Use the same Java class that you created in the previous lesson. Use the data from the table and write code to

- Demote the document that you got in the “Retrieve versionable objects” lesson.
- Promote the document.
- Display the version information for the document before and after each action.

Data

Item	Value
Object store name	`"Development"`
Folder path for the document	`"/APIFolder/APIVersions.doc"`

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that, after demoting the document, the correct version is displayed in the Command prompt window or in the console of Eclipse.
- Optionally, use Content Engine Enterprise Manager or Workplace to verify that the document was demoted to a minor version.
- Repeat the same verification after promoting the document.

Sample output (Demote)

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIVersions.doc  Document is retrieved
The document is demoted
Version = 3.1,  Version status = IN_PROCESS
Version = 3.0,  Version status = RELEASED
Version = 2.1,  Version status = SUPERSEDED
Version = 2.0,  Version status = SUPERSEDED
Version = 1.0,  Version status = SUPERSEDED
Released major version = 3
Released minor version = 0
Current major version = 3
Current minor version = 1
```

Sample output (Promote)

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIVersions.doc  Document is retrieved
The document is promoted
Version = 4.0,  Version status = RELEASED
Version = 3.0,  Version status = SUPERSEDED
Version = 2.1,  Version status = SUPERSEDED
Version = 2.0,  Version status = SUPERSEDED
Version = 1.0,  Version status = SUPERSEDED
Released major version = 4
Released minor version = 0
Current major version = 4
Current minor version = 0
```


Promote and demote a document: Walkthrough

Procedures: Demote a version for a document

Procedures: Promote a version for a document

Demote a version for a document

Use the same Java class that you created in the previous lesson.

Procedure 1: Write a method to demote a document

1. Define a method using the following signature:
   - Scope: `public`
   - Name: `demoteEDU`
   - Parameter: `Document` object
2. Write an `if` statement to check whether the given document is the current version by calling `document.get_IsCurrentVersion().booleanValue()`.
3. If the document is not the current version, get the current version by calling `document.get_CurrentVersion()`.
   a. Typecast the return value to a `Document` type and assign it to a variable.
4. Write an `if` statement using the following conditions for the object from step 3:
   - Version status is `VersionStatus.RELEASED_AS_INT` by calling the `document.get_VersionStatus().getValue()`.
   - Current version is not reserved by calling the `document.get_IsReserved.booleanValue()`.
5. Inside the `if` statement, call `document.demoteVersion()`.
6. Call `document.save(...)` to save the changes.
   - Parameter: `RefreshMode.REFRESH`
7. Display a message that the document is demoted successfully by calling `System.out.println(...)`.

8. Optionally, call the showVersionsEDU(Document) and the showVersionSeriesEDU(Document) methods that you wrote in the previous lesson to display the versions.

Procedure 2: Call the method inside the main() method

1. Call the demoteEDU(...) method inside the `try` block of the main() method.
   - Parameter: `Document` object that you retrieved in the “Retrieve versionable objects” lesson.
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt window.
2. Verify that the correct version after demoting the document is displayed in the output window.
3. Optionally, use Content Engine Enterprise Manager or Workplace to verify that the document was demoted to a minor version.

Sample output (Demote)

Refer to the sample output in Promote and demote a document: Challenge, page 8-31

Promote a version for a document

Use the same Java class that you created in the previous lesson.

Procedure 1: Write a method to promote a document

1. Define a method using the following signature:
   - Scope: `public`
   - Name: `promoteEDU`
   - Parameter: `Document` object
2. Write an `if` statement to check whether the given document is the current version by calling `document.get_IsCurrentVersion().booleanValue()`.
3. If the document is not the current version, get the current version by calling `document.get_CurrentVersion()`.
   a. Typecast the return value to a `Document` type and assign it to a variable.
4. Write an `if` statement using the following conditions for the object from step 3:
   - Version status is `VersionStatus.IN_PROCESS_AS_INT` by calling the `document.get_VersionStatus().getValue()`.
   - Current version is not reserved by calling the `document.get_IsReserved.booleanValue()`.
5. Inside the `if` statement, call `document.promoteVersion()`.
6. Call `document.save(...)` to save the changes.
   - Parameter: `RefreshMode.REFRESH`
7. Display a message that the document is promoted successfully by calling `System.out.println(...)`.
8. Optionally, call the `showVersionsEDU(Document)` and the `showVersionSeriesEDU(Document)` methods that you wrote in the previous lesson to display the versions.

Procedure 2: Call the method inside the main() method

1. Call the `promoteEDU(...)` method inside the `try` block of the main() method.
   - Parameter: `Document` object that you retrieved in the “Retrieve versionable objects” lesson.
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.

2. Verify that the correct version after promoting the document is displayed in the output window.
3. Optionally, use Content Engine Enterprise Manager or Workplace to verify that the document was promoted to a major version.

Sample output (Promote)

Refer to the sample output in Promote and demote a document: Challenge, page 8-31

Solution code for VersionsEDU.java

```
package com.ibm.filenet.edu;

import java.util.*;
import java.io.*;
import com.filenet.api.collection.*;
import com.filenet.api.constants.*;
import com.filenet.api.core.*;
import com.filenet.api.property.*;
@SuppressWarnings("unchecked")
public class VersionsEDU extends CEConnectionEDU {

public void showVersionSeriesEDU(Document document)
{
      VersionSeries versionSeries = document.get_VersionSeries();
      Document releasedDoc = (Document)versionSeries.get_ReleasedVersion();
      Integer majorNum = releasedDoc.get_MajorVersionNumber();
      Integer minorNum = releasedDoc.get_MinorVersionNumber();
      System.out.println("Released major version =" + majorNum);
      System.out.println("Released minor version =" + minorNum);

      Document CurMajorDoc = (Document)versionSeries.get_CurrentVersion();
      Integer CurMajorNum = CurMajorDoc.get_MajorVersionNumber();
      Integer CurMinorNum = CurMajorDoc.get_MinorVersionNumber();
      System.out.println("Current major version =" + CurMajorNum);
      System.out.println("Current minor version =" + CurMinorNum);
}

public void showVersionsEDU(Document document)
{
   VersionableSet versions = document.get_Versions();
   Versionable versionable;
   Iterator it = versions.iterator();
   while (it.hasNext())
   {
      versionable = (Versionable)it.next();
      Integer majorNum = versionable.get_MajorVersionNumber();
      Integer minorNum = versionable.get_MinorVersionNumber();
      System.out.print("Version = " + majorNum +"."+ minorNum);
      String versionStatus = versionable.get_VersionStatus().toString();
      System.out.println(",  Version status = " + versionStatus);
   }
}
```

```
public void checkoutEDU(Document document)
{
if (document.get_IsCurrentVersion().booleanValue()== false)
document = (Document)document.get_CurrentVersion();
if (document.get_IsReserved().booleanValue() == false){
      document.checkout(ReservationType.EXCLUSIVE, null, null, null);
      document.save(RefreshMode.REFRESH);
      System.out.println("The document is checked out");
      showVersionsEDU(document);
}
else
      System.out.println("The document is already checked out. Can not check
      out");
}

public void checkinEDU(Document document) throws FileNotFoundException
{
   if (document.get_IsCurrentVersion().booleanValue()== false){
      document = (Document)document.get_CurrentVersion();
      }
      if (document.get_IsReserved().booleanValue() == true {
         if(document.get_VersionStatus().getValue() !=
         VersionStatus.RESERVATION_AS_INT)){
            document = (Document)document.get_Reservation();
      }
      ContentElementList contentList = Factory.ContentElement.createList();
      com.filenet.api.core.ContentTransfer content1 =
      Factory.ContentTransfer.createInstance();
      content1.setCaptureSource(new FileInputStream("Model 200.GIF"));
      content1.set_ContentType("image/gif");
      contentList.add(content1);
      document.set_ContentElements(contentList);
      document.checkin(AutoClassify.DO_NOT_AUTO_CLASSIFY,
      CheckinType.MAJOR_VERSION);
      document.save(RefreshMode.REFRESH);
      System.out.println("The document is checked In");
      showVersionsEDU(document);
   }
}

public void cancelCheckoutEDU(Document document)
{
// Make sure we're looking at the current version
if ( document.get_IsCurrentVersion().booleanValue() == false )
      document = (Document)document.get_CurrentVersion();
```



```
// Can only cancel checkout if it actually is already checked out
if ( document.get_IsReserved().booleanValue() == true ){
      Document reservationdoc = (Document)document.cancelCheckout();
      reservationdoc.save(RefreshMode.REFRESH);
      System.out.println("Checked out is cancelled");
      document.refresh();
      showVersionsEDU(document);
}
else
      System.out.println("Cannot cancel checkout");
}

public void demoteEDU(Document document)
{
if (document.get_IsCurrentVersion().booleanValue() == false)
   document = (Document)document.get_CurrentVersion();
   if ((document.get_VersionStatus().getValue() ==
   VersionStatus.RELEASED_AS_INT) &&
   (document.get_IsReserved().booleanValue() == false)){
      document.demoteVersion();
      document.save(RefreshMode.REFRESH);
      System.out.println("The document is demoted");
      showVersionsEDU(document);
      showVersionSeriesEDU(document);
   }
}

public void promoteEDU(Document document)
{
if (document.get_IsCurrentVersion().booleanValue() == false)
      document = (Document)document.get_CurrentVersion();
   if ((document.get_VersionStatus().getValue() ==
   VersionStatus.IN_PROCESS_AS_INT) &&
   (document.get_IsReserved().booleanValue() == false))
   {
      document.promoteVersion();
      document.save(RefreshMode.REFRESH);
      System.out.println("The document is promoted");
      showVersionsEDU(document);
      showVersionSeriesEDU(document);
   }
}

public Document getDocumentEDU(ObjectStore store, String folderPath){
      PropertyFilter pf = new PropertyFilter();
```

```
      pf.addIncludeProperty(0, null, null, PropertyNames.NAME,1);
      pf.addIncludeProperty(0, null, null,
      PropertyNames.MINOR_VERSION_NUMBER,1);
      pf.addIncludeProperty(0, null, null,
      PropertyNames.MAJOR_VERSION_NUMBER,1);
      pf.addIncludeProperty(0, null, null, PropertyNames.VERSION_SERIES,1);
      pf.addIncludeProperty(0, null, null,
      PropertyNames.IS_CURRENT_VERSION,1);
      pf.addIncludeProperty(0, null, null, PropertyNames.IS_RESERVED,1);
      pf.addIncludeProperty(0, null, null, PropertyNames.RESERVATION,1);
      pf.addIncludeProperty(0, null, null, PropertyNames.VERSIONS,1);
      pf.addIncludeProperty(0, null, null, PropertyNames.VERSION_STATUS,1);
      pf.addIncludeProperty(0, null, null, PropertyNames.CURRENT_VERSION,1);
      Document document = Factory.Document.fetchInstance(store,folderPath,
      pf);
      String documentName = document.get_Name();
      System.out.println(documentName + "  Document is retrieved");
      return document;
}
public static void main(String[] args) {
   try {
      VersionsEDU myInstance = new VersionsEDU();
      Connection conn =
      myInstance.getCEConnectionEDU("p8admin","IBMFileNetP8");
      Domain domain = myInstance.getDomainEDU(conn);
      ObjectStore store = myInstance.getObjectStoreEDU
      (domain,"Development");
      Document document = myInstance.getDocumentEDU
      (store,"/APIFolder/APIVersions.doc");
      //myInstance.showVersionsEDU(document);
      //myInstance.showVersionSeriesEDU(document);
      // myInstance.checkoutEDU(document);
      // myInstance.checkinEDU(document);
      // myInstance.checkoutEDU(document);
      //myInstance.cancelCheckoutEDU(document);
      myInstance.promoteEDU(document);
      //myInstance.demoteEDU(document);
   }
   catch (Exception e)
   {
      e.printStackTrace();
   }
}
}
```


Unit 9. Security

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class in this unit extends the CEConnectionEDU class from the “Communication with the Content Engine” unit.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. (These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.)

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 9.1. Retrieve object permissions

Overview

Why is this lesson important to you?

Administrators want to be able to see what security permissions a particular folder or a document has without leaving the application. As the programmer, you are going to add this functionality to the application.

Activities

Retrieve object permissions: Challenge, page 9-7
Retrieve object permissions: Walkthrough, page 9-9

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Retrieve object permissions: Challenge

Challenge

Use the data in the table to create a Java class that extends CEConnectionEDU and write code to retrieve permissions for a folder or document and display the values.

Data

Item	Value

Packages to import	`java.util.Iterator`
	`com.filenet.api.collection.*`
	`com.filenet.api.constants.*`
	`com.filenet.api.core.*`
	`com.filenet.api.property.*`
	`com.filenet.api.security.*`
	`com.filenet.api.util.*`

Object store name	`"Development"`
Path for the folder	`"/APIFolder"`
Folder path for the document	`"/Research/Research Information Basic Model A.doc"`

Verification

- Run the program using Eclipse or the Command Prompt window.
- Verify that a list of access permissions with the values for grantee name, grantee type, permission source, access level, access type, and inheritable depth is displayed.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
API Folder is retrieved
Grantee name=CN=Managers, cn=groups, o=sample
Grantee type=GROUP
Permission source=SOURCE_DIRECT
Access level = FULL_CONTROL
Access type =ALLOW
Inheritable depth=-1
----------
Grantee name=#AUTHENTICATED-USERS
Grantee type=GROUP
Permission source=SOURCE_DEFAULT
Access level = WRITE_FOLDER
Access type =ALLOW
Inheritable depth=0
----------
Grantee name=cn=P8Admin,o=sample
Grantee type=USER
Permission source=SOURCE_DEFAULT
Access level = FULL_CONTROL
Access type =ALLOW
Inheritable depth=0
```

Note

The list of permissions that you get is much longer than what is shown here.

Retrieve object permissions: Walkthrough

Procedures

Procedure 1: Create a Java class that extends CEConnectionEDU

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the “New Java Class” window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type the class name `SecurityEDU` in the Name field.
5. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
6. Select `public static void main(String[] args)` for “Which method stubs would you like to create?” option.
   a. Clear other options.
7. Click Finish to add the class and close the window.
   a. Verify the new class listed under your package in the Project Explorer.
8. Write code inside the class that you just added.
9. Import the following packages before the declaration of the class:

```
java.util.Iterator
com.filenet.api.collection.*
com.filenet.api.constants.*
com.filenet.api.core.*
com.filenet.api.property.*
com.filenet.api.security.*
com.filenet.api.util.*
```

Procedure 2: Write a method to retrieve folder permissions

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getPermissionsEDU`
   - Parameter: `Containable` object

Note

The `Folder` object is used in this activity for the Containable object.

You can replace it with `Customobject` or `Document` object to retrieve those permissions.

2. Get the folder permissions by calling the `Folder.get_Permissions()`.
   a. Assign the return value to a variable of type `AccessPermissionList`.
3. Call `accessPermissionList.iterator()`.
   a. Assign the returned value to a variable of the type `Iterator`.
4. Write a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```

5. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into an `AccessPermission` type and assign it to a variable.
6. Retrieve and display the name of grantee for each permission by calling `System.out.println(…)`.
   - Parameter: `accessPermission.get_GranteeName()`
7. Repeat step 6 to retrieve and display the grantee type, permission source, access level, access type, and inheritable depth for each permission:
   a. Use each of the following parameters:
   - `accessPermission.get_GranteeType().toString()`
   - `accessPermission.get_PermissionSource().toString()`
   - `accessLevel.getInstanceFromInt(accessPermission.get_AccessMask().intValue()).toString())`
   - `accessPermission.get_AccessType().toString()`
   - `accessPermission.get_InheritableDepth()`

Procedure 3: Instantiate the class and call the methods that you wrote

1. Inside the `try` block of the main() method, create an instance of the class.

   Example: `SecurityEDU securityInstance = new SecurityEDU();`

2. Call the getCEConnectionEDU(...) method of the CEConnectionEDU class:
   - Parameters:
     - "p8admin"
     - "IBMFileNetP8"

   a. Assign the returned Connection object to a variable.
3. Call the getDomainEDU(…)method of the CEConnectionEDU class.
   - Parameter: Connection object variable from step 2

   a. Assign the returned Domain object to a variable.
4. Call the getObjectStoreEDU(…) method of the CEConnectionEDU class:
   - Parameters:
     - Domain object variable from step 3
     - "Development"

   a. Assign the return value to a variable of type ObjectStore.
5. Call the getFolderEDU(...) method of the CEConnectionEDU class:
   - Parameters:
     - Objectstore variable from step 4
     - "/APIFolder"

   a. Assign the return value to a variable of type Folder..
6. Call the getPermissionsEDU(...) method.
   - Parameter: Folder variable that you got in step 5

Procedure 4: Run the program and verify the results

1. Run the program using the Eclipse or the Command Prompt as instructed in the "Communication with the Content Engine Server" unit.
2. Verify that a list of access permissions with the values for grantee name, grantee type, permission source, access level, access type, and inheritable depth is displayed in the Output window.

Sample output

Refer to the sample output in Retrieve object permissions: Challenge, page 9-7

Lesson 9.2. Set object permissions

Overview

Why is this lesson important to you?

A number of older documents in the object store do not have the desired security settings. As the programmer, you are going to write code to update the security permissions on these documents so that they are the same as the folder they are filed in.

Prerequisites

Activity , Set up the folder and document, page 9-15

Activities

Set object permissions: Challenge, page 9-17
Set object permissions: Walkthrough, page 9-19

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Set up the folder and document

Important

This activity is required for both skill levels.

1. Sign in to the Content Engine Enterprise Manager as p8admin.
2. Expand Object Stores > Development > Root Folder.
3. Select APIFolder, right-click, and click Properties.
4. In the APIFolder Properties window, click the Security tab and select the `Managers` security group from the list in the Name column.
   a. Ensure that the value “This object and all children” is selected from the Apply To list.
   b. Set the access level is View Properties in the Level area.
   c. Click OK to save the changes and close the window.
5. In the `APIFolder`, create a document with no content, use the default values, and name it `APISecurityDoc`.
6. Open the Document Properties page.
   a. On the General tab of the APISecurityDoc Properties window, verify that the “Inherit Security from folder” option is **not** selected.

      This option will be set when you execute the code in the next activity.
   b. On the Security tab, note that the document does not have the `Managers` security group. You programmatically set this group in the following activity.
   c. Click OK to close the window.

Set object permissions: Challenge

Challenge

Use the data in the table and the same Java class that you created in the previous lesson and write code to do the following:

- Set permissions on a folder
- Set security of the parent folder to a document.

Data

Item	Value
Object store name	`"Development"`
Folder name and path	`"/APIFolder"`
Folder path for the document	`"/APIFolder/APISecurityDoc"`

Access permission property	Value to set
`GranteeName`	`"Managers"`
`AccessType`	`AccessType.ALLOW`
`InheritableDepth`	`-1`
`AccessMask`	`AccessLevel.FULL_CONTROL_FOLDER_AS_INT`

Verification (Permissions on a folder)

- Run the program using Eclipse or the Command Prompt window.
- Use Workplace or Content Engine Enterprise Manager to verify that the security group Managers has been updated with the values that you set (Owner Control) for the folder in the previous activity.
- Optionally, call the method that you wrote in the previous lesson to retrieve folder permissions and display the values.

Verification (Permissions on a document)

- Use Workplace or Content Engine Enterprise Manager to verify the following:
  - The security group Managers with values that you set for the folder has been added to the document security.
  - On the General tab of the APISecurityDoc document Properties window, verify that the “Inherit Security from folder” option is selected.
- Optionally, call the method that you wrote in the previous lesson to retrieve document permissions and display the values.

Sample output

```
Name of the domain: P8Domain
Name of the objectstore: Development
APIFolder is retrieved.
Folder security set.
APISecurityDoc document is retrieved
parent folder name: APIFOlder
Document security is set
```


Set object permissions: Walkthrough

Procedures: Set permissions on a folder

Procedures: Set the security folder to a document

Set permissions on a folder

Use the same Java class that you created in the previous lesson.

Procedure 1: Write a method to set permissions on a folder

1. Define a method using the following signature:
   - Scope: `public`
   - Name: `setFolderSecurityEDU`
   - Parameter: `Folder` object
2. Call `Factory.AccessPermission.createInstance()`.
   a. Assign the return value to a variable of type `AccessPermission`.
3. Set the values for the properties of access permission.
   a. Call `accessPermission.set_GranteeName(…)`.
      - Parameter: `"Managers"`
   b. Call `accessPermission.set_AccessType(…)`.
      - Parameter: `AccessType.ALLOW`
   c. Call `accessPermission.set_InheritableDepth(…)`.
      - Parameter: `new Integer(-1)`
   d. Call `accessPermission.set_AccessMask(…)`.
      - Parameter: `new Integer(AccessLevel.FULL_CONTROL_FOLDER_AS_INT)`

4. Retrieve the permissions list for the given folder by calling `folder.get_Permissions()`
   a. Assign the return value to a variable of type `AccessPermissionList`.
5. Call `accessPermissionList.add(…)`on the object from step 4.
   - Parameter: `AccessPermission` object from step 2
6. Set the permissions to the folder by calling `folder.set_Permissions(…)`.
   - Parameter: `AccessPermissionList` object from step 4
7. Save the changes by calling `folder.save(…)`.
   - Parameter: `RefreshMode.REFRESH`

Procedure 2: Call the method inside the main() method

1. Call the `setFolderSecurityEDU(…)` method inside the `try` block of the main() method.
   - Parameter: `Folder` variable that you got in the previous lesson
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt window.
2. Verify the results:
   a. Sign in to Workplace as p8admin.
   b. Browse to the folder on which you set the security: `Development` > `APIFolder`.
   c. Open the Properties page by clicking the Get Info icon at the top of the page.
   d. Click the Security link in the left pane.
   e. Verify that the security group Managers has been updated with the values that you set (Owner Control).
   f. Notice the folder inheritance icon. Hover over it and notice the text that is displayed: `This folder and all levels below`.
3. Optionally, you can call the method `getPermissionsEDU(Folder)` again to display the security information programmatically.

Sample output

Refer to the sample output in Retrieve object permissions: Challenge, page 9-7

Set the security folder to a document

Use the same Java class that you created in the previous lesson.

Procedure 1: Write a method to retrieve a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getDocumentEDU`
   - Parameters:
     - `ObjectStore` object
     - `String` for the folder path
   - Returns: `Document` object
2. Create a new instance a `PropertyFilter` object.
   a. Assign the return value to a variable of type `PropertyFilter` object.
3. Call `PropertyFilter.addIncludeProperty(...)`.
   - Parameters:
     - `0`
     - `null`
     - `null`
     - `PropertyNames.PERMISSIONS`
     - `1`
4. Repeat step 3 to include the following properties to the property filter:
   - `PropertyNames.CONTAINERS`
   - `PropertyNames.NAME`
   - `"style"`
5. Call `Factory.Document.fetchInstance(...)`.
   - Parameters:
     - `Objectstore` object
     - Folder path that are passed into this method
     - `PropertyFilter` object that you got in step 2
6. Get and display the document name by calling `System.out.println(…)`.
   - Parameter: `document.get_Name()`
7. Return the `Document` object.

Procedure 2: Write a method to set the security folder to a document

1. Define a method using the following signature:
   - Scope: `public`
   - Name: `setSecurityFolderToDocumentEDU`
   - Parameters:
     - `Document` object
     - `Folder` object
2. Retrieve the containment relationship objects for the folders in which the given document is filed by calling the `document.get_Containers()`.
   a. Assign the return value to a variable of type `ReferentialContainmentRelationshipSet`.
3. Call `referentialContainmentRelationshipSet.iterator()`.
   a. Assign the returned value to a variable of the type `Iterator`.
4. Write a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```

5. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into a `ReferentialContainmentRelationship` type and assign it to a variable.
6. Use an `if` statement to determine whether the object retrieved is same as the folder that you are interested in:

   ```
   if(referentialContainmentRelationship.get_Tail().equals(folder))
   ```

7. Inside the `if` statement, retrieve the folder by calling `referentialContainmentRelationship.get_Tail()`.
   a. Typecast the returned object into a `Folder` type and assign in to a variable.
   b. Optionally, display the folder name by calling `System.out.println(…)`.
      - Parameter: `folder.get_FolderName()`
   c. Call `document.set_SecurityFolder(...)` on the `Document` object that is passed into this method to set the security folder.
      - Parameter: `Folder` object from step 7
   d. Save the changes to the document by calling `document.save(…)`.
      - Parameter: `RefreshMode.REFRESH`
   e. Close the `if` statement.
8. Close the method.

Procedure 3: Call the method inside the main() method

1. Call the `getDocumentEDU(...)`method inside the `try` block of the main() method.
   - Parameters:
     - `ObjectStore` variable that you got in the previous lesson.
     - `"/APIFolder/APISecurityDoc"`

   a. Assign the return value to a variable of type `Document`.
2. Call the `setSecurityFolderToDocumentEDU(...)` method.
   - Parameter:
     - `Document` object from step 1
     - Folder object that you got in the previous activity
3. Comment out the method calls that are not tested in this activity.

Procedure 4: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt window.
2. Verify that the document has the Managers security group:
   a. Sign in to Workplace as p8admin.
   b. Browse to the document to which you set the security folder.
   c. Open the Properties page by clicking the Info icon next to the document name.
   d. Click the Security link in the left pane.
   e. Verify that the security group Managers with values that you set for the folder in the previous activity has been added to the document security.
3. Use Content Engine Enterprise Manager to verify the following:
   a. On the General tab of the APISecurityDoc document Properties window, the “Inherit Security from folder” option is selected.
4. Optionally, you can call the `getPermissionsEDU(Document)`method that you wrote in the previous lesson to display the security information programmatically.

Sample output

Refer to the sample output in Set object permissions: Challenge, page 9-17

Lesson 9.3. Apply a security template to an object

Overview

Why is this lesson important to you?

New security measures are being put in place for the application. Access to the documents is restricted to certain users based on the value of a document property. You are going to write code to apply a security template to a document. The code checks the value of a document property and selects the correct template for that document. As a result, the document can be accessed only by the designated users.

Prerequisites

Activity , Create a security policy, page 9-27
Activity , Create documents, page 9-29

Activities

Apply a security template to an object: Challenge, page 9-31
Apply a security template to an object: Walkthrough, page 9-35

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Create a security policy

Important

This activity is required for both skill levels.

In this procedure, you create a new security policy with two application security templates. You apply one template or the other, based on a condition specified in the following activity.

1. In Content Engine Enterprise Manager, go to Object Stores > Development > Security Policies.
2. Right-click Security Policies and click New Security Policy.

   The Create a Security Policy wizard starts.

   a. Type the following name for the security policy: `APISecurityPolicy`
3. Click Next to open the “Specify the security templates” window.
   a. Click Add to add a new security template.
   b. In the “New Custom Template” area of the “Add Templates” window, type the new template name `APISecurityTemplateBasic` and click Add.
   c. Verify that the new template is listed in the Templates area and ensure that the `APISecurityTemplateBasic` check box is selected.
   d. Repeat steps 3a through 3c to add a second security template with the name `APISecurityTemplateDeluxe`.
   e. Click OK to close the Add Templates window to return to the “Specify the security templates” window.
4. Modify the templates to add the specific permission (ACE).
   a. Select `APISecurityTemplateBasic` from the Security Templates area.
   b. Click Modify.
   c. Click the Template Security tab.
   d. Click Add to include the permission for the user name `adam`.
   e. In the “Search Criteria” area of Select Users and Groups window, ensure that the Short Name option is selected.
   f. In the text box, type the first few letters of the name of the user to search for, which is specified in the data table, and click Find.
   g. Select the user from the returned list and click OK.
   h. Set the permissions for that user with the values specified in the data table.

i. Use the following data to set permissions for the `APISecurityTemplateBasic` security template.

Item	Value
User	`adam`
Type	`Deny`
Level	`Full Control`
Apply To	`This object and all children`

5. Determine the GUID of the new application security template:
   a. Click the General tab of the Properties page and write down the first several digits of the GUID listed in the ID field.

      _____

      Example: {D8EBD79F-68BE-4599-90B1-52FAF044E718}

   b. Use this value to verify the security template that has been applied in the next activity.
6. Repeat steps 3 and 4 for the `APISecurityTemplateDeluxe` security template.
7. Use the following data to set permissions for the security template.

Item	Value
User	`charles`
Type	`Deny`
Level	`Full Control`
Apply To	`This object and all children`

8. Complete the Create a Security Policy wizard.
   a. Click OK to close the Properties page.
   b. Click Next to accept the new security template.
   c. Verify that the information on the last wizard screen is correct, and then click Finish to close the wizard.
9. Click OK when the confirmation message is displayed: `You have successfully created 'APISecurityPolicy' SecurityPolicy .`

Create documents

Important

This activity is required for both skill levels.

1. Use Content Engine Enterprise Manager to create two documents with content.
2. Use the following data to create the first document.

Item	Value
Object store name	`"Development"`
Document class	`Product`
Folder to file the document in	`APIFolder`
Document title	`APIBasic.doc`
Value for the `style` property	`Basic`
Location of the source document	`C:\CMJavaAPIProg\SampleDocuments\APIBasic.doc`

3. Use the following data to create the second document.

Item	Value
Object store name	`"Development"`
Document class	`Product`
Folder to file the document in	`APIFolder`
Document title	`APIDeluxe.jpg`
Value for the `style` property	`Deluxe (Default value set is Basic)`
Location of the source document	`C:\CMJavaAPIProg\SampleDocuments\APIDeluxe.jpg`

Apply a security template to an object: Challenge

Challenge

- Use the same Java class that you created in the previous lesson.
- Write code to apply each one of the security templates of the security policy that you created to the given Document class based on the conditions in the following table.
- Optionally, display the values for the security policy name, security templates names, documents names, and their property values.

Data

Item	Value
Document Class	`Product`
Folder path	`"/APIFolder”`

Condition

If value of the document property “style” is	Apply this Security template
`Basic`	`APISecurityTemplateBasic`
`Deluxe`	`APISecurityTemplateDeluxe`

Verification for the security policy

- Run the program using Eclipse or the Command Prompt.
- Use the Content Engine Enterprise Manager and verify that the security policy has been set to the document:
  - On the document Properties page, click the Security Policy tab.
  - Verify that Assigned Security Policy has a value (the name of the security policy you created) in the list.
  - Verify that “Security Templates” has values in the box.

Verification for the security template

- Verify that the security template has been applied to the document:
  - On the document Properties page, click the Security tab.
  - The permissions in the security template have been added to the list of permissions on the document.
  - Ensure that one additional user (the user you set in the previous activity) is listed as the last item under the Name column, with template as the source.
  - Verify that the security policy name, security templates names, documents names, and their property values are displayed in the Output window.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIFolder  folder is retrieved
Security policy = APISecurityPolicy
===================================
Document name: New Model
value for 'style' property: Basic
Security template name: APISecurityTemplateBasic
Security template Id: {E22EE167-C6FF-4CD2-AA98-B5D29BF80B57}
Apply state Id: {D8EBD79F-68BE-4599-90B1-52FAF044E718}
===================================
Document name: APIDeluxe.jpg
value for 'style' property: Deluxe
Security template name: APISecurityTemplateDeluxe
Security template Id: {B5F84566-2ACB-431F-B897-065EB299941A}
Apply state Id: {7E5BBA98-F11E-49FE-BE6F-EAA4F9762365}
===================================
Document name: APIOrderDoc
value for 'style' property: Basic
Security template name: APISecurityTemplateBasic
Security template Id: {E22EE167-C6FF-4CD2-AA98-B5D29BF80B57}
Apply state Id: {D8EBD79F-68BE-4599-90B1-52FAF044E718}
===================================
Document name: APIBasic.doc
value for 'style' property: Basic
Security template name: APISecurityTemplateBasic
Security template Id: {E22EE167-C6FF-4CD2-AA98-B5D29BF80B57}
Apply state Id: {D8EBD79F-68BE-4599-90B1-52FAF044E718}
===================================
```


Note

The sample output contains more documents than the two that you worked on because the folder has other documents that have a “Style” property with the same value. The code updates the security for other documents with the identical conditions.

Apply a security template to an object: Walkthrough

In this procedure, you apply the security template to a document based on the value for a given property.

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to set security policy

1. Define a method using the following signature:
   - Scope: `public`
   - Name: `setSecurityPolicyEDU`
   - Parameters:
     - `ObjectStore` object
     - `Folder` object
     - `String` for security policy name
     - `String` for security template1 name
     - `String` for security template2 name
2. Retrieve the document collection available in the given folder by calling `folder.get_ContainedDocuments()`.
   a. Assign the return value to a variable of type `DocumentSet`.

Procedure 2: Retrieve security policies from an object store

Continue with the same method from procedure 1.

1. Retrieve the security policies collection available in the object store by calling the `0bjectStore.get_SecurityPolicies()`.
   a. Assign the return value to a variable of type `SecurityPolicySet`.
2. Call `securityPolicySet.iterator()`.
   a. Assign the returned value to a variable of the type `Iterator`.

3. Write a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```

4. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into a `SecurityPolicy` type and assign it to a variable.
   b. Retrieve and display the security policy name by calling `System.out.println(...)`.
      - Parameter: `securityPolicy.get_Name()`
   c. Inside the `while` loop, write an `if` statement and call the `securityPolicy.get_Name()` method to check for the policy name that you are interested in.

      You are going to assign this policy to the Document objects in the following steps.

Procedure 3: Retrieve the documents and set the security policy

Continue with the same method from procedure 1.

1. Inside the `if` statement, retrieve the security template collection available for this policy by calling `securityPolicy.get_SecurityTemplates()`.
   a. Assign the return value to a variable of type `SecurityTemplateList`.
2. Call `documentSet.iterator()` on the object from procedure 1, step 2.
   a. Assign the returned value to a variable of the type `Iterator`.
3. Write a second `while` loop to retrieve each document from the collection.

   ```
   while (iterator.hasNext())
   ```

4. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into a `Document` type and assign it to a variable.
5. Write a second `if` statement and call the method `document.get_ClassDescription().describedIsOfClass("Product")` to determine if the Document class is `Product`.
6. Inside the `if` statement, retrieve the document properties collection from `document.get_Properties`().
   a. Assign the return value to a variable of type `Properties`.
7. Assign the security policy to the document by calling `properties.putValue(...)`.
   - Parameters:
     - `PropertyNames.SECURITY_POLICY`
     - security policy from step 4a in procedure 2
8. Retrieve the value for the style property by calling `properties.getStringValue(...)`.
   - Parameter: `"style"`
   a. Assign the return value to a variable of type `String`.

9. Retrieve and display the document name by calling `System.out.println(...)`
   - Parameter: `document.get_Name()`

Procedure 4: Retrieve security templates and apply to documents

Continue with the same method from procedure 1.

1. Call `securityTemplateList.iterator()`on the object from procedure 3, step 1.
   a. Assign the returned value to a variable of the type `Iterator`.
2. Write a third `while` loop to retrieve each security template from the collection.

   ```
   while (iterator.hasNext())
   ```
3. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into a `SecurityTemplate` type and assign it to a variable.
4. Write a third `if` statement to check the following conditions:
   - The template has the same name as the template1 that is passed into this method.
   - The property value that you retrieved in procedure 3, step 8 is "`Basic`".
5. Inside the `if` statement, display the security template name by calling `System.out.println(…)`.
   - Parameter: `SecurityTemplate.get_DisplayName()`
6. Retrieve the Id for the security template by calling `securityTemplate.get_Id()`.
   a. Assign the return value to a variable of type `Id`.
   b. Convert the Id to a `String` and display the Id by calling `System.out.println(...)`.
7. Retrieve the apply state Id by calling `securityTemplate.get_ApplyStateID` ().
   a. Assign the return value to a variable of type `Id`.
   b. Convert the Id to a String and display the Id by calling `System.out.println(...)`.
   c. Call `document.applySecurityTemplate(...)`to apply the template.
      - Parameter: `Id` that you got in step 7
8. Save the changes to the document object by calling `document.save(...)`.
   - Parameter: `RefreshMode.REFRESH`
9. Write a fourth `if` statement to check the following conditions:
   - The template has the same name as the template2 that is passed into this method.
   - The property that you retrieved in procedure 3, step 8 is "`Deluxe`".

   a. Repeat steps 5 through 8.
   b. Close the `if` statement.

10. Close the loops and the method.
    a. Close the third `while` loop.
    b. Close the second `if` statement.
    c. Close the second `while` loop.
    d. Close the first `if` statement loop.
    e. Close the first `while` loop.
    f. Close the method.

Procedure 5: Call the method inside the main() method

1. Call `setSecurityPolicyEDU(...)`method inside the `try` block of the main() method.
   - Parameters:
     - `ObjectStore` variable that you got in a previous activity
     - `APIFolder` object that you got in the previous lesson
     - `"APISecurityPolicy"`
     - `"APISecurityTemplateBasic"`
     - `"APISecurityTemplateDeluxe"`
2. Comment out the method calls that are not tested in this activity.

Procedure 6: Run the program and verify the results

1. Use the verification steps in Apply a security template to an object: Challenge, page 9-31

Sample output

Refer to the sample output in Apply a security template to an object: Challenge, page 9-31

Lesson 9.4. Retrieve security users and groups

Overview

Why is this lesson important to you?

The administrators want to be able to find all user names and group names that match a particular pattern, without leaving the application. As their programmer, you are going to add this functionality to the application.

Activities

Retrieve security users and groups: Challenge, page 9-41
Retrieve security users and groups: Walkthrough, page 9-43

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Retrieve security users and groups: Challenge

Challenge

Use the same Java class that you created in the previous lesson. Use the data in the table to write code to retrieve Realm object for the IBM FileNet P8 domain and to retrieve security users and groups.

Data

Item	Value
Search pattern (Names of the users or groups starting with the letter)	"m"

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that a list of security user names and group names is displayed in the Command prompt window or in the console of Eclipse.

Sample output for users

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIFolder  folder is retrieved
o=sample  Realm is retrieved
List of Security Users
-------------
Distinguished name =CN=mabel,cn=users,o=sample
Short name =mabel
Name =CN=mabel,cn=users,o=sample
Display name =mabel
-------------
Distinguished name =CN=mac,cn=users,o=sample
Short name =mac
Name =CN=mac,cn=users,o=sample
Display name =mac
-------------
```

```
  Distinguished name =CN=manny,cn=users,o=sample
  Short name =manny
  Name =CN=manny,cn=users,o=sample
  Display name =manny
  -------------
  Distinguished name =CN=mark,cn=users,o=sample
  Short name =mark
  Name =CN=mark,cn=users,o=sample
  Display name =mark
  -------------
  Distinguished name =CN=mary,cn=users,o=sample
  Short name =mary
  Name =CN=mary,cn=users,o=sample
  Display name =mary
  -------------
  Distinguished name =CN=matt,cn=users,o=sample
  Short name =matt
  Name =CN=matt,cn=users,o=sample
  Display name =matt
  -------------
  Distinguished name =CN=may,cn=users,o=sample
  Short name =may
  Name =CN=may,cn=users,o=sample
  Display name =may
  -------------
==========================
```

Sample output for groups

```
  List of Security Groups
  -------------
  Distinguished name = CN=Managers,cn=groups,o=sample
  short name =Managers
  Name =CN=Managers,cn=groups,o=sample
  Display name =Managers
  -------------
```


Retrieve security users and groups: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to retrieve Realm object for the IBM FileNet P8 domain

1. Define a method using the following signature:
   - Scope: `public`
   - Name: `getRealmEDU`
   - Parameter: `Connection` object
   - Returns: `Realm` Object
2. Create a new instance a `PropertyFilter` object.
   a. Assign the return value to a variable of type `PropertyFilter` object.
3. Call `PropertyFilter.addIncludeProperty(...)`.
   - Parameters:
     - `0`
     - `null`
     - `null`
     - `PropertyNames.MY_REALM`
     - `1`
4. Call `Factory.EntireNetwork.fetchInstance(...)` to retrieve the entire network object.
   - Parameters:
     - `Connection` object that is passed into this method
     - `PropertyFilter` object from step 2

   a. Assign the return value to a variable of type `EntireNetwork`.
5. Retrieve realm object by calling `entireNetwork.get_MyRealm()`.
   a. Assign the return value to a variable of type `Realm`.

6. Retrieve and display the name of the realm by calling `System.out.println(…)`.
   - Parameter: `realm.get_Name()`
7. Return the `Realm` object from step 5.

Procedure 2: Write a method to retrieve and display the list of users and groups

1. Define a method using the following signature:
   - Scope: `public`
   - Name: `showUserGroupsEDU`
   - Parameters:
     - `Realm` object
     - `String` for a pattern to search
2. Call `realm.findUsers(...)`.
   - Parameters:
     - `String` for a pattern to search
     - `PrincipalSearchType.PREFIX_MATCH`
     - `PrincipalSearchAttribute.SHORT_NAME`
     - `PrincipalSearchSortType.NONE`
     - `Integer.valueOf("50")`
     - `null`

   a. Assign the return value to a variable of type `UserSet`.
3. Call `userSet.iterator()`.

   a. Assign the returned value to a variable of the type `Iterator`.
4. Write a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```
5. Call `iterator.next()`within the `while` loop.

   a. Typecast the return value into a `User` type and assign it to a variable.

   b. Display distinguished name for each user by calling `System.out.println(…)`.
      - Parameter: `user.get_DistinguishedName()`
6. Repeat step 5 for user short name, name, and display name using the following parameters, respectively:
   - `user.get_ShortName()` for the short name
   - `user.get_Name()` for the name
   - `user.get_DisplayName()` for the display name
7. Repeat step 2 by calling `realm.findGroups(...)`with the same parameters to retrieve the groups.

   a. Assign the return value to a variable of type `GroupSet`.

b. Replace `User` object with `Group` object and repeat steps 3 through 6.

8. Close the method.

Procedure 3: Call the method inside the main() method

1. Call the `getRealmEDU(...)` method inside the `try` block of the main() method.
   - Parameter: `Connection` object from previous unit.
   a. Assign the return value to a variable of type `Realm`.
2. Call the `showUserGroupsEDU(...)` method.
   - Parameters:
     - `Realm` object that you got in step 1
     - `"m"` for the search pattern
3. Comment out the method calls that are not tested in this activity.

Procedure 4: Run the program and verify the results

1. Run the program and verify that a list of user names and group names is displayed in the output window.

Sample output

Refer to the sample output in Retrieve security users and groups: Challenge, page 9-41

Solution code for SecurityEDU.java

```
package com.ibm.filenet.edu;

import java.util.Iterator;
import com.filenet.api.collection.*;
import com.filenet.api.constants.*;
import com.filenet.api.core.*;
import com.filenet.api.property.*;
import com.filenet.api.security.*;
import com.filenet.api.util.*;

public class SecurityEDU extends CEConnectionEDU {

public void getPermissionsEDU(Containable CEObject)
{
   AccessPermissionList permissions = CEObject.get_Permissions();
   Iterator it = permissions.iterator();
   while (it.hasNext())
   {
      AccessPermission permission = (AccessPermission)it.next();
      System.out.println("Grantee name=" + permission.get_GranteeName());
      System.out.println("Grantee type=" +
      permission.get_GranteeType().toString());
      System.out.println("Permission source=" +
      permission.get_PermissionSource().toString());
      System.out.println("Access level = " +
      AccessLevel.getInstanceFromInt(permission.get_AccessMask().intValue()
      .toString());
      System.out.println("Access type =" +
      permission.get_AccessType().toString());
      System.out.println("Inheritable depth=" +
      permission.get_InheritableDepth());
      System.out.println("----------");
   }
}
Public void setFolderSecurityEDU(Folder folder)
{
   AccessPermission permission = Factory.AccessPermission.createInstance();
   permission.set_GranteeName("Managers");
   permission.set_AccessType(AccessType.ALLOW);
   permission.set_InheritableDepth(new Integer(-1)); // all children
   permission.set_AccessMask(new
   Integer(AccessLevel.FULL_CONTROL_FOLDER_AS_INT));
```



```
   AccessPermissionList permissions = folder.get_Permissions();
   Permissions.add(permission);
   Folder.set_Permissions(permissions);
   Folder.save(RefreshMode.REFRESH);
   System.out.println("Folder security is set");
}
public Document getDocumentEDU(ObjectStore store, String folderPath)
{
   PropertyFilter pf = new PropertyFilter();
   pf.addIncludeProperty(0, null, null, PropertyNames.PERMISSIONS,1);
   pf.addIncludeProperty(0, null, null, PropertyNames.CONTAINERS,1);
   pf.addIncludeProperty(0, null, null, PropertyNames.NAME,1);
   pf.addIncludeProperty(0, null, null, "style",1);
   Document document = Factory.Document.fetchInstance(store,folderPath, pf);
   String documentName = document.get_Name();
   System.out.println(documentName + "  Document is retrieved");
   return document;
}
public void setSecurityFolderToDocumentEDU(Folder folder, Document document)
{
   ReferentialContainmentRelationshipSet rels = document.get_Containers();
   ReferentialContainmentRelationship rel = null;
   Folder parentFolder =null;
   Iterator it = rels.iterator();
   while (it.hasNext()){
      rel = (ReferentialContainmentRelationship)it.next();
      if(rel.get_Tail().equals(folder)){
         parentFolder =(Folder) rel.get_Tail();
         System.out.println("Parent folder name: " +
         parentFolder.get_FolderName());
         document.set_SecurityFolder(parentFolder);
         document.save(RefreshMode.REFRESH);
         System.out.println("Document security is set");
      }
   }
}
```

```
public void setSecurityPolicyEDU(ObjectStore store, Folder folder, String
policyName, String templateName1, String templateName2)
{
   DocumentSet documents = folder.get_ContainedDocuments();
   Iterator documentIt = documents.iterator();
   Document document;
   SecurityPolicySet securityPolicies = store.get_SecurityPolicies();
   SecurityPolicy securityPolicy = null;
   Iterator policyIt = securityPolicies.iterator();
   while (policyIt.hasNext())
   {
      securityPolicy = (SecurityPolicy)policyIt.next();
      System.out.println("Security policy = " + securityPolicy.get_Name());
      if (securityPolicy.get_Name().equalsIgnoreCase(policyName)){
      SecurityTemplateList securityTemplates =
      securityPolicy.get_SecurityTemplates();
      System.out.println("====================================");
      while (documentIt.hasNext())
      {
         document = (Document)documentIt.next();
         if (document.get_ClassDescription().describedIsOfClass("Product"))
         {
         Properties properties = document.getProperties();
         properties.putValue(PropertyNames.SECURITY_POLICY,securityPolicy);
         System.out.println("Document name: " + document.get_Name());
         String docStyle = properties.getStringValue("style");
         System.out.println("value for 'style' property: " + docStyle);
         Iterator securityTemIt = securityTemplates.iterator();
         Id applyStateId;
         SecurityTemplate securityTemplate = null;
         while (securityTemIt.hasNext()) {
         securityTemplate = (SecurityTemplate)securityTemIt.next();
         if(docStyle.equalsIgnoreCase("Basic") &&
         securityTemplate.get_DisplayName().equalsIgnoreCase(templateName1))
         {
         System.out.println("Security template name: " +
         securityTemplate.get_DisplayName());
                                  System.out.println("Security template Id:
         " + securityTemplate.get_Id().toString());
         applyStateId = securityTemplate.get_ApplyStateID();
         System.out.println("Apply state Id: " + applyStateId.toString());
         document.applySecurityTemplate(applyStateId);
                                     document.save(RefreshMode.REFRESH);
         System.out.println("====================================");
                                  }// if(docStyle = =Basic)
```



```
        if(docStyle.equalsIgnoreCase("Deluxe")&&
        securityTemplate.get_DisplayName().equalsIgnoreCase(templateName2))
        {
        System.out.println("Security template name: " +
        securityTemplate.get_DisplayName());
        System.out.println("Security template Id: " +
        securityTemplate.get_Id().toString());
        applyStateId = securityTemplate.get_ApplyStateID();
        System.out.println("Apply state Id: " + applyStateId.toString());
        document.applySecurityTemplate(applyStateId);
                                document.save(RefreshMode.NO_REFRESH);
        System.out.println("====================================");
                        } // if (docStyle = Deluxe)
                    } //while (securityTemIt)
                } //if (document.ClassDescription)
            } // while (documentIt)
          }//if (securityPolicy.get_Name)
        } // while (policyIt.hasNext)
}

public Realm getRealmEDU(Connection conn){
      PropertyFilter pf = new PropertyFilter();
      pf.addIncludeProperty(0,null, null, PropertyNames.MY_REALM,1);
      EntireNetwork network = Factory.EntireNetwork.fetchInstance(conn, pf);
      Realm realm = network.get_MyRealm();
      String realmName = realm.get_Name();
      System.out.println(realmName + "  Realm is retrieved");
      return realm;
}

public void showUserGroupsEDU(Realm realm, String pattern)
{
UserSet users = realm.findUsers(pattern, PrincipalSearchType.PREFIX_MATCH,
PrincipalSearchAttribute.SHORT_NAME, PrincipalSearchSortType.NONE,
Integer.valueOf("50"), null);
com.filenet.api.security.User user;
Iterator it = users.iterator();
System.out.println("List of Security Users");
System.out.println("  -------------");
```

```
{
UserSet users = realm.findUsers(pattern, PrincipalSearchType.PREFIX_MATCH,
PrincipalSearchAttribute.SHORT_NAME, PrincipalSearchSortType.NONE,
Integer.valueOf("50"), null);
com.filenet.api.security.User user;
Iterator it = users.iterator();
System.out.println("List of Security Users");
System.out.println("  -------------");

while (it.hasNext())
   {
      user = (User)it.next();
      System.out.println("  Distinguished name =" +
      user.get_DistinguishedName());
      System.out.println("  Short name =" + user.get_ShortName());
      System.out.println("  Name =" + user.get_Name());
      System.out.println("  Display name =" + user.get_DisplayName());
      System.out.println("  -------------");
   }
System.out.println("==========================");
GroupSet groups = realm.findGroups(pattern,
PrincipalSearchType.PREFIX_MATCH, PrincipalSearchAttribute.SHORT_NAME,
PrincipalSearchSortType.NONE, Integer.valueOf("50"), null);
com.filenet.api.security.Group group;
Iterator groupIt = groups.iterator();
System.out.println("List of Security Groups");
while (groupIt.hasNext())
   {
      group = (Group)groupIt.next();
      System.out.println("  Distinguished name = " +
      group.get_DistinguishedName());
      System.out.println("  short name =" + group.get_ShortName());
      System.out.println("  Name =" + group.get_Name());
      System.out.println("  Display name =" + group.get_DisplayName());
      System.out.println("  -------------");
   }
}
```



```
public Folder getFolderEDU(ObjectStore store, String folderName){
      Folder folder= Factory.Folder.fetchInstance(store,folderName, null);
      folderName = folder.get_FolderName();
      System.out.println(folderName + "  folder is retrieved");
      return folder;
}

Public static void main(String[] args)
{
   try{
      SecurityEDU myInstance = new SecurityEDU();
      Connection conn =
      myInstance.getCEConnectionEDU("p8admin","IBMFileNetP8");
      Domain domain = myInstance.getDomainEDU(conn);
      ObjectStore store =
      myInstance.getObjectStoreEDU(domain,"Development");
      Folder folder = myInstance.getFolderEDU (store, "/APIFolder");
      //myInstance.getPermissionsEDU(folder);
      //myInstance.setFolderSecurityEDU(folder);
      //Document document =
      myInstance.getDocumentEDU(store,"/Research/Research Information Basic
      Model A.doc");
      //Document document =
      myInstance.getDocumentEDU(store,"/APIFolder/APISecurityDoc");
      //myInstance.setSecurityFolderToDocumentEDU(folder, document);
      //Realm realm = myInstance.getRealmEDU(conn);
      //myInstance.showUserGroupsEDU(realm, "m");
      myInstance.setSecurityPolicyEDU(store, folder, "APISecurityPolicy",
      "APISecurityTemplateBasic", "APISecurityTemplateDeluxe");
      System.out.println("Done");
   }
   Catch (Exception e) {
      e.printStackTrace();
   }
}
}
```


Unit 10. Events and Subscriptions

Unit overview

This unit contains the following lesson.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to a student machine configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- Some parts of the code from the “Document Lifecycle” unit are used.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. (These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.)

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 10.1. Implement a Java event handler

Overview

Why is this lesson important to you?

Your assignment is to automate the Content Engine actions for the following business scenarios:

- Log information when a document is created.
- Update an external database when a document is created.
- You are going to write code to implement a Java event handler and work with the system administrator to create event actions and subscriptions to automate these actions.

Activities

This lesson has only walkthrough-level activities.

- Implement a Java event handler: Walkthrough, page 10-7
- Update the event action with modified code: Walkthrough, page 10-13
- Error Handling: Walkthrough, page 10-15
- Optional: Update a database record: Walkthrough, page 10-19

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Implement a Java event handler: Walkthrough

Procedures

Procedure 1: Create a Java class that implements EventActionHandler

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the “New Java Class” window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type `LogEventActionEDU` as the class name in the Name field.
5. Click Finish to add the class and close the window.
   a. Verify that the new class is listed under your package in the Project Explorer.
6. Add to the class definition to implement `EventActionHandler` interface.
   a. Write code inside the class that you just added.
7. Import the following packages before the declaration of the class:
   - `java.io.*`
   - `com.filenet.api.core.*`
   - `com.filenet.api.engine.EventActionHandler`
   - `com.filenet.api.events.ObjectChangeEvent`
   - `com.filenet.api.exception.*`
   - `com.filenet.api.util.Id`
8. Add a custom method and implement one method as described in the following steps.

Procedure 2: Create a method to write data to a log file

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `writeToLogFileEDU`
   - Parameter: `String` for message to be written
   - Throws `IOException`
2. Create a new instance of File and assign it to a variable:

   ```
   File outputFile = new File(…);
   ```

   - Parameter: `"EventLog.txt"`
3. Create a new instance FileWriter and assign it to a variable:

   ```
   new FileWriter(...)
   ```

   - Parameters:
     - `File` object created in step 2
     - `true`
4. Call `fileWriter.write()` on the object that you got in step 3 to write to the file.
   - Parameter: `String` for message that is passed into this method
5. Call `fileWriter.close()`.
   a. Close the method.

Procedure 3: Implement the onEvent() Method

Use the same Java class that you created in procedure 1.

1. Implement the onEvent method using the following signature:
   - Scope: `public`
   - Name: onEvent
   - Parameters:
     - `ObjectChangeEvent` object
     - `Id` for subscriptionId
   - Throws `EngineRuntimeException`
2. Add `try-catch` blocks.
   a. Create a `catch` block for IOException and write the following inside the block:

   ```
   throw new EngineRuntimeException(ExceptionCode.E_FAILED, iOException);
   ```

   b. Write the following code in the `try` block.
3. Get the source object by calling `objectChangeEvent.get_SourceObject()` on the event object that is passed into this method.
   a. Typecast the returned value into a `Document` object and assign it to a variable.

4. Check to see if the event is a creation type:

   ```
   if (event.getClassName().equalsIgnoreCase("CreationEvent"))
   ```

   a. Write the code inside the `if` block.

5. Call `doc.getClassName()` to retrieve the document class name.

   a. Assign the return value to a `String`.

6. Call `doc.get_Name()` to retrieve the document name.

   a. Assign the return value to a `String`.

7. Call the `WriteToLogFileEDU(…)` method that you wrote earlier.

   - Parameter: `"A new document is created on: ” + new java.util.Date() + "\r\n"`

   a. Repeat step 7 to write the document name and class name into the log file using the String values you for in steps 5 and 6.

Procedure 4: Create a subscription on a Document class

Use the following instructions to create a subscription for the Product document class to be triggered when a document of that class is created.

In this process, you are also going to create a new event action that uses the Java code module that you wrote to create an entry in a log file whenever a document of this class is created.

1. Start the Content Engine Enterprise Manager and expand the Object Stores > Development > Document Class > Product.

   a. Right-click the Product document class and click Add Subscription.

      The Create a Subscription wizard starts.

2. Click Next.

3. In the Name and Describe the Subscription window, type `APILogSubscription` in the Name field and click Next.

4. On the Specify the Type of Object window, accept the default “Applies to all instances of the class Product” and click Next.

5. On the Specify Triggers window, select Creation Event to be the trigger:

   a. Select Creation Event in the Available Events list.

   b. Click Add to move Creation Event over to the Subscribed Events list.

   c. Click Next.

   d. Continue the wizard as described in the procedure 5.

Procedure 5: Create event actions for the subscription

1. On the Specify the Event Action window, click New to start the Create an Event Action wizard and click Next.
   a. Type `APILogEventAction` as the name for the event action.
   b. Click Next.
2. On the Specify the Type of Event Action window, identify the Java class to be used for the event action:
   a. Type `com.ibm.filenet.edu.LogEventActionEDU` as the Event Action Handler Java Class Name.

      Tip: This name is case-sensitive.
   b. Select the Configure Code Module check box.
   c. Select the Enabled check box for the Initial status.
   d. Click Next.
3. On the Specify the “Code Module to be configured” window, specify the Java class that you wrote:
   a. Type `LogEventActionEDU` in the Code Module Title field.
   b. Click Browse/Add.
   c. If you used Eclipse to create the Java class, locate your working directory where this class is located.

      Example: `<Project folder>\bin\com\ibm\filenet\edu`
   d. Otherwise, locate the folder where you have stored your Java class.

      Example: `C:\Source\com\ibm\filenet\edu`
   e. Select the `LogEventActionEDU.class` file and click Open.
   f. Click Next.
4. Verify the entries in the summary window of the Create an Event Action Wizard and click Finish.
5. Click OK when you receive the confirmation message.

   The Event Action Wizard closes, and you see the Create a Subscription wizard at the window where you left it.
   a. Verify that the event action that you just created is selected and click Next.
   b. On the Specify Additional Properties window, accept the default and click Next.
   c. Verify the entries in the summary window of the Create a Subscription Wizard and click Finish.
   d. Click OK when you receive the confirmation message.

Procedure 6: Test the subscription

1. In Content Engine Enterprise Manager, expand the Root Folder node of the Development object store so that you can see the APIFolder.
   a. Right-click the folder and click New Document.

      The Create New Document Wizard starts.
2. On the Create a document object window, do the following steps:
   a. Type `APISubscriptionDoc` as the document title.
   b. Select the “Without content” option.
   c. Click Next.
3. On the Class and Properties window, select the Product class from the Class list.
   a. Click Create.
   b. Verify that the new document appears in the list view of the Content Engine Enterprise Manager in the APIFolder.
4. Verify that a log file is created as a result of the event action code being successfully executed:
   a. Access the Linux server system (`ccv01135`).
   b. On the Linux desktop, select the computer icon, right click and click the Browse Folder from the context menu.
   c. In the Computer – File Browser screen, double click on the File System on the left pane and navigate to the following location of the file in the right pane:

      `"/opt/ibm/WebSphere/AppServer/profiles/AppSrv01/EventLog.txt"`
   d. Open the file and verify that the information identifying the date and time when your new document was created in the object store, the name of the Document class, and the name of the document are logged in the file.
   e. Close the `EventLog.txt` file.

Update the event action with modified code: Walkthrough

You might need to modify the Java code that you wrote for the event action after you have configured the code module. In this procedure, you are going to modify the code and update this Java class in the Content Engine by checking in the class as a new version of the code module. You also need to update the event action that references this code module.

Procedures

Procedure 1: Modify the Java event action handler

1. Check out the code module:
   a. In Content Engine Enterprise Manager, expand the Development object store > Root Folder > CodeModules.
   b. Select the CodeModules folder and check out (Exclusive) the code module in the right pane with the name that you created (`LogEventActionEDU`).
   c. Save the file when prompted.
2. Modify the Java event action handler source and compile.
   a. Open the `LogEventActionEDU.java` file in Eclipse or any text editor.
   b. Change the name of the log file that the Java code creates in the `writeToLogFileEDU(...)` Java method to `EventModule.txt`.
   c. Save and compile the Java file.

Procedure 2: Update the event action

1. Check in the new version of the code module.
   a. In Content Engine Enterprise Manager, check in the code module.
   b. Refer to Procedure 5, Create event actions for the subscription, page 10-10, step 3 for the location of your Java class.
2. Update the event action with the new code module version:
   a. After checkin, right-click the code module and click Copy Object Reference.
   b. Expand the event action: Development object store > Events > Events Action
   c. In the right pane, open the Properties page of the event action that references the code module.

d. On the Properties page, click the Properties tab and select the All Properties option.

e. Scroll down to the Code Module property at the bottom of the window.

f. Right-click the Code Module value field and click Paste Object.

g. In the Select Object from Paste Buffer window, select the latest version object reference and click OK.

h. In the Properties window, click Apply to save the changes.

i. Click OK to close the window.

Procedure 3: Test the updated event action

1. Test the modified code that has been updated in the code module and in the event action by repeating Procedure 6, Test the subscription, page 10-11.

   a. Create a document with a different name: `TestCodeModule`.

   b. Verify that a log file with the new name is created on the server system (`ccv01135`) in this location:
   `/opt/ibm/WebSphere/AppServer/profiles/AppSrv01/EventModule.txt`

   The file has the text that you added as a message.

Error Handling: Walkthrough

There may be run-time errors in the Java code that is written for the event action. In this procedure, you are going to modify the code to simulate a run-time error. You also need to checkin the new version of the code for the code module and update the event action that references this code module. Then you search for the queue to find the error that your code produced.

Procedures

Procedure 1: Modify the Java class

1. Check out the code module.
2. Modify the `LogEventActionEDU.java` file to include the following line of code in the if block of the `onEvent(...)` method in the last line:

   ```
   WriteToLogFileEDU("Document Title: ” +
   document.getProperties().getStringValue(“Document Title”)+ "\r\n");
   ```

   When this line of code executed, it adds an entry to the log file whenever a document is created. However, the symbolic name for the Document Title is `DocumentTitle` (without any space). So this code creates a run-time error.

Procedure 2: Update the event action and test

1. Use the activity, Update the event action with modified code: Walkthrough, page 10-13 to update the code module with the code you modified in procedure 1 and update the event action.
2. Use the Procedure 6, Test the subscription, page 10-11 to test the subscription.
3. As we intended, there are going to be errors. The event action is going to create the log file but the document title entry is not added to the log file.

Note

Each object store has a table called Queue Item that is created when an object store is created. An item is written to the queue if the subscription is asynchronous.

By default, Content Engine will try to re-execute the code module 7 times if there are any errors. Each retry happens in the order of 2^n starting with 30 seconds, 1 min, 2 min, 4 min, 8 min, 16 min etc.

When the RetryCount reaches zero, the event action times out. You need to reset this value before you try your new modified code again.

Procedure 3: Access the error information

1. Search for the queue items:
   a. In Enterprise Manager, select the Development object store. Select the Search Results node and right click.
   b. Select New Search from the context menu. Content Engine Query Builder opens.
   c. Select Event Queue Item from the list for the Select From Table field.
   d. Select the Criteria:
      - Column: `Retry Count`
      - Condition: `Equal To`
      - Value: `0`
   e. Click OK to start the search.
   f. Click Yes when the FNK_EnterpriseManager message opens.
   g. Click OK at the Query Status window.
   h. Review the entry in the list view on the right pane.

      The item is displayed with the date and time when you ran the subscription and a value between 0-7 for the Retry Count property.
2. Optionally, you can also get the error information from the `p8_server_error.log` which is located on the Linux server system (`ccv01135`).
   a. On the Linux desktop, select the computer icon, right click and click the Browse Folder from the context menu.
   b. In the Computer – File Browser screen, double click on the File System on the left pane and navigate to the following location of the file in the right pane:

      `"/opt/ibm/WebSphere/AppServer/profiles/AppSrv01/FileNet/server1"`
   c. Select p8_server_error.log, right click and select Open with “Text Editor” from the context menu.
   d. Scroll to the end of the file and review the most current errors as the following error:

      “The Document Title property was not found in the properties collection...”

      You will also find a reference to the LogEvenActionEDU class that you wrote.

Note

Fix the error in the code and update the event action before it times out as described in activity Procedure , Update the event action with modified code: Walkthrough, page 10-13.

Procedure 4: Reset retry count

If the event action is timed out, you run the ResetQueueItemRetryCount Saved Search to reset the retry count using the following steps. This template finds all Event Queue Item entries with a retry count of zero and resets that count to a value entered by the user.

1. View the Saved Search:
   a. In Enterprise Manager, expand the Development object store.
   b. Select the Saved Searches node in the left pane.
   c. In the right pane list view, the ResetQueueItemRetryCount is displayed.
2. Run the Saved Search:
   a. Select the Search Results node and right click.
   b. Click the Open Existing Search from the context menu.
   c. In the Open window, select the ResetQueueItemRetryCount.sch and click Open.
   d. In the Content Engine Search Template window, select Search and Bulk Operations option and click the Execute button.
   e. In the Content Engine Bulk Operations window, enter a value for Retry Count (example: 7) and click OK.
   f. Click OK on the Query Status window.
3. Verify the item is updated:
   a. In Enterprise Manager, select the Development object store. Select the Search Results node and right click.
   b. Select New Search from the context menu. Content Engine Query Builder opens.
   c. Select Event Queue Item from the list for the Select From Table field.
   d. Leave the Criteria without any condition entered. If there are values, clear them.
   e. Click OK to start the search.
   f. Click Yes when the FNK_EnterpriseManager message opens.
   g. Click OK at the Query Status window.
   h. Review the entry in the list view on the right pane.

i. Verify that the item is displayed with the date and time assigned by the system (When it is timed out, there was no time displayed) and a value between 0-7 for the Retry Count property.

Note

Fix the error in the code and update the event action before it times out as described in activity Procedure , Update the event action with modified code: Walkthrough, page 10-13.

Optional: Update a database record: Walkthrough

In this activity, you are going to implement a Java event handler to update an external database.

Procedures

Procedure 1: Create a Java class that implements EventActionHandler

1. Use the knowledge that you gained in the previous activity of this unit to do the procedure 1.

Procedure 2: Create a method to update the database

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `updateProductEDU`
   - Parameters:
     - `String` for product_id property value
     - `String` for style property value
   - Throws `Exception`
2. Use `try-catch` blocks.
   a. Create a `catch` block for `Exception` and write the following inside the block:

      ```
      throw new EngineRuntimeException (ExceptionCode.DB_ERROR,
      ioException);
      ```

   b. Write code in the try block as instructed in the following steps.
3. Type the following line of code to register DB2 Server driver:

   ```
   Class.forName("com.ibm.db2.jcc.DB2Driver");
   ```

4. Call `DriverManager.getConnection()`
   - Parameters:
     - `"jdbc:db2://ccv01135:3737/ProdDB"`
     - `"dsrdbm01"`
     - `"IBMFileNetP8"`

   a. Assign the return value to a `java.sql.Connection` object variable.
5. Write an `if` statement to check if the connection is `null` and print out a message saying that the connection failed.

   a. Continue the code inside the else condition as described in the following steps.
6. Create a CallableStatement by calling the `connection.prepareCall(…)` method on the object from step 4.
   - Parameter: `"{CALL InsertProduct(?,?)}"`

   a. Assign the return value to a `CallableStatement` object variable.
7. Set the value for the `IN` parameter and execute the stored procedure:

   a. Call `callableStatement.setString(…)` on the object from step 6.
   - Parameters:
     - `1`
     - `product_id`

   b. Call the `callableStatement.setString(…)` method again.
   - Parameters:
     - `2`
     - `style`

   c. Call `callableStatement.execute()`.

   d. Close the `else` block.

   e. Close the `try` block.

Procedure 3: Implement the onEvent() Method

Use the same Java class that you created in procedure 1 and the knowledge that you gained in the previous activity of this unit to Implement the `onEvent` method.

Procedure 4: Create an Event Action

Use the knowledge that you gained in the previous activity of this unit and the Java class that you wrote in the above procedures (Procedures 1,2 and 3) to create an event action.

Procedure 5: Create a Subscription on a Document class

Use the Event Action that you created in the procedure 4 and the knowledge that you gained in the previous activity of this unit to create a Subscription on the `Product` document class.

Procedure 6: Test the subscription

1. Test the subscription by creating a document and verifying the database entry.
2. Since the database update requires document property values, use the data in the table to create a document.

Wizard field	Value
Document Title	`TestDBEventAction`
Document Class	`Product`
product_id	EAC542
style	`Basic`

   f. Optionally, reuse the code from the previous activity to log the action in a text file.

Information

For your reference, a hard copy of the solution is given at the end of the instructions: Solution code for DBActionEDU.java, p. 10-23

Solution code is provided on the image as a Java file: `DBActionEDU.java`

Solution code for LogEventActionEDU.java

```
package com.ibm.filenet.edu;

import java.io.*;
import com.filenet.api.core.*;
import com.filenet.api.engine.EventActionHandler;
import com.filenet.api.events.ObjectChangeEvent;
import com.filenet.api.exception.*;
import com.filenet.api.util.Id;

public class LogEventActionEDU implements EventActionHandler {
   public void writeToLogFileEDU(String message)throws IOException {
      // Setup FileWriter
      File outputFile = new File("EventLog.txt");
      FileWriter out = new FileWriter(outputFile, true); // true = append
      // Output the message to log file
      out.write(message);
      // Close FileWriter
      out.close();
   }
   public void onEvent(ObjectChangeEvent event, Id subscriptionId)
   throws EngineRuntimeException {
   try{
      // In this case, we assume it's a document
      Document doc = (Document)event.get_SourceObject();
      if (event.getClassName().equalsIgnoreCase("CreationEvent")){
         String docClassName = "   Document class name= " +
         doc.getClassName();
         String docName = "   Document name= " + doc.get_Name();
         // Output data to log
         writeToLogFileEDU("A new document is created on: " + new
         java.util.Date() + "\r\n");
         writeToLogFileEDU(docClassName + "\r\n");
         writeToLogFileEDU(docName + "\r\n");
         writeToLogFileEDU(Document Title +
         doc.getProperties().getStringValue(“Document Title”) + "\r\n");
      }
   }
      catch (IOException e){
         throw new EngineRuntimeException (ExceptionCode.E_FAILED, e);
      }
   }
}
```


Solution code for DBActionEDU.java

```
package com.ibm.filenet.edu;
import com.filenet.api.core.*;
import com.filenet.api.engine.EventActionHandler;
import com.filenet.api.events.ObjectChangeEvent;
import com.filenet.api.exception.*;
import com.filenet.api.util.Id;
import java.io.*;
import java.sql.*;
public class DBActionEDU implements EventActionHandler {
   public void updateProductEDU(String product_id, String style) throws
   Exception {
   try {
      CallableStatement cs;
      Class.forName("com.ibm.db2.jcc.DB2Driver");
      java.sql.Connection connection = DriverManager.getConnection
      ("jdbc:db2://ccv01135:3737/ProdDB", "dsrdbm01", "IBMFileNetP8");
      if (connection == null){
         System.out.println("Connection failed.");
      }// if
      else{
         cs = connection.prepareCall("{call InsertProduct(?,?)}");
         // Set the value for the IN parameter
         cs.setString(1, product_id);
         cs.setString(2, style);
         // Execute the stored procedure
      }//else
   }//try

   catch (SQLException e)
   {
      throw new EngineRuntimeException (ExceptionCode.DB_ERROR, e);
   }
}
   public void writeToLogFileEDU(String message)throws IOException {
   // Setup FileWriter
   File outputFile = new File("DBLog.txt");
   FileWriter out = new FileWriter(outputFile, true); // true = append
   // Output the message to log file
   out.write(message);
   // Close FileWriter
      out.close();
   }
```

```
   public void onEvent(ObjectChangeEvent event, Id subscriptionId)
   throws EngineRuntimeException {
   // In this case, we assume it's a custom object
   Document product = (Document)event.get_SourceObject();
   try {
      if (event.getClassName().equalsIgnoreCase("CreationEvent")){
         writeToLogFileEDU("A new document is created on: " + new
         java.util.Date() + "\r\n");
         writeToLogFileEDU("Document class name" + product.getClassName() +
         "\r\n");
         writeToLogFileEDU("Document name" + product.get_Name() + "\r\n");
         updateProductEDU(product.getProperties().getStringValue("product_id
         "),product.getProperties().getStringValue("style"));
      }
   }
   catch (Exception e){
   throw new EngineRuntimeException(ExceptionCode.E_FAILED, e);
   }
   }
}
```


Unit 11. Auditing

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class in this unit extends the CEConnectionEDU class from the “Communication with the Content Engine” unit.
- Concepts and some parts of the code from Properties unit is used.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt window.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. (These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.)

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 11.1. Retrieve the audit history for existing objects

Overview

Why is this lesson important to you?

- The administrator of your company has configured a document class for the auditing. Management wants to retrieve the auditing history of a document of that class.
- An object has been updated. Management wants to have audit data values before and after modification.
- As their programmer, you are going to write code to perform these tasks.

Prerequisites

Activity , Configure auditing, page 11-7

Activities

- Retrieve the audit history for existing objects: Challenge, page 11-9
- Retrieve the audit history for existing objects: Walkthrough, page 11-13
- Optional: Audit the changes to a specific property of a class: All levels, page 11-21

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Configure auditing

Important

This activity is required for both skill levels.

Procedure 1: Enable auditing

1. Log on to the Content Engine Enterprise Manager as p8admin.
2. Expand the Development object store node, right-click, and click Properties.

   The Properties window opens.
3. In the Properties window, click the General tab and then select the Auditing Enabled check box.
4. Click Apply to save the changes.
5. Click OK to close the window.

Procedure 2: Set audit definition

1. In Content Engine Enterprise Manager, expand the Object Stores > Development > Document Class > Product.
2. Select the Product document class and open its Properties window.
3. Click the Audit Definitions tab.
4. Add the events to be audited:
   a. Select Creation from the Event list.
   b. Select the Success and Failure check boxes for Audit Type.
   c. Select the Is Enabled check box.
   d. Click Add as New.
   e. Repeat steps 4a through 4d for the Checkout, Checkin, Update event, and Deletion events.
   f. In addition, for the Update event, select “Original and modified objects” from the list for the Object State Recording Level field.
   g. Click Apply to save the changes.
   h. Click Yes to close the Apply Changes window.
   i. Click OK to close the Properties window.
5. Select the Development object store, right click and click Refresh to reflect the changes made.

Procedure 3: Test the configuration

1. Sign in to Workplace as p8admin and locate Development > APIFolder.
2. Create a new document using the specifications in the following data table.

Item	Value
Document class	`Product`
Document Title	`APIAuditEvent.doc`
Add as major version:	`Yes`
product_id	`PDT108`
style	`Basic (Default)`
Location for the content for the document	`C:\CMJavaAPIProg\SampleDocuments\APIAuditEvent.doc`

3. Check out the document and check in the document as a major version.
4. View the Audit history for the document.
   a. Open the information page of the document.
   b. Click the History link in the left pane.
   c. Select the Events check box.
   d. Select yes from the Include All Versions list.
   e. Click Search at the bottom of the window.
   f. Verify that a list of events with details is displayed.
   g. Exit from the page.
5. Test the update Event.
   a. Select the document information page.
   b. Modify the properties using the following values and save the changes.
      - style: `Deluxe`
      - product_id: `CAR123`
   c. Repeat step 4 to verify the audit history information for the update event (select only update event).
6. View the audit history for a given document version.
   a. Open the information page of the document.
   b. Click the Versions link and then click the information icon for the version you created in step 3.
   c. Repeat step 4 to verify the audit history information.

Retrieve the audit history for existing objects: Challenge

Challenge

Use the data in the table to create a Java class that extends CEConnectionEDU and write code to retrieve the following:

- A document with audit related properties
- Audited events from a Document object
- Updated properties and their values from the Update Event by reusing the code that you wrote in the “Properties” unit
- Audited events for document versions

Data

Item	Value

Packages to import	`java.util.Iterator`
	`com.filenet.api.collection.*`
	`com.filenet.api.constants.*`
	`com.filenet.api.core.*`
	`com.filenet.api.property.*`
	`com.filenet.api.query.*`
	`com.filenet.api.events.*`
	`com.filenet.api.meta.*`

Object store name	`"Development"`
Folder path for the document	`"/APIFolder/APIAuditEvent.doc"`

Document properties to retrieve	`PropertyNames.NAME`
	`PropertyNames.AUDITED_EVENTS`
	`PropertyNames.VERSIONS`
	`PropertyNames.CLASS_DESCRIPTION`
	`PropertyNames.MAJOR_VERSION_NUMBER`
	`PropertyNames.MINOR_VERSION_NUMBER`

Note

If you did not complete the “Property” unit, the solution has the `PropertyEDU` class with the `displaypropertyEDU(...)` method.

Verification

Run the program using Eclipse or the Command Prompt and verify that the following are displayed in the output window:

- A list of audited events for the specified document and the events details
- A list of the updated properties from the update event
- A list of audited events for each document version and the events details

Sample output for a checkin and creation event

```
   Got the connection
   Name of the domain: P8Domain
   Name of the object store: Development
   APIAuditEvent.doc  Document is retrieved
   object:APIAuditEvent.doc
   ----------------------------
   The event name:CheckinEvent
   Initiating user: cn=P8Admin, o=sample
   Time created:Fri Mar 25 15:56:03 PDT 2011
   Event status: 0
   Document version:2.0
   ---------------
   The event name:CreationEvent
   Initiating user: cn=P8Admin, o=sample
   Time created:Fri Mar 25 15:50:03 PDT 2011
   Event status: 0
   Document version:1.0
---------------
```


Sample output for an update event

```
Got the connection
Name of the domain: P8Domain
Name of the object store: Development
APIAuditEvent.doc  Document is retrieved
object:APIAuditEvent.doc
-----------------------------
The event name:UpdateEvent
Initiating user: cn=P8Admin, o=sample
Time created:Fri Mar 25 15:58:03 PDT 2011
Event status: 0
Document version:2.0
----------------
new value:
Property name = DateLastModified
Property value = Mar 25 15:58:03 PDT 2011
old value:
Property name = DateLastModified
Property value = Mar 25 15:56:04 PDT 2011
----------------
new value:
Property name = model_code
Property value = CAR123
old value:
Property name = model_code
Property value = null
----------------
new value:
Property name = style
Property value = Deluxe
old value:
Property name = style
Property value = Basic
...
----------------
```

Note

The list of property values that you get can be much longer than what is shown here.

Retrieve the audit history for existing objects: Walkthrough

Procedure: Retrieve the audited events for a Document object

Procedure: Retrieve the audited events for document versions

Retrieve the audited events for a Document object

Procedure 1: Create a Java class that extends CEConnectionEDU

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the “New Java Class” window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type `AuditEDU` as the class name in the Name field.
5. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
6. Select `public static void main(String[] args)` for “Which method stubs would you like to create?” option.
   a. Clear other options.

7. Click Finish to add the class and close the window.
   a. Verify that the new class is listed under your package in the Project Explorer.
8. Write code inside the class that you just added.
9. Import the following packages before the declaration of the class:
   - `java.util.Iterator`
   - `com.filenet.api.collection.*`
   - `com.filenet.api.constants.*`
   - `com.filenet.api.core.*`
   - `com.filenet.api.property.*`
   - `com.filenet.api.query.*`
   - `com.filenet.api.events.*`
   - `com.filenet.api.meta.*`
10. Inside the main method, write a `try-catch` block.
    a. Call `exception.printStackTrace()` in the catch block to display the exceptions thrown.
11. Add custom methods and call it from the main method as described in the following steps.

Procedure 2: Write a method to retrieve a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getDocumentEDU`
   - Parameters:
     - `ObjectStore` object
     - `String` for the folder path
   - Returns: `Document` object
2. Create a new instance a `PropertyFilter` object
   a. Assign the return value to a variable of type `PropertyFilter` object.
3. Call `PropertyFilter.addIncludeProperty(...)`.
   - Parameters:
     - `0`
     - `null`
     - `null`
     - `PropertyNames.NAME`
     - `1`

4. Repeat step 3 to include the following properties to the property filter:
   - `PropertyNames.AUDITED_EVENTS`
   - `PropertyNames.VERSIONS`
   - `PropertyNames.CLASS_DESCRIPTION`
   - `PropertyNames.MAJOR_VERSION_NUMBER`
   - `PropertyNames.MINOR_VERSION_NUMBER`
5. Call `Factory.Document.fetchInstance(...)`.
   - Parameters:
     - `Objectstore` object
     - Folder path that is passed into this method
     - `PropertyFilter` object that you got in steps 2 through 4
6. Get and display the document name by calling `System.out.println(…)`.
   - Parameter: `Document.get_Name()`
7. Return the `Document` object.

Procedure 3: Write a method to retrieve audited events from a Document object

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getEventsEDU`
   - Parameter: `Containable` object

Note

The Document object is used in this activity for the Containable object.

You can replace it with the Folder or CustomObject object to retrieve their audited events.

2. Retrieve and display the document name by calling `System.out.println(…)`.
   - Parameter: `document.get_Name()`
3. Retrieve the events by calling `document.get_AuditedEvents()`.
   a. Assign the return value to a variable of type `EventSet`.
4. Call `eventSet.iterator()`.
   a. Assign the returned value to a variable of the type `Iterator`.
5. Write a `while` loop and iterate through the collection.

```
while (iterator.hasNext())
```

6. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into an `Event` type and assign it to a variable.
7. Retrieve and display the class name of the event by calling `System.out.println(…)`.
   - Parameter: `event.getClassName()`
8. Repeat step 7 to retrieve and display the initiating user, date created, and event status for each event.
   a. Use each of the following parameters:
      - `event.get_InitiatingUser()`
      - `event.get_DateCreated()`
      - `event.get_EventStatus().toString()`
9. Ensure that the Containable object is the Document object in an `if` statement.

   ```
   if(containable instanceof Document)
   ```

   a. Typecast into a `Document` object type and assign it to a variable.
10. Inside the `if` block, retrieve the document version numbers by calling the following methods:
    - `document.get_MajorVersionNumber()`
    - `document.get_MinorVersionNumber()`

    a. Display the values by calling `System.out.println(...)`.
11. Close the `while` loop and the method.

Procedure 4: Retrieve the modified properties for an object from an update event

Continue with the same method from procedure 2.

1. Determine if the event is an update event in an `if` statement:

   ```
   if (event.getClassName().equalsIgnoreCase("UpdateEvent"))
   ```

   a. Write code inside the `if` statement as described in the following steps.
   b. Typecast the event to `UpdateEvent` type and assign the event to a variable.
2. Retrieve the updated properties by calling `updateEvent.get_ModifiedProperties()`on the object from step 1.
   a. Assign the return value to a variable of type `StringList`.
3. Call `stringList.iterator()`on the object from step 3.
   a. Assign the returned value to a variable of the type `Iterator`.
4. Create a new instance a `PropertyFilter` object
   a. Assign the return value to a variable of type `PropertyFilter` object.

5. Write a second `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```

6. Call `iterator.next()`within the `while` loop to retrieve each property name.
   a. Typecast the return value into a `String` type and assign it to a variable.
7. Call `PropertyFilter.addIncludeProperty(...)`.
   - Parameters:
     - `0`
     - `null`
     - `null`
     - `String` value from step 6
     - `1`
8. Close the second `while` loop.
9. Repeat step 7 to include `PropertyNames.CLASS_DESCRIPTION`.
10. Get the source object for the update event by calling `updateEvent.get_SourceObject()`.
    a. Assign the return value to a variable of type `IndependentObject` object.
11. Get the original object before the update event by calling `updateEvent.get_OriginalObject()`.
    a. Assign the return value to a variable of type `IndependentObject` object.
12. Instantiate the `PropertiesEDU` class that you wrote in the earlier unit and assign the return value to a variable.

    ```
    PropertiesEDU propertiesInstance = new PropertiesEDU();
    ```

    Tip: You are going to use the method of this class inside the `while` loop in the following steps.
13. Write a third `while` loop and iterate through the collection.
    a. Call `iterator.next()`on the object from step 3 inside the `while` loop to retrieve each property name.
    b. Typecast the return property name into a `String` type and assign it to a variable.
14. Retrieve each property for the source object by calling `independentObject.getProperties().get(…)` on the object from step 10.
    - Parameter: `String` from step 13.

    a. Assign the returned value to a variable of the type `Property`.

b. Display the name and new updated value of the property by calling the `displayPropertyEDU(…)`method of the `PropertiesEDU` class from step 13.

- Parameters:
  - `IndependentObject` object from step 4
  - `Property` object from step 10

15. Repeat step 14 to display the old values of the properties of original object before the object is updated.

a. Use the `IndependentObject` from step 11.

Note

If you did not complete the “Property” unit, the solution has the `PropertyEDU` class with the `displaypropertyEDU(...)` method. Optionally, you can copy the `displaypropertyEDU(...)` method from the `PropertyEDU` class into this `AuditEdu` class.

16. End the method:

a. `Close the inner (second) while` loop.

b. Close the `if` statement.

c. Close the outer (first) `while` loop.

d. End the method.

Procedure 5: Instantiate the class and call the methods that you wrote

1. Inside the `try` block of the main() method, create an instance of the class.

   Example: `AuditEDU auditInstance = new AuditEDU();`

2. Call the `getCEConnectionEDU(...)` method of the `CEConnectionEDU` class:

   - Parameters:
     - `"p8admin"`
     - `"IBMFileNetP8"`

   a. Assign the returned `Connection` object to a variable.

3. Call the `getDomainEDU(…)`method of the `CEConnectionEDU` class.

   - Parameter: `Connection` object from step 2

   a. Assign the returned `Domain` object to a variable.

4. Call the `getObjectStoreEDU(…)` method of the `CEConnectionEDU` class:
   - Parameters:
     - `Domain` object from step 3
     - `"Development"`

   a. Assign the returned `ObjectStore` object to a variable.
5. Call the `getDocumentEDU(...)` method.
   - Parameters:
     - `ObjectStore` object from step 4
     - `"/APIFolder/APIAuditEvent.doc"` (folder path)

   a. Assign the return value to a variable of type `Document`.
6. Call the `getEventsEDU(...)` method.
   - Parameter: `Document` object from step 5

Procedure 6: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt as instructed in the "Communication with the Content Engine Server" unit.
2. Verify that a list of audited events for the specified document, the events details, and the updated properties from the update event are displayed in the output window.

Sample output

Refer to the sample output in Retrieve the audit history for existing objects: Challenge, page 11-9.

Retrieve the audited events for document versions

Procedure 1: Write a method to retrieve audited events for document versions

1. Use the same Java class and define the method using the following signature:
   - Scope: `public`
   - Name: `getVersionsEventsEDU`
   - Parameter: `Document` object
2. Retrieve the versions by calling `document.get_Versions()`.
   a. Assign the return value to a variable of type `VersionableSet`.
3. Call `VersionableSet.iterator()`on the object from step 2.
   a. Assign the returned value to a variable of the type `Iterator`.
4. Write a `while` loop and iterate through the collection.

5. Call `iterator.next()`within the `while` loop to retrieve each document version.
   a. Typecast the return value into a `Document` type and assign it to a variable.
6. Call `getEventsEDU(…)` method that you wrote in the procedure 3 of the previous activity to display the event details.
   - Parameter: `Document` object from step 5.

   a. Close the `while` loop and the method.

Procedure 2: Call the method inside the main() method

1. Call the `getVersionsEventsEDU(...)` method inside the `try` block of the main() method.
   - Parameter: `Document` variable that you got in the previous lesson

2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program and verify that a list of audited events for each document version and the events details are displayed in the output window.
   a. The sample output for each document version is similar to the document that is shown in the Retrieve the audit history for existing objects: Challenge, page 11-9.

Optional: Audit the changes to a specific property of a class: All levels

Procedure: Retrieve the audited events for a Document object

Procedure 1: Assign properties to the Update Event

1. Log on to the Content Engine Enterprise Manager as P8admin.
2. Locate the Update Event class:
   a. Expand the Development object store > Other Classes > Event > Object Change Event > Update Event node.
   b. Right-click and click Properties.
3. In the Properties window, click the Property Definitions tab.
4. Click the Add/Remove button.

   The Add/Remove Properties window opens.
5. Select the `style` property from the left pane (Available) and click the double arrow button to move the property to the right pane (Selected).
6. Repeat step 5 for the `product_id` property.
7. Click OK to close the window.
8. Click Yes to the Apply Changes window.
9. In the main window, click the Update Event node and verify that the properties that you selected are displayed in the right pane.

Procedure 2: Configure auditing for a property of a class

1. In Content Engine Enterprise Manager, expand the Object Stores > Development > Document Class > Product.
2. Select the Product document class, right-click and open its Properties window.
3. In the Properties window, click the Property Definitions tab.
4. Set the `Audit As` value for each property:
   a. Select product_id from the list.

b. Click the Edit button.

c. In the product_id Properties window, click the More tab.

d. In the Audit As section, click the Set Value button.

e. In the Select Property Template for Auditing window, select `product_id` from the list.

f. Click OK.

g. Verify that Audit As field has the GUID for `product_id`.

h. Click OK to close the product_id Properties window.

5. Repeat steps 4a through 4h for the `style` property.
6. Leave the Product Class Properties window open for the next procedure.

Procedure 3: Update the audit definitions

1. In Product Class Properties window, click the Audit Definitions tab.
2. Edit the Update Event:
   a. Select the Update event from the list.
   b. For the Object State Recording Level field, change the “Original and modified objects” value to None by selecting from the list.
   c. Click the Update Existing button.
   d. Click Apply to save the changes.
   e. Click Yes to close the Apply Changes window.
   f. Click OK to close the Properties window.
3. Select the Development object store, right-click, and click Refresh to reflect the changes made.

Procedure 4: Add and update a document

1. Sign in to Workplace as P8admin and locate the Development > APIFolder.
2. Create a new document (without content) using the specifications in the following data table.

Item	Value
Document class	`Product`
Document Title	`TestAuditProps.doc`
Add as major version:	`Yes`
product_id	`PID888`
style	`Basic (Default)`

3. Update the document properties.
   a. Select the document information page.
   b. Modify the properties using the following values and save the changes.
      - style: Luxury
      - product_id: VEH123

Procedure 5: Modify the code and test the configuration

1. Use the same Java class that you created in the previous activity.
2. Add the following lines of code to the `getEventsEDU(...)` method that you created in the previous activity:
   a. Add the lines just before you retrieve the update event modified properties.

```
System.out.println("New product ID:" +
updateEvent.getProperties().getStringValue("Product_id"));
System.out.println("New product style:" +
updateEvent.getProperties().getStringValue("style"));
```

3. Inside the `try` block of the main() method, before calling the `getEventsEDU(...)` method, call the `getDocumentEDU(...)` method.
   - Parameters:
     - `ObjectStore` object from previous activity
     - `"/APIFolder/TestAuditProps.doc"` (folder path)
   a. Assign the return value to a variable of type `Document`.
4. Comment out the method calls that are not tested in this activity.
5. Save and execute the code.
6. Verify that only the properties that you configured for auditing are retrieved and displayed in the console.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the object store: Development
TestAuditProps.doc Document is retrieved
object: TestAuditProps.doc
-----------------------------
The event name:UpdateEvent
Initiating user: cn=P8Admin, o=sample
Time created:Fri April 08 11:02:58 PDT 2011
Event status: 0
Document version:1.0
----------------
New product ID: VEH123
New product style: Luxury
```


Lesson 11.2. Retrieve the audit history for deleted objects

Overview

Why is this lesson important to you?

From time to time, objects are deleted, and management wants to get information about who deletes the objects and when. As their programmer, you are going to write code to do this task.

Activities

- Retrieve the audit history for deleted objects: Challenge, page 11-27
- Retrieve the audit history for deleted objects: Walkthrough, page 11-29

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Retrieve the audit history for deleted objects: Challenge

Challenge

Use the same Java class that you created in the previous lesson. Use the data in the table to retrieve the audited events for deleted objects and display the names and details of the events.

Data

Item	Value
Object store name	`"Development"`
SQL string	`sql = "SELECT * FROM DeletionEvent"` `sql += "WHERE (InitiatingUser like '%" + user + "%')"`

Hint

Use SearchScope and SearchSQL objects with the SQL string from the preceding data table to retrieve the EventSet.

Steps to retrieve the information about the events are similar to the previous lesson activity.

Verification

Run the program and verify that a list of audited events for the deleted objects and the events details is displayed in the output window.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the object store: Development
APIAuditEvent.doc  Document is retrieved
The event name: DeletionEvent
Initiating user: cn=p8admin,o=sample
Time created: Fri Mar 25 12:14:21 PDT 2011
Event status: 0
Object name: {D21A2820-FA78-4678-A068-D88B950586AA}
---------------
object:SampleSubsDoc
-----------------------------
The event name:DeletionEvent
Initiating user: cn=p8admin,o=sample
Time created:Fri Mar 25 15:46:29 PDT 2011
Event status: 0
Document version:2.0
---------------
The event name:CheckinEvent
Initiating user: cn=p8admin,o=sample
Time created:Fri Mar 25 15:41:20 PDT 2011
Event status: 0
Document version:2.0
---------------
The event name:CreationEvent
Initiating user: cn=p8admin,o=sample
Time created:Fri Mar 25 15:40:12 PDT 2011
Event status: 0
Document version:1.0
---------------
```

Note

The list of audited events for the deleted objects and the events details that you get can be longer than what is shown here.

Retrieve the audit history for deleted objects: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to retrieve audit history

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getDeletionEventsEDU`
   - Parameters:
     - `ObjectStore` object
     - `String` for the user name
2. Create an instance of SearchScope class.

   `new SearchScope(os)`

   - Parameter: `ObjectStore` object that is passed into the method.

   a. Assign the return value to a variable of type `SearchScope`.
3. Build an SQL statement using the data below and assign it to a `String` variable.

```
sql = "SELECT * FROM DeletionEvent"
sql += " WHERE (InitiatingUser like '%" + user + "%') "
```

4. Create an instance of SearchSQL class: `new SearchSQL(...)`.
   - Parameter: `String` variable for SQL statement from step 3

   a. Assign the return value to a variable of type `SearchSQL`.
5. Create a new instance a `PropertyFilter` object

   a. Assign the return value to a variable of type `PropertyFilter` object.
6. Call `PropertyFilter.addIncludeProperty(...)` on the object from step 5.
   - Parameters:
     - `0`
     - `null`
     - `null`
     - `PropertyNames.SOURCE_OBJECT`
     - `1`

7. Call `SearchScope.fetchObjects(...)`.
   - Parameters:
     - `SearchSQL` object from step 4
     - `Integer.valueOf("50")`
     - `PropertyFilter` object from step 6
     - `Boolean.valueOf(true)`

   a. Assign the return value to a variable of type `EventSet`.
8. Call `eventSet.iterator()`.

   a. Assign the returned value to a variable of the type `Iterator`.
9. Write a `while` loop to retrieve each deletion event from the collection.

   ```
   while (iterator.hasNext())
   ```
10. Call `iterator.next()`within the `while` loop.

    a. Typecast the return value into a `Event` type and assign it to a variable.
11. Retrieve and display the class name of the event by calling `System.out.println(…)`.
    - Parameter: `event.getClassName()`
12. Repeat step 9 to retrieve and display the initiating user, date created, event status, and object name for each event using each of the following parameters:
    - `event.get_InitiatingUser()`
    - `event.get_DateCreated()`
    - `event.get_EventStatus().toString()`
    - `event.get_Name()`

Information

Optional steps:

These two steps get the information for other events as described in the previous lesson.

13. Retrieve the `Document` (or any `Containable`) object by calling `event.getProperties().getObjectValue(...)`.
    - Parameter: `PropertyNames.SOURCE_OBJECT`

    a. Assign the return value to a variable of type `Containable`.
14. Call the `getEventsEDU(...)` method that you wrote in the previous lesson to retrieve other events for this deleted object.
    - Parameter: `Containable` from step 11

15. Close the `while` loop and the method.

Procedure 2: Call the method inside the main() method

1. Call the `getDeletionEventsEDU(...)` method inside the `try` block of the main() method.
   - Parameters:
     - `Objectstore` variable that you got in the previous lesson
     - `"p8admin"`
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program and verify that a list of audited events for the deleted objects is displayed in the output window.
   a. Refer to the sample output in Retrieve the audit history for deleted objects: Challenge, page 11-27.

Lesson 11.3. Work with custom events

Overview

Why is this lesson important to you?

You have been instructed to audit custom events. As the programmer, you are going to write code to raise the custom event.

Prerequisites

- Activity , Create a custom event subclass, page 11-35

Activities

- Raise a custom event: Challenge, page 11-37
- Raise a custom event: Walkthrough, page 11-39

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Create a custom event subclass

Important

This activity is required for both skill levels.

In this activity, you are going to create a custom event subclass and add this event to the list of audit definitions for a given document class. You then raise this event programmatically in the following activity.

Procedures

Procedure 1: Create a custom event subclass

1. Sign in to the Content Engine Enterprise Manager as CEAdmin.
2. Go to Objectstores > Development > Other Classes > Event > Object Change Event > Custom Event.
3. Right-click and click New Class.

   The Content Engine Create a Class Wizard opens.
4. Click Next and enter the name for the event: `CustomJavaAPIEvent`
5. Click Next to access the Select Properties window.
   a. In the Available pane, select the following options and click Add to add the properties to the Selected pane.
      - `Document Title`
      - `Description`
   b. Click Next two times and verify that the information on the final wizard screen is correct, and then click Finish to close the wizard.
   c. When you receive the confirmation message, click OK to close the window.

Procedure 2: Set audit definition to a document class

1. In Content Engine Enterprise Manager, expand the Document Class node.
2. Select the Product document class and open its Properties window.

3. Click the Audit Definitions tab.
4. Add the `CustomJavaAPIEvent` that you just created in procedure 1.
   a. Select the event from the Event list.
   b. Select the Success and Failure check boxes.
   c. Select Is Enabled.
   d. Click Add as New.
   e. Click Apply to save the changes.
   f. Click Yes to close the Apply Changes window.
   g. Click OK to close the Properties window.
5. Select the Development object store node and click Refresh to reflect the changes made.

Procedure 3: Create a document

1. Sign in to Workplace as p8admin and locate Development > APIFolder.
2. Create a new document using the specifications in the following data table.

Item	Value
Document class	`Product`
Document name	`CustomAPIEvent.doc`
product_id	`PDT789`
Add as a major version	`Yes`
Location for the content for the document	`C:\CMJavaAPIProg\SampleDocuments\CustomAPIEvent.doc`

Raise a custom event: Challenge

Challenge

Use the same Java class that you created in the previous lesson and the data in the table to write code to raise a custom event for a document.

Data

Item	Value
Object store name	`"Development"`
Folder path for the document	`"/APIFolder/CustomAPIEvent.doc"`
Custom Event Name	`"CustomJavaAPIEvent"`

Verification

Run the program and verify the results using Workplace.

- Open the information page of the given document.
- Click the History link in the left pane.
- Select the Events check box and click Search at the bottom of the window.
- Verify that the event that you just raised is added to the list.

Raise a custom event: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to raise a custom event for a document

1. Define a method using the following signature:
   - Scope: `public`
   - Name: `raiseCustomEventEDU`
   - Parameters:
     - `ObjectStore` object
     - `Document` object
     - `String` for custom event name
2. Call `Factory.CustomEvent.createInstance(...)`.
   - Parameters:
     - `ObjectStore` object
     - `String` for custom event name

   a. Assign the return value to a variable of type `CustomEvent`.
3. Call the `customEvent.set_EventStatus()` on the object from step 2 to set the value.
   - Parameter: `new Integer(99)`
4. Call `document.raiseEvent(...)`.
   - Parameter: `CustomEvent` object from step 2.
5. Call `document.save(...)`.
   - Parameter: `RefreshMode.REFRESH`
6. Display a message confirming that a event has been raised by calling `System.out.println(...)`.

Procedure 2: Call the method inside the main() method

1. Call the `getDocumentEDU(...)` method.
   - Parameters:
     - `ObjectStore` object that you got in first lesson
     - `"/APIFolder/CustomAPIEvent.doc"` (folder path)
2. Call the `raiseCustomEventEDU(...)` method inside the `try` block of the main() method.
   - Parameters:
     - `ObjectStore` object variable that you got in first lesson
     - `Document` object from step 1
     - `"CustomJavaAPIEvent"`
3. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the code and verify the results:
   a. Sign in to Workplace as p8admin.
   b. Open the information page of the given document (`CustomAPIEvent.doc`).
   c. Click the History link in the left pane.
   d. Select the Events check box.
   e. Click Search at the bottom of the window.
   f. Verify that the event that you just raised is added to the list.

Sample output

```
Got the connection
Name of the Domain: P8Domain
Objectstore is: Development
CustomAPIEvent.doc  Document is retrieved
CustomJavaAPIEvent is raised
```


Solution code for AuditEDU.java

```
package com.ibm.filenet.edu;

import java.util.Iterator;
import com.filenet.api.collection.*;
import com.filenet.api.constants.*;
import com.filenet.api.core.*;
import com.filenet.api.property.*;
import com.filenet.api.query.*;
import com.filenet.api.events.*;
import com.filenet.api.meta.*;

public class AuditEDU extends CEConnectionEDU {
   public void getEventsEDU(Containable CEObject){
      System.out.println("object:" + CEObject.get_Name());
      System.out.println("----------------------------");
      EventSet auditedEvents  = CEObject.get_AuditedEvents();
      Iterator eventIt = auditedEvents.iterator();
      while (eventIt.hasNext()){
         Event event = (Event)eventIt.next();
         System.out.println("The event name:" + event.getClassName());
         System.out.println("Initiating user: " +
         event.get_InitiatingUser());
         System.out.println("Time created:" + event.get_DateCreated());
         System.out.println("Event status: " +
         event.get_EventStatus().toString());
         if (CEObject instanceof Document){
            Document doc = (Document)CEObject;
            System.out.println("Document version:" +
            doc.get_MajorVersionNumber() +"." +
            doc.get_MinorVersionNumber());
         }
         System.out.println("----------------");
         if (event.getClassName().equalsIgnoreCase("UpdateEvent")){
            UpdateEvent updateEvent = (UpdateEvent)event;
            System.out.println(“New product ID:” +
            updateEvent.getProperties().getStringValue(“ProductID”));
            System.out.println(“New product style:” +
            updateEvent.getProperties().getStringValue(“style”));
            StringList modifiedProperties =
            updateEvent.get_ModifiedProperties();
            Iterator it = modifiedProperties.iterator();
            PropertyFilter pf = new PropertyFilter();
            while (it.hasNext()){
```

```
            pf.addIncludeProperty(0, null, null, (String)it.next(),1);
            }
            pf.addIncludeProperty(0, null, null,
            PropertyNames.CLASS_DESCRIPTION,1);
            IndependentObject sourceObj = updateEvent.get_SourceObject();
            IndependentObject originalObj =
            updateEvent.get_OriginalObject();
            it = modifiedProperties.iterator();
            while (it.hasNext()){
            String propertyName = (String)it.next();
            System.out.println("new value:");
            displayPropertyEDU(sourceObj,
            sourceObj.getProperties().get(propertyName));
            System.out.println("old value:");
            displayPropertyEDU(originalObj,
            originalObj.getProperties().get(propertyName));
            } // while
         } //if
      }
   public void getDeletionEventsEDU(ObjectStore os, String user){
      SearchScope search = new SearchScope(os);
      String sql = "SELECT * FROM DeletionEvent";
      sql += " WHERE (InitiatingUser like '%" + user + "%') ";
      SearchSQL searchSQL = new SearchSQL(sql);
      EventSet events = (EventSet)search.fetchObjects
      (searchSQL,Integer.valueOf("50"), null, Boolean.valueOf(true));
      Event event;
      Iterator it = events.iterator();
      while (it.hasNext()){
         event = (Event)it.next();
         System.out.println("The event name: " + event.getClassName());
         System.out.println("Initiating user: " +
         event.get_InitiatingUser());
         System.out.println("Time created: " + event.get_DateCreated());
         System.out.println("Event status: " +
         event.get_EventStatus().toString());
         System.out.println("Object name: " + event.get_Name());
         System.out.println("----------------");
         Containable CEObject =
         (Containable)event.getProperties().getObjectValue(PropertyNames.SOU
         RCE_OBJECT);
         getEventsEDU(CEObject);
      }
   }
```



```
public void raiseCustomEventEDU (ObjectStore objectStore, Document
document, String eventName){
   CustomEvent customEvent =
   Factory.CustomEvent.createInstance(objectStore,eventName);
   customEvent.set_EventStatus(new Integer(99));
   document.raiseEvent(customEvent);
   document.save(RefreshMode.REFRESH);
   System.out.println("CustomJavaAPIEvent is raised");
}

public void displayMultiStringValuesEDU(StringList values){
   Iterator it = values.iterator();
   String value;
   while (it.hasNext()){
      value = (String)it.next();
      System.out.print(value + ",");
   }
   System.out.println("");
}
public PropertyDescription getPropertyDescriptionEDU(EngineObject
document, String propertyName){
   PropertyDescription propertyDescription = null;
   ClassDescription classDescription = document.get_ClassDescription();
   PropertyDescriptionList propDescs =
   classDescription.get_PropertyDescriptions();
   //propertyDescription = propDescs.get(propDescs.indexOf(property))
   Iterator propIt = propDescs.iterator();
   while (propIt.hasNext()){
propertyDescription = (PropertyDescription)propIt.next();
   if (propertyDescription.get_SymbolicName().
   equalsIgnoreCase(propertyName)){
      System.out.println("Property name = " +
      propertyDescription.get_SymbolicName());
      System.out.println("Read only? " +
      propertyDescription.get_IsReadOnly().toString());
      System.out.println("Required? " +
      propertyDescription.get_IsValueRequired().toString());
      System.out.println("Settability " +
      propertyDescription.get_Settability().toString());
      System.out.println("System? " +
      propertyDescription.get_IsSystemGenerated().toString());
      System.out.println("Data type = " +
      propertyDescription.get_DataType().toString());
      System.out.println("Cardinality " +
```

```
        propertyDescription.get_Cardinality().toString());
        break;
     }
  }
  return propertyDescription;
  }
  public void displayPropertyEDU(EngineObject document, Property property)
  {
     PropertyDescription propertyDescription =
     getPropertyDescriptionEDU(document, property.getPropertyName());
     System.out.print("Property value = ");
     switch(propertyDescription.get_DataType().getValue()){
        case TypeID.DATE_AS_INT:
  if (propertyDescription.get_Cardinality().getValue() ==
  Cardinality.SINGLE_AS_INT){
  if (property.getDateTimeValue() != null)
  System.out.println(property.getDateTimeValue().toString());
  }
  else
  if (property.getDateTimeListValue() != null)
  System.out.println(property.getDateTimeListValue().toString());
  break;
  case TypeID.STRING_AS_INT:
  if (propertyDescription.get_Cardinality().getValue() ==
  Cardinality.SINGLE_AS_INT)
  System.out.println(property.getStringValue());
  else
  displayMultiStringValuesEDU(property.getStringListValue());
  //System.out.println(property.getStringListValue().toString());
  break;
  case TypeID.BOOLEAN_AS_INT:
  if (propertyDescription.get_Cardinality().getValue() ==
  Cardinality.SINGLE_AS_INT){
  if (property.getBooleanValue() != null)
  System.out.println(property.getBooleanValue().toString());
  }
  else
  if (property.getBooleanListValue() != null)
  System.out.println(property.getBooleanListValue().toString());
  break;
  case TypeID.DOUBLE_AS_INT:
  if (propertyDescription.get_Cardinality().getValue() ==
  Cardinality.SINGLE_AS_INT){
  if (property.getFloat64Value() != null){
```



```
   System.out.println(property.getFloat64Value().toString());
   }
   else
   System.out.println(property.getFloat64ListValue().toString());
   break;
   case TypeID.LONG_AS_INT:
   if (propertyDescription.get_Cardinality().getValue() ==
   Cardinality.SINGLE_AS_INT)
   {
   if (property.getInteger32Value() != null)
   System.out.println(property.getInteger32Value().toString());
   }
   else
   System.out.println(property.getInteger32ListValue().toString());
   break;
   case TypeID.GUID_AS_INT:
   if (propertyDescription.get_Cardinality().getValue() ==
   Cardinality.SINGLE_AS_INT){
   if (property.getIdValue() != null)
   System.out.println(property.getIdValue().toString());
   }
   else
   System.out.println(property.getIdListValue().toString());
   break;
   default:
   System.out.println("");
   break;
   }
   System.out.println("-----------------------------------");
   }

public void getVersionsEventsEDU(Containable CEObject){
   if (CEObject instanceof Document){
      Document doc = (Document)CEObject;
      VersionableSet Docversions = doc.get_Versions();
      Document Docversion;
      Iterator it = Docversions.iterator();
      while (it.hasNext()){
         Docversion = (Document)it.next();
         getEventsEDU(Docversion);
      }
   }
}
```

```
   public Document getDocumentEDU(ObjectStore store, String folderPath){
      PropertyFilter pf = new PropertyFilter();
      pf.addIncludeProperty(0, null, null, PropertyNames.NAME,1);
      pf.addIncludeProperty(0, null, null, PropertyNames.AUDITED_EVENTS,1);
      pf.addIncludeProperty(0, null, null, PropertyNames.VERSIONS,1);
      pf.addIncludeProperty(0, null, null,
      PropertyNames.CLASS_DESCRIPTION,1);
      pf.addIncludeProperty(0, null, null,
      PropertyNames.MAJOR_VERSION_NUMBER,1);
      pf.addIncludeProperty(0, null, null,
      PropertyNames.MINOR_VERSION_NUMBER,1);
      Document document = Factory.Document.fetchInstance(store,folderPath,
      pf);
      String documentName = document.get_Name();
       System.out.println(documentName + "  Document is retrieved");
       return document;
   }
   public static void main(String[] args)
   {
      try {
         AuditEDU auditInstance = new AuditEDU();
         Connection conn =
         auditInstance.getCEConnectionEDU("p8admin","IBMFileNetP8");
         Domain domain = auditInstance.getDomainEDU(conn);
         ObjectStore store =
         auditInstance.getObjectStoreEDU(domain,"Development");
         //Document document =
         auditInstance.getDocumentEDU(store,"/APIFolder/APIAuditEvent.doc");
         //auditInstance.getEventsEDU(document);
         //auditInstance.getVersionsEventsEDU(document);
         //auditInstance.getDeletionEventsEDU(store, "p8admin");
         Document document = auditInstance.getDocumentEDU
         (store,"/APIFolder/CustomAPIEvent.doc");
         auditInstance.raiseCustomEventEDU(store,document,"CustomJavaAPIEven
         t");
      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```


Unit 12. Batches

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The Java class in this unit extends the CEConnectionEDU class from the “Communication with the Content Engine” unit.
- Concepts and some parts of the code from Documents unit is used.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt window.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. (These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.)

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 12.1. Batch update

Overview

Why is this lesson important to you?

The Product Development team creates content, adds it as a document, and files it in a folder. As their programmer, you are going to write code to perform a batch update operation for these tasks.

Prerequisites

- Activity , Change security on the folder, page 12-7

Activities

- Perform a batch update operation: Challenge, page 12-9
- Perform a batch update operation: Walkthrough, page 12-11

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	CEAdmin	filenet
Custom application	p8admin	IBMFileNetP8
Custom application	CEAdmin	filenet

Change security on the folder

Important

This activity is required for both skill levels.

In the next activity, you do a batch update. The updates include creating a Document object and filing it in a folder.

The batch update process is transactional. If the folder does not have the permissions for the user to file the object, the batch operation fails and the Document object is not created.

If these steps are executed without the batch update, the document is created, but filing it in the folder fails.

Use the following steps to change security on the folder:

1. Sign in to the Content Engine Enterprise Manager or Workplace.
2. Locate Development object store > Batches > Properties page and click the Security tab.
3. Under Type, select the Deny option to deny the permissions to the `p8admin`.
4. Click Apply and then OK to save the changes and close the window.

Perform a batch update operation: Challenge

Challenge

Use the data in the table to create a Java class that extends CEConnectionEDU and write code to do the following:

- Retrieve a folder to file a document.
- Execute a batch update that includes creating a document with content and filing it in a folder.

Data

Item	Value

Packages to import	`java.*`
	`java.util.Iterator`
	`com.filenet.api.collection.*`
	`com.filenet.api.constants.*`
	`com.filenet.api.core.*`
	`com.filenet.api.exception.*`
	`com.filenet.api.property.*`

Object store name	`"Development"`
Document title	`BatchAPI.gif`
Document class	`"Product"`
File name for the content	`"C:\\CMJavaAPIProg\\SampleDocuments\\BatchAPI.gif"`
Folder path to file the document	`"/Batches"`
Mime type	`"image/gif"`

Verification

Run the program using Eclipse or the Command Prompt and verify the following:

- You receive an error that batch operation cannot be executed due to insufficient access rights.
- Sign in as `CEAdmin` user and change security type on the Batches folder to allow the `p8admin` to have full control.
- Run the program again and verify that a document is created with the specified name and filed in the specified folder.

Sample output for the error

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
Batches  folder is retrieved

com.filenet.api.exception.EngineRuntimeException: E_ACCESS_DENIED: The
requester has insufficient access rights to perform the requested
operation.
```


Perform a batch update operation: Walkthrough

Procedures

Procedure 1: Create a Java class that extends CEConnectionEDU

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the “New Java Class” window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type `BatchEDU` as the class name in the Name field.
5. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
6. Select `public static void main(String[] args)` for “Which method stubs would you like to create?” option.
   a. Clear other options.
7. Click Finish to add the class and close the window.
   a. Verify that the new class is listed under your package in the Project Explorer.
8. Write code inside the class that you just added.

9. Import the following packages before the declaration of the class:
   - `java.io`
   - `java.util.Iterator`
   - `com.filenet.api.collection.*`
   - `com.filenet.api.constants.*`
   - `com.filenet.api.core.*`
   - `com.filenet.api.exception.*`
   - `com.filenet.api.property.*`
10. Inside the main method, write a `try-catch` block.
    a. Call `exception.printStackTrace()` in the catch block to display the exceptions thrown.
11. Add custom methods and call it from the main method as described in the following steps.

Procedure 2: Write a method to execute a batch update

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `executeBatchEDU`
   - Parameters:
     - `ObjectStore` object
     - `Folder` object
2. Retrieve the domain by calling `objectStore.get_Domain()`.
   a. Assign the return value to a variable of type `Domain`.
3. Call `UpdatingBatch.createUpdatingBatchInstance(...)`.
   - Parameters:
     - `Domain` object from step 2
     - `RefreshMode.REFRESH`
   a. Assign the return value to a variable of type `UpdatingBatch`.

Note

In Procedures 3 and 4, the first update creates a Document object and adds it to the batch. You can copy and paste the code for creating a document and setting content and properties from `DocumentsEDU.java` file that you created in the “Create Documents objects” lesson of the “Documents” unit. Use a different document title.

Procedure 3: First update to the batch: Create a Document object

1. Write the first update to be added to the batch by calling `Factory.Document.createInstance(...)`.
   - Parameters:
     - `ObjectStore` object
     - `"Product"`
   a. Assign the return value to a variable of type `Document`.
2. Call `Factory.ContentElement.createList()`.
   a. Assign the return value to a variable of type `ContentElementList`.
3. Call `Factory.ContentTransfer.createInstance()`.
   a. Assign the return value to a variable of type `ContentTransfer`.
4. Create a new instance of `FileInputStream` using the constructor.
   - Parameter: `"C:\\CMJavaAPIProg\\SampleDocuments\\BatchAPI.gif"`
   a. Assign the return value to a variable of type `FileInputStream`.
5. Call `ContentTransfer.setCaptureSource(...)`on the object from step 2.
   - Parameter: `FileInputStream` object that you got in the previous step
6. Call `ContentElementList.add(...)`.
   - Parameter: `ContentTransfer` object that you got in the previous step.
7. Call `document.set_ContentElements(...)`
   - Parameter: `ContentElementList` that you created in the previous steps.
8. Call `document.checkin(...)`.
   - Parameters:
     - `AutoClassify.DO_NOT_AUTO_CLASSIFY`
     - `CheckinType.MAJOR_VERSION`

Procedure 4: Set values for the properties of the Document object

1. Call `document.getProperties()`.
   a. Assign the return value to a variable of type `com.filenet.api.property.Properties`.
2. Call `properties.putValue(...)` to set value to the `DocumentTitle` property.
   - Parameters:
     - `"DocumentTitle"`
     - `"BatchAPI.gif"`

3. Repeat calling `properties.putValue(...)` to set value to the `product_id` property.
   - Parameters:
     - `"product_id"`
     - `"PDT101"`
4. Call `document.set_MimeType()` to set the `MimeType`.
   - Parameter: `"image/gif"`

Procedure 5: Second update to the batch: File the document into a folder

1. Write the second update to be added to the batch by calling `folder.file(...)` on the folder object that is passed into this method.
   - Parameters:
     - `Document` object that you created in procedures 3 and 4
     - `AutoUniqueName.AUTO_UNIQUE`
     - `"BatchAPI.gif"` for the containment name
     - `DefineSecurityParentage.DO_NOT_DEFINE_SECURITY_PARENTAGE`

   a. Assign the return value to a variable of type `ReferentialContainmentRelationship`.

Procedure 6: Add the updates to the batch and execute

1. Add the first update to the updating batch by calling `updatingBatch.add(...)`.
   - Parameters:
     - `Document` object that you created in procedures 3 and 4
     - `null`
2. Repeat step 1 to add the second update to the batch.
   - Parameters:
     - `ReferentialContainmentRelationship` object from procedure 5
     - `null`
3. Call `updatingBatch.updateBatch()` to execute the batch.
4. Optionally display a message that the Document object has been added.

Note

Notice that the changes to the objects are not explicitly saved using the save(...) method in this activity because the batch update persists the changes to the object store.

Procedure 7: Instantiate the class and call the methods that you wrote

1. Inside the `try` block of the main() method, create an instance of the class and assign the return value to a variable.

   Example: `BatchEDU batchInstance = new BatchEDU();`

2. Call the `getCEConnectionEDU(...)` method of the `CEConnectionEDU` class:
   - Parameters:
     - `"p8admin"`
     - `"IBMFileNetP8"`

   a. Assign the returned `Connection` object to a variable.

3. Call the `getDomainEDU(…)`method of the `CEConnectionEDU` class
   - Parameter: `Connection` object from step 2

   a. Assign the returned `Domain` object to a variable.

4. Call the `getObjectStoreEDU(…)`method of the `CEConnectionEDU` class:
   - Parameters:
     - `Domain` object from step 3
     - `“Development”` (name of the object store to be retrieved)

   a. Assign the returned `ObjectStore` object to a variable.

5. Call the `getFolderEDU(...)` method of the `CEConnectionEDU` class.
   - Parameters:
     - `Objectstore` variable from step 4
     - `"/Batches"`

Note

You wrote `getFolderEDU(...)` and `getObjectStoreEDU(...)` methods in `ConnectionEDU.Java` class.

6. Call the `executeBatchEDU(...)` method.
   - Parameters:
     - `Objectstore` variable from step 4
     - `Folder` variable from step 5

Procedure 8: Run the program and verify the results

1. Run the program using Eclipse or Command Prompt as instructed in the "Communication with the Content Engine Server" unit and verify the results.

- You receive an error that batch operation cannot be executed due to insufficient access rights.

Procedure 9: Change security on the folder and run the program

1. Sign in to Content Engine Enterprise Manager as `CEAdmin` user.
2. Locate Development object store > Batches > Properties page and click the Security tab.
3. Under Type, select the Allow option to give full control to the `p8admin`.
4. Click Apply and then click OK to save the changes and close the window.
5. Run the program again and verify that a document is created with the specified name and filed in the specified folder.

Sample output for the error

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
Batches  folder is retrieved

com.filenet.api.exception.EngineRuntimeException: E_ACCESS_DENIED: The
requester has insufficient access rights to perform the requested
operation.
```


Lesson 12.2. Batch retrieval

Overview

Why is this lesson important to you?

The billing department in your company wants to retrieve information for many customers. As their programmer, you are going to write code to retrieve this information in a batch operation to increase the performance.

Activities

Perform a batch retrieval operation: Challenge, page 12-19
Perform a batch retrieval operation: Walkthrough, page 12-21

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	CEAdmin	filenet
Custom application	p8admin	IBMFileNetP8
Custom application	CEAdmin	filenet

Perform a batch retrieval operation: Challenge

Challenge

Use the same Java class that you created in the previous lesson and the data in the table below to write code to do the following:

- Execute a batch retrieval of documents.
- Retrieve and display the properties of the documents that are retrieved.

Data

Item	Value
Object store name	`"Development"`
Folder path for document 1	`"/Research/Research Information Basic Model A.doc"`
Folder path for document 2	`"/Research/Research Information Basic Model C.doc`
Properties to retrieve and display	`Document name "product_id"`

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that a list of document names, document class names, and product Ids that are retrieved from the batch is displayed in the Command Prompt window or the console of the Eclipse.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
Batches  folder is retrieved
Batch retrieval executed successfully
==================================
1. Document name: Research Information Basic Model A.doc
  Document class: Product
  Product Id: B00A01
==================================
2. Document name: Research Information Basic Model C.doc
  Document class: Product
  Product Id: B00C01
```


Perform a batch retrieval operation: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to execute a batch retrieval

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `retrieveBatchEDU`
   - Parameter: `ObjectStore` object
2. Retrieve the domain by calling `objectStore.get_Domain()`.
   a. Assign the return value to a variable of type `Domain`.
3. Call `RetrievingBatch.createRetrievingBatchInstance(...)`.
   - Parameter: `Domain` object from step 2

   a. Assign the return value to a variable of type `RetrievingBatch`.
4. Retrieve the first document to be added to the batch by calling `Factory.Document.getInstance(...)`.
   - Parameters:
     - `Objectstore` object
     - `null`
     - `"/Research/Research Information Basic Model A.doc"`

   a. Assign the return value to a variable of type `Document`.
5. Repeat step 4 to get the second document to be added to the batch.
   - Parameters:
     - `Objectstore` object that is passed into this method
     - `null`
     - `"/Research/Research Information Basic Model C.doc"`
6. Add the first document to the RetrievingBatch instance by calling `retrievingBatch.add(...)`.
   - Parameter: `Document` from step 4

7. Repeat step 6 to add the second document.
   - Parameter: `Document` from step 5
8. Execute the batch by calling the `RetrievingBatch.RetrieveBatch()`.
   a. Display a message that "Batch retrieval executed successfully"

Procedure 2: Get batch item handles and display the values

1. Continue the code inside the same method from procedure 1.
   a. Get batch item handles by calling `retrievingBatch.getBatchItemHandles(...)`.
      - Parameter: `null`
   b. Assign the return list to a variable of type `List`.
2. Call `list.iterator()`on the object from step 1.
   a. Assign the returned value to a variable of the type `Iterator`.
3. Write a `while` loop to retrieve each BatchItemHandle in the list from step 1.

   ```
   while (iterator.hasNext())
   ```
4. Call `iterator.next()`within the while loop.
   a. Typecast the return value into a `BatchItemHandle` type and assign it to a variable.
5. Call `batchItemHandle.getObject()`to retrieve each object.
   a. Assign the return value to a variable of type `Document`.
6. Retrieve the class name of the object from step 5 and display the value by calling `System.out.println(…)`.
   - Parameter: `document.getClassName()`
7. Repeat step 6 to retrieve document name and product Id and display the values.
   a. Use the following methods to get the values:
      - `document.get_Name()`
      - `document.getProperties().getStringValue("product_id")`
8. Use an `if` statement to determine whether the batch item handle has any exception.

   ```
   batchItemHandle.hasException()
   ```
9. Inside the `if` statement, get the exception by calling `batchItemHandle.getException()`.
   a. Assign the return value to a variable of type `EngineRuntimeException`.
10. Get the message from the `EngineRuntimeException.get_Message()` method.
11. Display the message by calling `System.out.println(…)`.

Procedure 3: Call the method inside the main() method

1. Call the `retrieveBatchEDU(...)` method that you wrote inside the main() method.
   - Parameter: `ObjectStore` object that you retrieved in the previous activity
2. Comment out the method calls that are not tested in this activity.

Procedure 4: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt.
2. Verify that a list of document names, document class names, and product Ids that are retrieved from the batch is displayed in the Command Prompt window or the console of the Eclipse.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
Batches  folder is retrieved
Batch retrieval executed successfully
==================================
1. Document name: Research Information Basic Model A.doc
  Document class: Product
  Product Id: B00A01
==================================
2. Document name: Research Information Basic Model C.doc
  Document class: Product
Product Id: B00C01
```

Solution code for BatchEDU.java

```
package com.ibm.filenet.edu;

import java.io.*;
import java.util.Iterator;
import com.filenet.api.collection.*;
import com.filenet.api.constants.*;
import com.filenet.api.core.*;
import com.filenet.api.property.*;
import com.filenet.api.exception.*;

public class BatchEDU extends CEConnectionEDU {
   public void executeBatchEDU (ObjectStore objectStore, Folder
   folder)throws Exception
   {
      Domain domain = objectStore.get_Domain();
      UpdatingBatch ub = UpdatingBatch.createUpdatingBatchInstance(domain,
      RefreshMode.REFRESH);
      // First update to be included in batch.
      Document myDoc =
      Factory.Document.createInstance(objectStore,"Product");
      ContentElementList contentList = Factory.ContentElement.createList();
      ContentTransfer content = Factory.ContentTransfer.createInstance();
      FileInputStream file = new FileInputStream
      ("C:\\CMJavaAPIProg\\SampleDocuments\\BatchAPI.gif");
      content.setCaptureSource(file);
      contentList.add(content);
      myDoc.set_ContentElements(contentList);
      myDoc.checkin(AutoClassify.DO_NOT_AUTO_CLASSIFY,
      CheckinType.MAJOR_VERSION);
      com.filenet.api.property.Properties properties =
      myDoc.getProperties();
      properties.putValue("DocumentTitle", "BatchAPI.gif");
      properties.putValue("product_id","PDT101");
      myDoc.set_MimeType("image/gif");
      //Second update to be included in batch.
      ReferentialContainmentRelationship rel = folder.file(myDoc,
      AutoUniqueName.AUTO_UNIQUE,"BatchAPI.gif",
      DefineSecurityParentage.DO_NOT_DEFINE_SECURITY_PARENTAGE);
      ub.add(myDoc, null);
      ub.add(rel, null);
      ub.updateBatch();
      System.out.println ("Document object is created");
   }
```



```
public void retrieveBatchEDU(ObjectStore objectStore) {
   Domain domain = objectStore.get_Domain();
   RetrievingBatch   rb =
   RetrievingBatch.createRetrievingBatchInstance(domain);
   // Retrieve the objects to be added to the batch.
   Document doc1 = Factory.Document.getInstance(objectStore, null,
   "/Research/Research Information Basic Model A.doc");
   Document doc2 = Factory.Document.getInstance(objectStore,  null,
   "/Research/Research Information Basic Model C.doc");
   rb.add(doc1,  null);
   rb.add(doc2,  null);
   rb.retrieveBatch();
   System.out.println("Batch retrieval executed successfully");
   java.util.List batchItemHandles = rb.getBatchItemHandles(null);
   Iterator it = batchItemHandles.iterator();
   BatchItemHandle batchItemHandle;
   Document document;
   int count = 1;
   while (it.hasNext()){
      System.out.println("==================================");
      batchItemHandle = (BatchItemHandle)it.next();
      document = (Document)batchItemHandle.getObject();
      System.out.println(count + ". Document name: " +
      document.get_Name());
      System.out.println("  Document class: " + document.getClassName());
      System.out.println("  Product Id: " +
      document.getProperties().getStringValue("product_id"));
      count = count + 1;
      if(batchItemHandle.hasException()){
      // Displays the exception
         EngineRuntimeException thrown = batchItemHandle.getException();
         System.out.println("Exception: " + thrown.getMessage());
      }
   }
}

public Folder getFolderEDU(ObjectStore store, String folderName){
   Folder folder= Factory.Folder.fetchInstance(store,folderName, null);
   folderName = folder.get_FolderName();
   System.out.println(folderName + "  folder is retrieved");
   return folder;
}
```

```
public static void main(String[] args) {
   try{
      BatchEDU batchInstance = new BatchEDU();
      Connection conn = batchInstance.getCEConnectionEDU("p8admin",
      "IBMFileNetP8");
      Domain domain = batchInstance.getDomainEDU(conn);
      ObjectStore store =
      batchInstance.getObjectStoreEDU(domain,"Development");
      Folder folder = batchInstance.getFolderEDU(store, "/Batches");
      // batchInstance.executeBatchEDU (store, folder);
      batchInstance.retrieveBatchEDU(store);
   }
   catch (Exception e)
   {
      e.printStackTrace();
   }
   }
}
```


Unit 13. Annotations

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class in this unit extends the CEConnectionEDU class from the “Communication with the Content Engine” unit.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt window.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. (These files include the make.bat file for compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.)

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 13.1. Create Annotation objects

Overview

Why is this lesson important to you?

The sales team is reviewing documents and wants to add comments and footnotes as annotations. As their programmer, you are going to write code to create Annotation objects, set content, and associate them with the documents.

Prerequisites

Activity , Work with graphical annotations in Image Viewer, page 13-7

Activities

- Create Annotation objects: Challenge, page 13-9
- Create Annotation objects: Walkthrough, page 13-11

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Work with graphical annotations in Image Viewer

Important

This activity is required for both skill levels.

In this exercise, you use Image Viewer to manually create and view annotations that are graphic elements with and without text.

1. Add an image as a document using Workplace.
   a. Use the data from the following table.
   b. File the document in a folder.
   c. Add the document as a major version.

Item	Value
Name of the object store	`Development`
Document class	`Document`
Folder to file the document	`APIFolder`
Document title	`APIImageViewerDoc.gif`
Location of the source document	`C:\CMJavaAPIProg\SampleDocuments\APIImageViewerDoc.gif`

2. Use Image Viewer to view the image and add an annotation to it:
   a. Open the document (image) that you just created by clicking the document title.

      The Image Viewer applet loads and displays your image file.
   b. If a Warning - Security window is displayed, click No to close it.
   c. Use the tools provided in the left pane of the viewer to add some text and graphical annotations to the image.

      Example: a sticky note and a highlight
   d. Save the annotations and close the Image Viewer.
3. Verify the annotation of the document:
   a. Start Content Engine Enterprise Manager and log in as p8admin.
   b. Navigate to the `APIFolder` in the `Development` object store.
   c. Open the Properties page for the document `APIImageViewerDoc.gif` that you just added and click the Annotations tab.

d. Verify that the document has the annotations that you created:

- You see a GUID entry for each annotation.
- In the Annotation Content Elements section, you see an entry for the sticky note that you added.

Create Annotation objects: Challenge

Challenge

Use the data in the table to create a Java class that extends CEConnectionEDU. Write code to retrieve a document, create an Annotation object and associate it with a Document object.

Data

Item	Value

Packages to import	`java.io.FileInputStream`
	`java.util.Iterator`
	`com.filenet.api.collection.*`
	`com.filenet.api.constants.*`
	`com.filenet.api.core.*`
	`com.filenet.api.property.*`

Object store name	`"Development"`
Folder path for document	`"/APIFolder/APIImageViewerDoc.gif"`

Properties to retrieve for the document	`PropertyNames.ANNOTATIONS`
	`PropertyNames.MINOR_VERSION_NUMBER`
	`PropertyNames.MAJOR_VERSION_NUMBER`
	`PropertyNames.VERSIONS`

File path for annotation content	`"C:\\CMJavaAPIProg\\Source\\annotationText.txt"`

Verification

Run the program using Eclipse or the Command Prompt and verify the results:

- Start Content Engine Enterprise Manager and log in as p8admin.
- Locate the `APIFolder` in the Development object store.
- Open the Properties page for the document `APIImageViewerDoc.gif` and click the Annotations tab.
- Verify that the document has the annotations that you created.
- You see a GUID entry for each annotation.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the object store: Development
APIImageViewerDoc.gif  Document has been retrieved
{4A1954FD-A01C-49FE-9F18-E4047FC28495} Annotation is created
```


Create Annotation objects: Walkthrough

Procedures

Procedure 1: Create a Java class that extends CEConnectionEDU

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the "New Java Class" window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type `AnnotationsEDU` as the class name in the Name field.
5. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
6. Select `public static void main(String[] args)` for "Which method stubs would you like to create?" option.
   a. Clear other options.
7. Click Finish to add the class and close the window.
   a. Verify that the new class is listed under your package in the Project Explorer.
8. Write code inside the class that you just added.
9. Import the following packages before the declaration of the class:

```
java.util.Iterator
java.io.FileInputStream
com.filenet.api.core.*
com.filenet.api.collection.*
com.filenet.api.constants.*
com.filenet.api.property.*
```

10. Inside the main method, write a `try-catch` block.
    a. Call `exception.printStackTrace()` in the catch block to display the exceptions thrown.
11. Add custom methods and call it from the main method as described in the following steps.

Procedure 2: Write a method to retrieve a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getDocumentEDU`
   - Parameters:
     - `ObjectStore` object
     - `String` for the folder path
   - Returns: `Document` Object
2. Create a new instance of the `PropertyFilter` object.
   a. Assign the return value to a variable of type `PropertyFilter` object.
3. Call `PropertyFilter.addIncludeProperty(...)`.
   - Parameters:
     - `0`
     - `null`
     - `null`
     - `PropertyNames.NAME`
     - `1`
4. Repeat step 3 to include the following properties to the property filter:
   - `PropertyNames.ANNOTATIONS`
   - `PropertyNames.MINOR_VERSION_NUMBER`
   - `PropertyNames.MAJOR_VERSION_NUMBER`
   - `PropertyNames.VERSIONS`
5. Call `Factory.Document.fetchInstance(...)`.
   - Parameters:
     - `Objectstore` object that is passed into this method
     - Folder path that is passed into this method
     - `PropertyFilter` object that you got in steps 2 through 4
6. Get and display the document name by calling `System.out.println(…)`.
   - Parameter: `document.get_Name()`
7. Return the `Document` object.

Procedure 3: Write a method to create an annotation

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `createAnnotationEDU`
   - Parameters:
     - `ObjectStore` object
     - `Document` object
2. Call `Factory.Annotation.createInstance(...)`.
   - Parameters:
     - `ObjectStore` object that is passed into this method
     - `ClassNames.ANNOTATION`

   a. Assign the return value to a variable of type `Annotation`.
3. Call `Factory.ContentElement.createList()`.

   a. Assign the return value to a variable of type `ContentElementList`.
4. Call `Factory.ContentTransfer.createInstance()`.

   a. Assign the return value to a variable of type `ContentTransfer`..
5. Create a new instance of `FileInputStream` using the constructor.
   - Parameter: `"C:\\CMJavaAPIProg\\Source\\annotationText.txt"`

   a. Assign the return value to a variable of type `FileInputStream`.
6. Call `contentTransfer.setCaptureSource(...)`on the object from step 4.
   - Parameter: `FileInputStream` object from step 5
7. Call `contentTransfer.set_ContentType(...)`on the object from step 4.
   - Parameter: `"text/plain"`
8. Call `ContentElementList.add(...)`.
   - Parameter: `ContentTransfer` object from step 4.
9. Call `annotation.set_ContentElements(...)` on the object from step 2.
   - Parameter: `ContentElementList` that you created in the previous steps.
10. Call `annotation.set_AnnotatedObject(...)`.
    - Parameter: `Document` object that is passed into this method
11. Save the changes to the object by calling `annotation.save(...)` method.
    - Parameter: `RefreshMode.REFRESH`

12. Display the message `The annotation object is created successfully` by calling the `System.out.println(...)` method.
    a. Optionally, retrieve the Id of the annotation by calling `annotation.get_Id()` and display the value.

Procedure 4: Instantiate the class and call the methods that you wrote

1. Inside the `try` block of the main() method, create an instance of the class.

   Example: `AnnotationsEDU annotationsInstance = new AnnotationsEDU();`

2. Call the `getCEConnectionEDU(...)` method of the `CEConnectionEDU` class:
   - Parameters:
     - `"p8admin"`
     - `"IBMFileNetP8"`

   a. Assign the returned `Connection` object to a variable.

3. Call the `getDomainEDU(…)`method of the `CEConnectionEDU` class.
   - Parameter: `Connection` object from step 2

   a. Assign the returned `Domain` object to a variable.

4. Call the `getObjectStoreEDU(…)` method of the `CEConnectionEDU` class:
   - Parameters:
     - `Domain` object from step 3
     - `"Development"`

   a. Assign the returned `ObjectStore` object to a variable.

5. Call the `getDocumentEDU(...)` method.
   - Parameters:
     - `Objectstore` variable from step 4
     - `"/APIFolder/APIImageViewerDoc.gif"`

6. Call the `createAnnotationEDU(...)` method.
   - Parameters:
     - `Objectstore` variable from step 4
     - `Document` variable from step 5

Procedure 5: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt as instructed in the “Communication with the Content Engine Server” unit.
2. Verify the results in the Development object store using Content Engine Enterprise Manager:
   a. Locate the `APIFolder` in the Development object store.

b. Open the Properties page for the document and click the Annotations tab.

c. Verify that the document has the annotations that you created.

   You see a GUID entry for each annotation.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the object store: Development
APIImageViewerDoc.gif  Document has been retrieved
{4A1954FD-A01C-49FE-9F18-E4047FC28495} Annotation is created
```


Lesson 13.2. Retrieve annotations

Overview

Why is this lesson important to you?

The Sales team adds their comments to documents in the form of annotations. Anna, a Product Development team member, needs to retrieve their comments and incorporate them into the documents. As the programmer, you are going to write code to retrieve the annotation content when Anna accesses the comments.

Activities

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Retrieve annotations: Challenge

Challenge

Use the same Java class that you created in the previous lesson and the data in the table to write code to retrieve the annotations for a given document. Retrieve and display the name, creator, and the date created for each annotation.

Data

	Item	Value
	Object store name	`"Development"`
	Folder path for document	`"/APIFolder/APIImageViewerDoc.gif"`

Verification

Run the program and verify that a list of GUID values, creator and the date created for the annotations of the given document is displayed in the output window.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the object store: Development
APIImageViewerDoc.gif  Document has been retrieved
Annotation ={99CF477F-ED2C-4654-8902-010A588BA569}, created by p8admin on
Fri Mar 25 14:13:01 PDT 2011
Annotation ={983F86BA-AD88-4085-926F-D4137E1532BE}, created by p8admin on
Fri Mar 25 14:30:26 PDT 2011
Annotation ={4A1954FD-A01C-49FE-9F18-E4047FC28495}, created by p8admin on
Fri Mar 25 14:13:01 PDT 2011
```


Retrieve annotations: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to retrieve annotations for a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getAnnotationsEDU`
   - Parameter: `Document` object
2. Retrieve the annotations for the given document by calling `document.get_Annotations()`.
   a. Assign the return value to a variable of type `AnnotationSet`.
3. Call `annotationSet.iterator()`.
   a. Assign the returned value to a variable of the type `Iterator`.
4. Write a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```
5. Call `iterator.next()`within the `while` loop.
   a. Typecast the return value into an `Annotation` type and assign it to a variable.
6. Retrieve and display each annotation name by calling `System.out.println(…)`.
   - Parameter: `annotation.get_Name()`
7. Repeat step 6 to retrieve and display the creator and the date created for each annotation:
   a. Use each of the following parameters:
      - `annotation.get_Creator()`
      - `annotation.get_DateCreated.toString()`
8. Close the `while` loop and the method.

Procedure 2: Call the method inside the main() method

1. Call the `getAnnotationsEDU(...)` method inside the `try` block of the main() method.
   - Parameter: `Document` object that you retrieved in the previous lesson
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program and verify that a list of GUID values, creator, and the date created for the annotations of the given document is displayed in the Command prompt window or in the console of Eclipse.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the object store: Development
APIImageViewerDoc.gif  Document has been retrieved
Annotation ={99CF477F-ED2C-4654-8902-010A588BA569}, created by p8admin on
Fri Mar 25 14:13:01 PDT 2011
Annotation ={983F86BA-AD88-4085-926F-D4137E1532BE}, created by p8admin on
Fri Mar 25 14:30:26 PDT 2011
Annotation ={4A1954FD-A01C-49FE-9F18-E4047FC28495}, created by p8admin on
Fri Mar 25 14:13:01 PDT 2011
```


Lesson 13.3. Optional: Copy annotations

Overview

Why is this lesson important to you?

Anna, on the Product Development team, has modified documents based on the sales team's suggestions that were in the annotations. When Anna checks in the documents, her manager needs to view the sales recommendations as well as her revisions. Annotations are not automatically copied over to the next version of the document. As the programmer, you are going to write code to copy annotations from one version to the another.

Prerequisites

- Activity , Create a new version for the document, page 13-25

Activities

- Copy annotations: Challenge, page 13-27
- Copy annotations: Walkthrough, page 13-29

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Create a new version for the document

Important

This activity is required for both skill levels.

In the previous lesson, you created a document (version 1) and annotations for the document. In this exercise, you create version 2 of that document. Annotations are not automatically copied over to the next document version, so version 2 does not have annotations. In the next activity, you copy the annotations from version 1 to version 2 programmatically.

1. Start Content Engine Enterprise Manager.
2. Locate the `APIFolder` in the Development object store.
3. Check out (Exclusive Check Out) and check in the `APIImageViewerDoc.gif` document as a major version to create a new document version (version 2).
4. Verify that the new version of the document does not have annotations, but that version 1 does have annotations.
   a. Open the Properties page for the document and click the Annotations tab.

      You do not see any GUID entry for annotation.
   b. Click the Versions tab.
   c. Select version 1.0 (Superseded) and click Display Property Page.
   d. Click the Annotations tab.

      You see GUID entries for each annotation.
   e. After running the program for copying the annotations, you see annotations for both versions of the document.

Copy annotations: Challenge

Challenge

Use the same Java class that you created in the previous lesson. Use the data in the table to write code to copy annotations from one version of the document to another version. Optionally, display the annotations Ids that are copied.

Data

Item	Value
Object store name	`"Development"`
Folder path for document	`"/APIFolder/APIImageViewerDoc.gif"`
Version from which to copy the annotations	`1`

Verification

- Run the program using Eclipse or the Command Prompt.
- Use Content Engine Enterprise Manager to verify that the current version of the document has the annotations that you copied from the previous version.

Sample output

```
   Got the connection
   Name of the domain: P8Domain
   Name of the object store: Development
   APIImageViewerDoc.gif  Document is retrieved
   Document Version: 1 is retrieved
-------------------------------------
   Annotation from document version 1:
   1. {04EB5576-D0E2-4702-AD11-68CA94FF5066} annotation is copied
   -------------------------------------
   Annotation from document version 1:
   2. {C4483C76-19BF-4647-BB27-752909340B91} annotation is copied
-------------------------------------
```


Copy annotations: Walkthrough

Use the same Java class that you created in the previous lesson. In this activity, you write to code to copy annotations from one version of the document to another version.

Procedures

Procedure 1: Write a method to copy annotations

1. Define the method using the following signature:
   - Scope: `public`
   - Name: copyAnnotationsEDU
   - Parameters:
     - `ObjectStore` object
     - `Document` object
     - `int` for version number of the document from which you want to copy the annotation
2. Retrieve the versions for the given document from the `document.get_Versions()`.
   a. Assign the return value to a variable of type `VersionableSet`.
3. Write a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```

   a. Call `iterator.next()`within the while loop.
   b. Typecast the return value into a `Versionable` type and assign it to a variable.
   c. Retrieve the major version number by calling `versionable.get_MajorVersionNumber()` on the object from step 3b.
   d. Write an `if` statement to determine whether the version retrieved is the version from which you want to copy the annotations.
   e. Typecast the `Versionable` object from step 3b into a `Document` type and assign it to a variable.
   f. Display the version number by calling `System.out.println(…)`.
4. Retrieve the annotations from the document from step 3e by calling `document.get_Annotations()`.
   a. Assign the return value to a variable of type `AnnotationSet`.
   b. Write a second `if` statement to determine if the `AnnotationSet` from step a has annotations.

5. Call annotationSet.iterator().
   a. Assign the returned value to a variable of the type Iterator.
6. Write a second while loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```
7. Call iterator.next()within the while loop.
   a. Typecast the return value into an Annotation type and assign it to a variable.
8. Call annotation.get_ContentElements().get(0) on the object from step 7.
   a. Typecast the return value into a ContentTransfer type and assign it to a variable.
9. Create a new annotation object by calling Factory.Annotation.createInstance(...).
   - Parameters:
     - ObjectStore object that is passed into this method
     - ClassNames.ANNOTATION

   a. Assign the return value to a variable of type Annotation.
10. Call Factory.ContentElement.createList().
    a. Assign the return value to a variable of type ContentElementList.
11. Call Factory.ContentTransfer.createInstance().
    a. Assign the return value to a variable of type ContentTransfer.
12. Copy the content from the old annotation of the document version from step 7 to the newly created annotation object in step 9.
    a. Call contentTransfer.setCaptureSource(...)on the new annotation content from step 11.
       - Parameter: annotation.accessContentStream(0)on the old annotation from step 7
13. Call contentTransfer.set_RetrievalName(...)on the new annotation content from step 11.
    - Parameter: contentTransfer.get_RetrievalName() on the old annotation content from step 8
14. Call contentTransfer.set_ContentType(...)on the new annotation content from step 11.
    - Parameter: contentTransfer.get_ContentType() on the old annotation from step 8
15. Call contentElementList.add(...).
    - Parameter: ContentTransfer object from step 10

16. Call `annotation.set_ContentElements(...)` on the object from step 8.
    - Parameter: `ContentElementList` that you created in the previous steps
17. Call `annotation.set_AnnotatedObject(...)`.
    - Parameter: `Document` object that is passed into this method
18. Save the changes to the object by calling `annotation.save(...)` method.
    - Parameter: `RefreshMode.REFRESH`
19. Display the message `The annotation object is copied successfully` by calling the `System.out.println(...)` method.
    a. Optionally, display the annotations Ids that are copied by calling `annotation.get_Id()`.
    b. Close the second `while` loop.
    c. Close the second `if` statement.
20. Save the changes to the document by calling the `document.save(...)` method.
    - Parameter: `RefreshMode.NO_REFRESH`
    a. Close the first `if` statement.
    b. Close the first `while` loop.
    c. Close the method.

Procedure 2: Call the method inside the main() method

1. Call the copyAnnotationsEDU(...) method that you wrote inside the main() method.
    - Parameters:
      - `ObjectStore` object that you retrieved in the previous lesson
      - `Document` object that you retrieved in the previous lesson
      - `1` (Document version number from which you want to copy the annotation)
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program and verify the results using Content Engine Enterprise Manager.
    a. Locate the `APIFolder` in the Development object store.
    b. Open the Properties page for the document and click the Annotations tab.
    c. Verify that the current version of the document has the annotations that you copied from the previous version: You see a GUID entry for each annotation.

Sample output

```
   Got the connection
   Name of the domain: P8Domain
   Name of the object store: Development
   APIImageViewerDoc.gif  Document is retrieved
   Document Version: 1 is retrieved
   -------------------------------------
   Annotation from document version 1:
   1. {04EB5576-D0E2-4702-AD11-68CA94FF5066} annotation is copied
   -------------------------------------
   Annotation from document version 1:
   2. {C4483C76-19BF-4647-BB27-752909340B91} annotation is copied
-------------------------------------
```


Solution code for AnnotationsEDU.java

```
package com.ibm.filenet.edu;

import java.io.FileInputStream;
import java.util.Iterator;
import com.filenet.api.collection.*;
import com.filenet.api.constants.*;
import com.filenet.api.core.*;
import com.filenet.api.property.*;
public class AnnotationEDU extends CEConnectionEDU {
   public void createAnnotationEDU(ObjectStore os, Document document) throws
   Exception{
      try{
         Annotation myAnno =
         Factory.Annotation.createInstance(os,ClassNames.ANNOTATION);
         ContentElementList contentList =
         Factory.ContentElement.createList();
         ContentTransfer content = Factory.ContentTransfer.createInstance();
         content.setCaptureSource(new
         FileInputStream("C:\\CMJavaAPIProg\\Source\\annotationText.txt"));
         content.set_ContentType("text/plain");
         contentList.add(content);
         myAnno.set_ContentElements(contentList);
         myAnno.set_AnnotatedObject(document);
         myAnno.save(RefreshMode.REFRESH);
         System.out.println(myAnno.get_Id() + " Annotation is created");
      }
      catch (Exception e){
         e.printStackTrace();
      }

   public void getAnnotationsEDU(Document document)
   {
      AnnotationSet annos = document.get_Annotations();
      Iterator it = annos.iterator();
      while (it.hasNext()){
         Annotation anno = (Annotation)it.next();
         System.out.print("Annotation =" + anno.get_Name() );
         System.out.println(", created by " + anno.get_Creator() + " on " +
         anno.get_DateCreated().toString());
      }
   }
```

```
public Document getDocumentEDU(ObjectStore store, String folderPath){
   PropertyFilter pf = new PropertyFilter();
   pf.addIncludeProperty(0, null, null, PropertyNames.NAME,1);
   pf.addIncludeProperty(0, null, null, PropertyNames.VERSIONS,1);
   pf.addIncludeProperty(0, null, null, PropertyNames.ANNOTATIONS,1);
   pf.addIncludeProperty(0, null, null,
   PropertyNames.MINOR_VERSION_NUMBER,1);
   pf.addIncludeProperty(0, null, null,
   PropertyNames.MAJOR_VERSION_NUMBER,1);
   Document document = Factory.Document.fetchInstance(store,folderPath,
   pf);
   String documentName = document.get_Name();
   System.out.println(documentName + "  Document is retrieved");
   return document;
}
public void copyAnnotationEDU(ObjectStore os, Document document, int
versionNumber) throws Exception{
   VersionableSet versions = document.get_Versions();
   Versionable documentVersion;
   Document docCopy = null;
   Iterator versionableIt;
   versionableIt = versions.iterator();
   while (versionableIt.hasNext()){
      documentVersion = (Versionable)versionableIt.next();
      Integer majorNum = documentVersion.get_MajorVersionNumber();
      if (versionNumber == majorNum){
         docCopy = (Document)documentVersion;
         System.out.println("Document Version: " + majorNum + " is
         retrieved");
         AnnotationSet annotations = docCopy.get_Annotations();
         if (annotations != null){
         Iterator annotIt = annotations.iterator();
         int counter = 1;
         while (annotIt.hasNext()){
         Annotation annotation = (Annotation)annotIt.next();
         ContentTransfer oldContent = (ContentTransfer)
         annotation.get_ContentElements().get(0);
         Annotation newAnnotation =
         Factory.Annotation.createInstance(os,ClassNames.ANNOTATION);
         ContentElementList contentList =
         Factory.ContentElement.createList();
         DontentTransfer content =
         Factory.ContentTransfer.createInstance();
         content.setCaptureSource(annotation.accessContentStream(0));
         content.set_RetrievalName(oldContent.get_RetrievalName());
```



```
            content.set_ContentType(oldContent.get_ContentType());
            contentList.add(content);

            newAnnotation.set_ContentElements(contentList);
            newAnnotation.set_AnnotatedObject(document);
            newAnnotation.save(RefreshMode.REFRESH);
            System.out.println("Annotation from document version " +
            majorNum + ":");
            System.out.println(counter+ ". "+ annotation.get_Id()+ "
            annotation is copied");
            System.out.println("-------------------------------------");
            counter = counter + 1;
         }// while(annoIt)
         }// if(annotations)
            document.save(RefreshMode.NO_REFRESH);
         }//if(versionNumber)
      }//while (versionableIt
   } // copyAnnotationEDU

   public static void main(String[] args) {
      try{
         AnnotationEDU annotationsInstance  = new AnnotationEDU();
         Connection conn = annotationsInstance.getCEConnectionEDU("p8admin",
         "IBMFileNetP8");
         Domain domain = annotationsInstance.getDomainEDU(conn);
         ObjectStore store
         =annotationsInstance.getObjectStoreEDU(domain,"Development");
         Document document =
         annotationsInstance.getDocumentEDU(store,"/APIFolder/APIImageViewer
         Doc.gif");
         //annotationsInstance.createAnnotationEDU(store, document);
         //annotationsInstance.getAnnotationsEDU(document);
         annotationsInstance.copyAnnotationEDU(store,document,1);
      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```


Unit 14. Document Lifecycle

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class in this unit extends the CEConnectionEDU class from the “Communication with the Content Engine” unit.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt window.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. (These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.)

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 14.1. Change a document lifecycle state

Overview

Why is this lesson important to you?

The Product Development team in your company wants to create its content as document objects and then move these objects through four states: Draft, Approved, Published, and Archived. You, as their programmer, are going to write code to change a document lifecycle state.

Prerequisites

- Activity , Create a lifecycle policy and assign it to a Document class, page 14-7

Activities

- Change a document lifecycle state: Challenge, page 14-11
- Change a document lifecycle state: Walkthrough, page 14-13

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Create a lifecycle policy and assign it to a Document class

Important

This activity is required for both skill levels.

In this activity, you are going to create a new lifecycle policy and associate with a Document class. You are going to programmatically change the lifecycle state of a document in the following activity.

Procedures

Procedure 1: Create a lifecycle policy

1. Open the Content Engine Enterprise Manager and go to Object Stores > Development > Document Lifecycles > Document Lifecycle Policies.
2. Right-click Document Lifecycle Policies and click New Lifecycle Policy.

   The Create a Lifecycle Policy wizard opens.
3. Complete the wizard using the following steps:
   a. Click Next.
   b. In the Name and Describe the Lifecycle Policy window, type `APILifecyclePolicy` in the Name field.
   c. Click Next.
4. In the Specify the Lifecycle States window, add the first state of the document lifecycle.
   a. Click Add.
   b. In the Add New Lifecycle State window, type the first lifecycle state name from the data table. (Ensure that the two check boxes are **not** selected.)
   c. Click OK to close the window for the first lifecycle state.

5. Repeat step 4 to add the Approved, Published, and Archived states to the policy.
   a. Define the settings as specified in the data table.

Sequence	Lifecycle state name	Settings for the check boxes	
		Allow Demotion from State?	**Apply Security Template on Entry?**
1	Draft	No	No
2	Approved	Yes	No
3	Published	Yes	No
4	Archived	No	No

6. Verify that the sequence of the states as given in the table:
   a. If the order is different, use Move Up and Move Down as necessary to place these states into the correct sequence.
   b. If any one of the entries is not correct, you can select that item from the list and click Modify to correct it.
7. Click Next to go to the Assign a Document Lifecycle Action page.
8. Accept the default value of “none” for this policy.
9. Click Next and then click Finish.

Note

In the next lesson, you assign a Document Lifecycle Action to the policy.

Procedure 2: Assign the policy to a Document class

1. In Content Engine Enterprise Manager, expand the Object stores > Development > Document Class > LifeCycleDoc.
2. Right-click the LifeCycleDoc Document Class and click Properties.

   The Properties window opens.
3. On the General tab, select the `APILifecyclePolicy` that you created in procedure 1 from the Default Document Lifecycle Policy list.
4. Click Apply to save the changes.
   a. Click OK to close the Properties window.
5. Refresh the object store to reflect the changes.
   b. Select the Development object store node and click Refresh.

Procedure 3: Create a document

1. In Content Engine Enterprise Manager, expand the Object Stores > Development > Root Folder > APIFolder.
   a. Select the APIFolder, right-click, and click New Document.

      The Create New Document wizard opens.

   b. Complete the wizard using the data from the table.

Item	Value
Document class	LifeCycleDoc
Document title	APILifecycle.jpg
Location of the file for the content	C:\CMJavaAPIProg\SampleDocuments\APILifecycle.jpg
Add a major version	Yes

Procedure 4: Change the lifecycle state for a document

1. In Content Engine Enterprise Manager, expand the Object Stores > Development > Root Folder > APIFolder.
2. From the right-hand pane, select the APILifecycle.jpg that you just added, right-click, and click Properties.
3. In the Properties window, click the Lifecycle Policy tab, and then click Promote to advance the document from the Draft state to the Approved state.
4. Verify that the Lifecycle State field now shows the value as Approved.
5. Click OK to close the Properties window.

Change a document lifecycle state: Challenge

Challenge

Use the data in the table to create a Java class that extends CEConnectionEDU and write code to do the following:

- Retrieve the document that you created in the previous activity.
- Retrieve lifecycle policy and lifecycle states for the given document.
- Change the lifecycle state of the document (promote and then demote).

Data

Item	Value

Packages to import	`java.util.Iterator`
	`com.filenet.api.core.*`
	`com.filenet.api.events.*`
	`com.filenet.api.property.*`
	`com.filenet.api.constants.*`
	`com.filenet.api.collection.*`

Object store name	`"Development"`
Folder path for the document	`"/APIFolder/APILifecycle.jpg"`

Properties to retrieve for the document	`PropertyNames.NAME`
	`PropertyNames.CURRENT_STATE`
	`PropertyNames.DOCUMENT_LIFECYCLE_POLICY`

LifecycleChangeFlags value to promote	`LifecycleChangeFlags.PROMOTE`
LifecycleChangeFlags value to demote	`LifecycleChangeFlags.DEMOTE`

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that a list of possible lifecycle states for the document and current lifecycle state for the document (before and after changing the state) are displayed in the Command prompt window or in the console of Eclipse.
- Optionally, verify the current state of the document using Content Engine Enterprise Manager.

Sample output (Promote)

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APILifecycle.jpg  Document is retrieved
---------------------------------
Current state= Approved
Possible lifecycle states for this document:
  1. Draft
  2. Approved
  3. Published
  4. Archived
Promoted the lifecycle Status
Document state after the change = Published
```

Sample output (Demote)

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APILifecycle.jpg  Document is retrieved
---------------------------------
Current state= Published
Possible lifecycle states for this document:
  1. Draft
  2. Approved
  3. Published
  4. Archived
Demoted the lifecycle Status
Document state after the change = Approved
```


Change a document lifecycle state: Walkthrough

Procedures

Procedure 1: Create a Java class that extends CEConnectionEDU

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the “New Java Class” window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type `DocumentLifeCycleEDU` as the class name in the Name field.
5. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
6. Select `public static void main(String[] args)` for “Which method stubs would you like to create?” option.
   a. Clear other options.
7. Click Finish to add the class and close the window.
   a. Verify that the new class is listed under your package in the Project Explorer.

8. Write code inside the class that you just added.
9. Import the following packages before the declaration of the class:

```
java.util.Iterator
com.filenet.api.core.*
com.filenet.api.events.*
com.filenet.api.property.*
com.filenet.api.collection.*
com.filenet.api.constants.*
```

10. Inside the main method, write a `try-catch` block.
    a. Call `exception.printStackTrace()` in the catch block to display the exceptions thrown.
11. Add custom methods and call it from the main method as described in the following steps.

Procedure 2: Write a method to retrieve a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getDocumentEDU`
   - Parameters:
     - `ObjectStore` object
     - `String` for the folder path
   - Returns: `Document` Object
2. Create a new instance of `PropertyFilter` object.
   a. Assign the return value to a variable of type `PropertyFilter` object.
3. Call `PropertyFilter.addIncludeProperty(...)`.
   - Parameters:
     - `0`
     - `null`
     - `null`
     - `PropertyNames.NAME`
     - `1`
4. Repeat step 3 to include the following properties to the property filter:

```
PropertyNames.CURRENT_STATE
PropertyNames.DOCUMENT_LIFECYCLE_POLICY
```


5. Call `Factory.Document.fetchInstance(...)`.
   - Parameters:
     - `Objectstore` object that is passed into this method
     - Folder path that is passed into this method
     - `PropertyFilter` object that you got in steps 2 through 4
6. Get and display the document name by calling `System.out.println(…)`.
   - Parameter: `document.get_Name()`
7. Return the `Document` object.

Procedure 3: Write a method to change a document lifecycle state

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `changeLifeCycleEDU`
   - Parameters:
     - `Document` object
     - `LifecycleChangeFlags` object
2. Write code inside the method as described in procedures 4 through 8.

Procedure 4: Retrieve lifecycle policy for a document

1. Retrieve the lifecycle policy for the given document by calling the `document.get_DocumentLifecyclePolicy()` method.
   a. Assign the return value to a variable of type `DocumentLifecyclePolicy`.
2. Write an `if` statement to ensure that the document has a lifecycle policy by checking the return value in step 1.
   a. If the return value is `null`, display a message that the document does not have a policy.
   b. Close the `if` block and in the `else` block, write the code as in the following procedures.

Procedure 5: Retrieve current lifecycle state for a document

1. Retrieve the properties for the given document from `document.getProperties()`.
   a. Assign the return value to a variable of type `Properties`.
2. Call `properties.getStringValue(...)` to retrieve the current state of the given document.
   - Parameter: `PropertyNames.CURRENT_STATE`

   a. Assign the return value to a `String` variable.
   b. Display the value by calling `System.out.println(...)`.

Procedure 6: Retrieve the lifecycle states for a given policy

1. Retrieve the lifecycle states for the given policy by calling the `documentLifecyclePolicy.get_DocumentStates(...)` method on the object from procedure 4.
   a. Assign the return value to a variable of type `DocumentStateList`.
2. Call `documentStateList.iterator()`.
   a. Assign the returned value to a variable of the type `Iterator`.
3. Write a `while` loop and iterate through the document lifecycle states collection.

   ```
   while (iterator.hasNext())
   ```
4. Call `iterator.next()`within the while loop.
   a. Typecast the return value into a `DocumentState` type and assign it to a variable.
5. Display each document state name by calling `System.out.println(…)`.
   - Parameter: `documentState.get_StateName()` on the object from step 4

Procedure 7: Change the lifecycle state of a document

1. Write a second `if` statement to verify that the given state name from procedure 6 is the current state for the document.
   a. Call `documentState.get_CanBeDemoted().booleanValue()` within the `if` statement.
   b. Assign the returned value to a variable of the type `boolean`.
   c. Close the second `if` block.
   d. Close the `while` loop.
2. Write a third `if` statement to verify that the user wants to demote the document using the `LifecycleChangeFlags` parameter that was passed into this method.
   - Value to check: `LifecycleChangeFlags.DEMOTE`
3. Write the fourth `if` statement to verify that the `booleanValue` from step 1b is true in order to demote the document.
4. Inside the `if` statement, demote the document state by calling `document.changeState(...)`.
   a. Parameter: `LifecycleChangeFlags` that was passed into this method
   b. Optionally, display a message that the document state is demoted.
   c. Close the fourth `if` block.

5. If the booleanValue from step 1 is false, in the else block (for the fourth if statement in step 3), display a message that the document cannot be demoted.
   a. Close the fourth else block.
   b. Close the third if block.
6. If the user wants to promote the document, in the else block (for the third if statement in step 2), promote the document state by calling document.changeState(...).
   a. Parameter: LifecycleChangeFlags that was passed into this method
   b. Optionally, display a message that the document state is promoted.
   c. Close the third else block.

Procedure 8: Save the changes to the document and retrieve the current state

1. Call document.save(...).
   - Parameter: RefreshMode.REFRESH
2. Repeat procedure 5 to retrieve the current state of the document after the change has been made and display the new value.
   a. Close the first else statement.
   b. Close the method.

Procedure 9: Instantiate the class and call the methods that you wrote

1. Inside the try block of the main() method, create an instance of the class and assign the return value to a variable.

   Example: `DocumentLifeCycleEDU lifeCycleInstance = new DocumentLifeCycleEDU();`
2. Call the getCEConnectionEDU(...) method of the CEConnectionEDU class:
   - Parameters:
     - "p8admin"
     - "IBMFileNetP8"
   a. Assign the returned Connection object to a variable.
3. Call the getDomainEDU(…)method of the CEConnectionEDU class.
   - Parameter: Connection object from step 2
   a. Assign the returned Domain object to a variable.

4. Call the `getObjectStoreEDU(…)` method of the `CEConnectionEDU` class:
   - Parameters:
     - `Domain` object from step 3
     - `"Development"`

   a. Assign the returned `ObjectStore` object to a variable.
5. Call the `getDocumentEDU(...)` method.
   - Parameters:
     - `Objectstore` variable from step 4
     - `"/APIFolder/APILifecycle.jpg"`

   a. Assign the return value to a variable of type `Document`.
6. Call the `changeLifeCycleEDU(...)` method to promote a document.
   - Parameters:
     - `Document` object from step 5
     - `LifecycleChangeFlags.PROMOTE`
7. Repeat step 9 using the following parameters to demote a document:
   - `Document` object from step 5
   - `LifecycleChangeFlags.DEMOTE`

Procedure 10:Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt as instructed in the “Communication with the Content Engine Server” unit.
2. Verify that the following are displayed in the output window.
   - A list of possible lifecycle states for the document
   - Current lifecycle state for the document before and after changing the state
3. Optionally, verify the current state of the document using Content Engine Enterprise Manager.

Sample output

Refer to the sample output in Change a document lifecycle state: Challenge, page 14-11

Lesson 14.2. Create lifecycle actions

Overview

Why is this lesson important to you?

You are going to write Java classes that will be used in lifecycle actions to do the following:

- File a document to a folder when the document is promoted.
- Unfile the document from that folder when the document is demoted.
- Log information to a file when the document state changes.

Activities

This lesson has only walkthrough-level activities.

- Create a code module for a lifecycle action: Walkthrough, page 14-21
- Deploy the code module and test the lifecycle policy: Walkthrough, page 14-29

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8
DB2 database	dsrdbm01	IBMFileNetP8

Create a code module for a lifecycle action: Walkthrough

In this activity, you are going to write code for an lifecycle action, which creates a log file when the document state is changed. This code also updates a database or deletes a record in the database, based on the document state change.

Procedures

Procedure 1: Create a Java class that implements DocumentLifecycleActionHandler

1. Create a Java class using the instructions in the first lesson of this unit. Because this Java class is deployed as part of the lifecycle action, the main() method is not needed.
   - Package: `com.ibm.filenet.edu`
   - Declaration:
     - Scope: `public`
     - Name: `LifeCycleActionEDU`
     - Implements: `DocumentLifecycleActionHandler`
2. Import the following packages before the declaration of the class:

```
java.io
java.sql.*
com.filenet.api.engine.*
com.filenet.api.exception.*
com.filenet.api.events.*
com.filenet.api.core.*
com.filenet.api.constants.*
```

3. Add three custom methods and implement four methods as described in the following steps.

Procedure 2: Create a method to write data to a log file

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `logDataEDU`
   - Parameters:
     - `Document` object
     - `String` specifying what message to write to log file
   - Throws `IOexception`
2. Create a new instance of File and assign it to a variable:

   ```
   File file = new File(…);
   ```

   - Parameter: `"Logger.txt"`
3. Create a new instance FileWriter and assign it to a variable:

   ```
   new FileWriter(...)
   ```

   - Parameters:
     - `File` object created in step 2
     - `true`
4. Call `fileWriter.write()` on the object that you got in step 3.
   - Parameter: `"\r\n Name: "+doc.get_Name() + message + "\r\n"`
5. Call `fileWriter.close()`.

Information

Procedures 3 and 4 contain database APIs. They are not part of the IBM Filenet Content Engine APIs. They are given here to show how an external database can be updated using a lifecycle action of the Content Engine. These APIs are different for each version and type of the database server used.

Procedure 3: Create a method to update the database

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `updateProductEDU`
   - Parameters:
     - `String` for product_id property value
     - `String` for style property value
   - Throws `Exception`

2. Use `try-catch` blocks.
   a. Create a `catch` block for `Exception` and write the following inside the block:

      ```
      throw new EngineRuntimeException (ExceptionCode.DB_ERROR,
      ioException);
      ```

   b. Write code in the try block as instructed in the following steps.
3. Type the following line of code to register DB2 Server driver:

   ```
   Class.forName("com.ibm.db2.jcc.DB2Driver");
   ```

4. Call `DriverManager.getConnection()`
   - Parameters:
     - `"jdbc:db2://ccv01135:3737/ProdDB"`
     - `"dsrdbm01"`
     - `"IBMFileNetP8"`

   a. Assign the return value to a `java.sql.Connection` object variable.
5. Write an `if` statement to check if the connection is `null` and print out a message saying that the connection failed.
   a. Continue the code inside the else condition as described in the following steps.
6. Create a CallableStatement by calling the `connection.prepareCall(…)` method on the object from step 4.
   - Parameter: `"{CALL InsertProduct(?,?)}"`

   a. Assign the return value to a `CallableStatement` object variable.
7. Set the value for the `IN` parameter and execute the stored procedure:
   a. Call `callableStatement.setString(…)` on the object from step 6.
   - Parameters:
     - `1`
     - `product_id`

   b. Call the `callableStatement.setString(…)` method again.
   - Parameters:
     - `2`
     - `style`

   c. Call `callableStatement.execute()`.
   d. Close the `else` block.
   e. Close the `try` block.

Procedure 4: Create a method to delete a record in the database

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `deleteProductEDU`
   - Parameter: `String` for product_id property value
   - Throws `Exception`
2. Use `try-catch` blocks.
   a. Create a `catch` block for `Exception` and write the following inside the block:

      ```
      throw new EngineRuntimeException (ExceptionCode.DB_ERROR,
      ioException);
      ```

   b. Write the following code in the `try` block.
3. Type the following line of code to register DB2 Server driver:

   ```
   Class.forName("com.ibm.db2.jcc.DB2Driver");
   ```

4. Call `DriverManager.getConnection()`
   - Parameters:
     - `"jdbc:db2://ccv01135:3737/ProdDB"`
     - `"dsrdbm01"`
     - `"IBMFileNetP8"`

   a. Assign the return value to a `java.sql.Connection` object variable.
5. Write an `if` statement to check if the connection is `null` and print out a message saying that the connection failed.
   a. Continue the code inside the `else` condition as described in the following steps.
6. Create a CallableStatement by calling the `connection.prepareCall(…)` method on the object from step 4.
   - Parameter: `"{CALL DeleteProduct(?)}"`

   a. Assign the return value to a `CallableStatement` object variable.
7. Set the value for the `IN` parameter and execute the stored procedure:
   a. Call `callableStatement.setString(…)` on the object from step 6.
      - Parameters:
        - `1`
        - `product_id`

   b. Call `callableStatement.execute()`.
   c. Close the `else` block.
   d. Close the `try` block.

Procedure 5: Implement onDocumentPromote method

Use the same Java class that you created in the procedure 1.

1. Implement the onDocumentPromote method using the following signature:
   - Scope: `public`
   - Name: `onDocumentPromote`
   - Parameters:
     - `Document` object
     - `DocumentLifecyclePolicy` object
   - Throws `EngineRuntimeException`
2. Use `try-catch` blocks.
   a. Create a `catch` block for `IOException` and write the following inside the block:

   ```
   throw new EngineRuntimeException(ExceptionCode.E_FAILED, ioException);
   ```

   b. Repeat step 2a for `SQLException`.
   c. Write the following code in the `try` block.
3. Call the `logDataEDU(…)` method that you wrote earlier.
   - Parameters:
     - `Document` object
     - `"Document state is promoted to " + doc.get_CurrentState()`
4. Check to see if the document current state is `Archived`:

   ```
   if (doc.get_CurrentState().equalsIgnoreCase("Archived"))
   ```

   a. Write code inside the `if` statement as described in the following steps.
5. File the document to a folder:
   a. Call `document.getObjectStore().getObject()`on the `Document` object that is passed into this method.
      - Parameters:
        - `"Folder"`
        - `"/Archived"`
   b. Typecast the return value to a `Folder` object and assign it to a variable.
   c. Call `folder.file(…)` on the Folder object from the previous step.
      - Parameters:
        - `document`
        - `AutoUniqueName.AUTO_UNIQUE`
        - `document.get_Name()`
        - `DefineSecurityParentage. DO_NOT_DEFINE_SECURITY_PARENTAGE`
   d. Assign the return value to a `ReferentialContainmentRelationship` object.

e. Call `referentialContainmentRelationship.save(…)`.
   - Parameter: `RefreshMode.NO_REFRESH`

6. Call `deleteProductEDU(...)` to delete the record in the database.
   - Parameter: `doc.getProperties().getStringValue("product_id")`

   a. Close the `if` statement.

7. Check to see if the document current state is `Published`:

   ```
   else if (doc.get_CurrentState(). equalsIgnoreCase("Published"))
   ```

   a. Write code inside the `else if` statement as described in the following steps.

8. Call `updateProductEDU(...)` to update the record in the database.
   - Parameters:
     - `doc.getProperties().getStringValue("product_id")`
     - `doc.getProperties().getStringValue("style")`

   a. Close the `else if` statement.

Procedure 6: Implement onDocumentDemote method

Use the same Java class that you created in the procedure 1.

1. Implement the onDocumentDemote method using the following signature:
   - Scope: `public`
   - Name: `onDocumentDemote`
   - Parameters:
     - `Document` object
     - `DocumentLifecyclePolicy` object
   - Throws `EngineRuntimeException`

2. Use `try-catch` blocks.

   a. Create a `catch` block for IOException and write the following inside the block:

   ```
   throw new EngineRuntimeException(ExceptionCode.E_FAILED, iOException);
   ```

   b. Write the following code in the `try` block.

3. Call the `logDataEDU(…)` method that you wrote earlier.
   - Parameters:
     - `Document` object
     - `" Document state is demoted to " + doc.get_CurrentState()`

Procedure 7: Implement a method to reset lifecycle

Use the same Java class that you created in the procedure 1.

1. Implement the onDocumentResetLifecycle method using the following signature:
   - Scope: `public`
   - Name: `onDocumentResetLifecycle`
   - Parameters:
     - `Document` object
     - `DocumentLifecyclePolicy` object
   - Throws `EngineRuntimeException`
2. Use `try-catch` blocks.
   a. Create a `catch` block for IOException and write the following inside the block:

   ```
   throw new EngineRuntimeException(ExceptionCode.E_FAILED, iOException);
   ```

   b. Write the following code in the `try` block.
3. Call the `logDataEDU(…)` method that you wrote earlier.
   - Parameters:
     - `Document` object
     - `"Document state is reset to " + doc.get_CurrentState()`

Important

Although you are not writing any code for the onDocumentSetException and onDocumentClearException methods, you must implement these methods so that the DocumentLifecycleActionHandler works.

Procedure 8: Implement methods for exceptions

Use the same Java class that you created in the procedure 1.

1. Implement the onDocumentSetException method using the following signature:
   - Scope: `public`
   - Name: `onDocumentSetException`
   - Parameters:
     - `Document` object
     - `DocumentLifecyclePolicy` object
   - Throws `EngineRuntimeException`

2. Implement the onDocumentClearException method using the following signature:
   - Scope: public
   - Name: onDocumentClearException
   - Parameters:
     - Document object
     - DocumentLifecyclePolicy object
   - Throws EngineRuntimeException

Note

You use the compiled code from this activity in the next activity.

Deploy the code module and test the lifecycle policy: Walkthrough

Procedures

Procedure 1: Create a lifecycle action

Use the following instructions to create a lifecycle action and associate the Java code module that you wrote.

Wizard Field	Value
Name	`JavaAPILifecycleAction`
DocumentLifecycleActionHandler Java Class Name that you wrote	`com.ibm.filenet.edu.LifeCycleActionEDU`

1. In Content Engine Enterprise Manager, expand the Object Stores > Development > Document Lifecycles > Document Lifecycle Actions node.
2. Right-click the Document Lifecycle Actions node and click New Lifecycle Action.

   The Create a Lifecycle Action wizard starts.
3. Click Next.
4. In the Name and Describe the Lifecycle Action window, type the name for the lifecycle action as specified in the preceding table.
5. Click Next.
6. In the Specify the Lifecycle Action Handler window, identify the Java class to be used for the lifecycle actions:
   a. Type the DocumentLifecycleActionHandler Java Class Name from the preceding data table. (the name is case-sensitive).
   b. Select the Configure Code Module check box.
   c. Click Next.
7. In the “Specify the Code Module to be configured” window, specify the name of the Java class:
   a. Type `LifeCycleActionEDU` in the Code Module Title field.

b. Click Browse/Add.

c. If you used Eclipse to create the Java class, locate your working directory where this class is located.

   Example: `<Project folder>\classes\com\ibm\filenet\edu`

d. Otherwise, browse to the folder where you have stored your Java class.

   Example: `C:\CMJavaAPIProg\Source\com\ibm\filenet\edu`

e. Select the `LifeCycleActionEDU.class` file and click Open.

f. Click Next.

8. Verify the entries in the summary screen of the Create a Lifecycle Action wizard and click Finish.

   a. Click OK when you receive the confirmation message stating that you successfully created the JavaAPI Lifecycle Action.

      The Create a Lifecycle Action wizard closes.

9. Refresh the Development object store to reflect the changes.

Procedure 2: Create a lifecycle policy

Create a new lifecycle policy and associate the lifecycle action that you just created, as described in the “Create a lifecycle policy and assign it to a Document class” prerequisite activity in the first lesson of this unit, with the following changes:

1. In procedure 1, step 3b, type the name as `JavaAPILifecycleActionPolicy`.
2. In procedure 1, step 8, select `JavaAPILifecycleAction` from the list.
3. In procedure 2, step 1, select the Document Class > Product > LifeCycle Class.
4. In procedure 2, step 3, use the Lifecycle Policy that you created in this procedure.

Hint

The document class in the Procedure 2 is different from the one that was used in the previous activity.)

Procedure 3: Test the lifecycle policy and lifecycle action

Use the following instructions to test the lifecycle policy and lifecycle action code.

1. In Content Engine Enterprise Manager, select the Development object store node and click Refresh.
2. Expand the Development object store > Root Folder.
3. Right-click the APIFolder and click New Document.

The Create New Document wizard starts.

4. Click Next.
5. In the “Create a document object” window, do the following steps:
   a. In the Document Title field, type the name for the new document that is specified in the following table.
   b. Select the “Without content” option.
   c. Click Next.

Wizard field	Value
Document Title	`Test Lifecycle Action`
Document Class	`LifeCycle`
product_id	`NCH342`
style	`Deluxe`

6. In the Class and Properties window, select the `LifeCycle` Document class from the Class list.

   The list of properties changes to reflect the new document class.
7. Because the Java code module requires values for the `product_id` and `style`, type values for these two properties as specified in the preceding table, and click Create to complete the Create New Document Wizard.
8. Expand the APIFolder, right-click the document that you just added, and click Properties.
9. Select the Lifecycle Policy tab, and then click Promote.

   Name: `JavaAPILifecycleActionPolicy`
   a. Click Apply to save the changes.
   b. Leave this window open to perform the following steps.

Procedure 4: Verify the results

In this procedure, you are going to change the lifecycle state of the document several times by promoting, demoting, and resetting the document. Verify that each action causes logging to a file and updating or deleting the record in the database as you specified in the code module. When the document reaches the Archived state, a copy of the document is filed in the Archived folder.

1. After you promote the document to Approved state in the previous procedure, verify that the Lifecycle State field shows the document lifecycle state as Approved.
2. Verify that a log file is created on the server system:
   a. Access the Linux server system (`ccv01135`).

b. On the Linux desktop, select the computer icon, right click and click the Browse Folder from the context menu.

c. In the Computer – File Browser screen, double click on the File System on the left pane and navigate to the following location of the file in the right pane:

`"/opt/ibm/WebSphere/AppServer/profiles/AppSrv01/Logger.txt"`

d. Verify that the log file has the text that you added as a message.

e. Close the file.

3. Promote the document again to Published state and do the following steps:

a. Verify that the Logger.txt file has the message that the document state is promoted to Published.

4. On the server (ccv01135) system, open the DB2 database table to verify the update:

a. Right-click the server desktop and open a Terminal.

b. Type each of the following commands and press Enter:

- `xhost +`

  If you have not logged in as root, type the following commands and press Enter:

  - `sudo su -`
  - `filenet`
- `su - dsrdbm01`
- `db2cc`

The Control Center View for DB2 opens.

c. Accept the default settings. (Advanced)

d. Click OK.

e. In the left pane of the Control Center, expand All Databases > ProdDB > Tables.

f. Click the Products table at the top of the right pane.

g. Click Open at the bottom of the right pane.

h. In the Open Table Products screen, verify that a row with the values that you specified is added.

i. Close the Products table.

5. On the client image, promote the document again to Archived state and do the following steps.

a. On the server image, verify that the Logger.txt file has the message that the document state is promoted to Archived.

b. Use the instructions in step 3 to verify that the DB2 database record in the Products table is deleted.

c. Close the DB2 Control Center.

d. Verify that a copy of the document is filed in the Archived folder of the Development object store using the Content Engine Enterprise Manager.

6. Optionally, reset the document lifecycle state back to Draft and verify that the log file has the text.

Procedure 5: Optional: Update the lifecycle action with a new code module version

You might need to modify the Java code that you wrote for the lifecycle action after you have configured the code module. In this procedure, you are going to modify the code and update this Java class in the Content Engine by checking in the class as a new version of the code module. You also need to update the lifecycle action that references this code module.

1. Check out the code module:
   a. In Content Engine Enterprise Manager, expand Development object store > Root Folder > CodeModules.
   b. Select the CodeModules folder and check out (Exclusive) the code module in the right pane with the name that you created (`LifeCycleActionEDU`).
   c. Save the file when prompted.
2. Modify the Java action handler source and compile.
   a. Open the `LifeCycleActionEDU.java` file in Eclipse or any text editor.
   b. Change the name of the log file that the Java file creates in the `logDataEDU(...)` method to `LifeCycleModule.txt`.
   c. Save and compile the Java file.
3. Check in the new version of the code module and update the lifecycle action.
   a. In Content Engine Enterprise Manager, check in the code module.
   b. Refer to procedure 1, step 7 for the location of your Java class.
   c. After check in, right-click the code module and click Copy Object Reference.
   d. Open the Properties page of the lifecycle action that references the updated code module.
   e. On the Properties page, click the Properties tab and then select the All Properties option.
   f. Scroll down to the Code Module property at the bottom of the window.
   g. Right-click the Code Module value field and click Paste Object.
   h. In the Select Object from Paste Buffer window, select the latest version object reference and click OK.
   i. In the Properties window, click Apply to save the changes.

j. Click OK to close the window.

4. Test the modified Java code that has been updated in the code module and in the lifecycle action.

   a. Repeat procedure 3, steps 8 and 9 to promote or demote the document.

   b. Verify that a log file with the new name is created on the server system (ccv01135) in this location: "/opt/ibm/WebSphere/AppServer/profiles/APPSrv01/"

   c.

   d. Verify that the log file has the text that you added as a message.

Solution code for DocumentLifeCycleEDU.java

```
package com.ibm.filenet.edu;

import java.util.Iterator;
import com.filenet.api.events.*;
import com.filenet.api.collection.*;
import com.filenet.api.constants.*;
import com.filenet.api.core.*;
import com.filenet.api.property.*;

public class DocumentLifecycleEDU extends CEConnectionEDU {
   public void changeLifeCycleEDU(Document document, LifecycleChangeFlags
   lifeCycleFlag){
      DocumentLifecyclePolicy lifeCyclePolicy =
      document.get_DocumentLifecyclePolicy();
      if (lifeCyclePolicy == null){
      System.out.println("The document does not have lifecyle policy");
      }//first if
      else{
      Properties props = document.getProperties();
      String currentState
      =props.getStringValue(PropertyNames.CURRENT_STATE);
      System.out.println("--------------------------------");
      System.out.println("Current state= " + currentState);
      DocumentStateList lifeCycleStates =
      lifeCyclePolicy.get_DocumentStates();
      Iterator stateIt = lifeCycleStates.iterator();
      boolean bCanDemote = false;
      int count = 1;
      System.out.println("Possible lifecycle states for this document:");
      while (stateIt.hasNext()){
      DocumentState state = (DocumentState)stateIt.next();
      String stateName = state.get_StateName();
      System.out.println("  " + count + ". " + stateName);
      count = count + 1;
      if (stateName.equalsIgnoreCase(currentState)){
      bCanDemote = state.get_CanBeDemoted().booleanValue();
      } //second if
      }//while

      if (lifeCycleFlag.equals(LifecycleChangeFlags.DEMOTE)){
      if (bCanDemote){
      document.changeState(lifeCycleFlag);
```

```
System.out.println("Demoted the lifecycle Status");
} //fourth if
else {
System.out.println("Can not demote document");
}//fourth else
}//third if
else{
document.changeState(lifeCycleFlag);
System.out.println("Promoted the lifecycle Status");
}//third else
PropertyFilter pf = new PropertyFilter();
pf.addIncludeProperty(0, null, null, PropertyNames.CURRENT_STATE,1);
document.save(RefreshMode.REFRESH, pf);
//document.refresh();
currentState =
document.getProperties().getStringValue(PropertyNames.CURRENT_STATE);
System.out.println("Document state after the change = " +
currentState);
}//first else
}
public Document getDocumentEDU(ObjectStore store, String folderPath){
PropertyFilter pf = new PropertyFilter();
pf.addIncludeProperty(0, null, null, PropertyNames.NAME,1);
pf.addIncludeProperty(0, null, null, PropertyNames.CURRENT_STATE,1);
pf.addIncludeProperty(0, null, null,
PropertyNames.DOCUMENT_LIFECYCLE_POLICY,1);
Document doc = Factory.Document.fetchInstance(store,folderPath, pf);
String documentName = doc.get_Name();
System.out.println(documentName + "  Document is retrieved");
return doc;
}

public static void main(String[] args) {
try {
   DocumentLifecycleEDU lifeCycleInstance = new
   DocumentLifecycleEDU();
   Connection conn =
   lifeCycleInstance.getCEConnectionEDU("p8admin","IBMFileNetP8");
   Domain domain = lifeCycleInstance.getDomainEDU(conn);
   ObjectStore store =
   lifeCycleInstance.getObjectStoreEDU(domain,"Development");
   Document document =
   lifeCycleInstance.getDocumentEDU(store,"/APIFolder/APILifecycle.jpg
   ");
```



```
            lifeCycleInstance.changeLifeCycleEDU(document,
            LifecycleChangeFlags.DEMOTE);
            lifeCycleInstance.changeLifeCycleEDU(document,
            LifecycleChangeFlags.PROMOTE);
         } catch (Exception e) {
         e.printStackTrace();
      }
      }
}
```

Solution code for LifeCycleActionEDU.java

```
package com.ibm.filenet.edu;
import java.io.*;
import java.sql.*;
import com.filenet.api.core.*;
import com.filenet.api.engine.*;
import com.filenet.api.exception.*;
import com.filenet.api.constants.*;
import com.filenet.api.events.*;
public class LifeCycleActionEDU implements DocumentLifecycleActionHandler {
public void onDocumentPromote( Document doc, DocumentLifecyclePolicy policy
throws EngineRuntimeException{
   try{
      logDataEDU(doc, " Document state is promoted to " +
      doc.get_CurrentState());
      if (doc.get_CurrentState().equalsIgnoreCase("Archived")){
         Folder folder = (Folder)doc.getObjectStore().getObject("Folder",
         "/Archived");
         ReferentialContainmentRelationship rel =
         folder.file(doc,AutoUniqueName.AUTO_UNIQUE,doc.get_Name(),DefineSec
         urityParentage.DO_NOT_DEFINE_SECURITY_PARENTAGE);
         rel.save(RefreshMode.NO_REFRESH);
         deleteProductEDU(doc.getProperties().getStringValue("product_id"));
      }
      else if (doc.get_CurrentState().equalsIgnoreCase("Published")){
         updateProductEDU(doc.getProperties().getStringValue("product_id"),
         doc.getProperties().getStringValue("style"));
      }
   }
   catch (IOException e){
   throw new EngineRuntimeException (ExceptionCode.E_FAILED, e);
   }
   catch (SQLException e){
   throw new EngineRuntimeException (ExceptionCode.DB_ERROR, e);
   } catch (Exception e) {
   // TODO Auto-generated catch block
   e.printStackTrace();
   }
}
public void onDocumentDemote(Document doc, DocumentLifecyclePolicy policy )
throws EngineRuntimeException {
try {
logDataEDU(doc, " Document state is demoted to " + doc.get_CurrentState());
```



```
}
catch (IOException e){
throw new EngineRuntimeException (ExceptionCode.E_FAILED, e);
}
}

public void onDocumentSetException( Document doc, DocumentLifecyclePolicy
policy ) throws EngineRuntimeException {
}
public void onDocumentClearException( Document doc, DocumentLifecyclePolicy
policy )throws EngineRuntimeException {
}

public void onDocumentResetLifecycle(Document doc, DocumentLifecyclePolicy
policy )throws EngineRuntimeException{
try{
logDataEDU(doc, " Document state is reset to " + doc.get_CurrentState());
}
catch (IOException e){
throw new EngineRuntimeException (ExceptionCode.E_FAILED, e);
}
}

private void logDataEDU(Document doc, String message) throws IOException
{
// Setup FileWriter
File file = new File("Logger.txt");
FileWriter fileWriter = new FileWriter(file, true); // true = append
fileWriter.write("\r\n Name: " + doc.get_Name() + message + "\r\n");
// Close FileWriter
fileWriter.close();
}

public void updateProductEDU(String product_id, String style) throws
Exception{
try {
CallableStatement cs;
Class.forName("com.ibm.db2.jcc.DB2Driver");
java.sql.Connection connection = DriverManager.getConnection
("jdbc:db2://ccv01135:3737/ProdDB", "dsrdbm01", "IBMFileNetP8");
if (connection == null){
System.out.println("Connection failed.");
} // if
else {
```

```
cs = connection.prepareCall("{CALL InsertProduct(?,?)}");

// Set the value for the IN parameter
cs.setString(1, product_id);
cs.setString(2, style);

// Execute the stored procedure
cs.execute();
}//else
}//try

catch (SQLException e){
throw new EngineRuntimeException (ExceptionCode.DB_ERROR, e);
}
}

public void deleteProductEDU(String product_id) throws Exception{
try {
CallableStatement cs;
Class.forName("com.ibm.db2.jcc.DB2Driver");
java.sql.Connection connection = DriverManager.getConnection
("jdbc:db2://ccv01135:3737/ProdDB", "dsrdbm01", "IBMFileNetP8");
if (connection == null){
System.out.println("Connection failed.");
} // if
else{
cs = connection.prepareCall("{call DeleteProduct(?)}");
// Set the value for the IN parameter
cs.setString(1, product_id);
// Execute the stored procedure
cs.execute();
} //else
}//try

catch (SQLException e){
throw new EngineRuntimeException (ExceptionCode.DB_ERROR, e);
}
}

}
```


Unit 15. Publishing

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class in this unit extends the CEConnectionEDU class from the “Communication with the Content Engine” unit.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt window.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. (These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.)

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 15.1. Publish a document

Overview

Why is this lesson important to you?

The Marketing department creates documents that describe their products. When the documents are ready to be put on the company web site, the Marketing department needs to publish the documents so that they are filed in the proper folder, with the proper security. As the programmer, you are going to write code to publish a document.

Activities

- Publish a document: Challenge, page 15-7
- Publish a document: Walkthrough, page 15-9

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Publish a document: Challenge

Challenge

Use the data in the table to create the following:

- Folders for storing publish template and published documents
- A document to publish
- A publish template using Workplace
- A Java class that extends CEConnectionEDU and write code to publish a document

Data

Item	Value

Packages to import	`java.util.*`
	`com.filenet.api.core.*`
	`com.filenet.api.constants.*`
	`com.filenet.api.collection.*`
	`com.filenet.api.property.*`
	`com.filenet.api.query.*`
	`com.filenet.api.publishing.*`

Object store name	`"Development"`
Name of the folder for the publications	`PublishedDocuments`
Name of the folder to file the publish template in	`PublishTemplates`
Name of the folder to file the source documents in	`APIFolder`
Name of the document to create	`APIPublish.jpg`
Location for the source documents	`C:\CMJavaAPIProg\SampleDocuments`
Folder path of the document to retrieve	`"/APIFolder/APIPublish.jpg"`

Document properties to retrieve	`PropertyNames.NAME`
	`PropertyNames.DEPENDENT_DOCUMENTS`
	`PropertyNames.DESTINATION_DOCUMENTS`

Name and path of the publish template	`""/PublishTemplates/APIPublishTemplate"`

Sample XML String	`"<publishoptions><publicationname>" +_`
	`document.get_Name()+"_Publication_`
	`</publicationname></publishoptions>"`

Verification

- Run the program using Eclipse or the Command Prompt.
- Locate the PublishedDocuments folder and verify that the document is published with the name that you specified using Content Engine Enterprise Manager or Workplace.

Note

Sometimes, the published document may not be listed. Refresh the browser if you do not see your document published.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIPublish.jpg  Document is retrieved
PUBLISHING_IN_QUEUE
The document is published
```


Publish a document: Walkthrough

Procedures: Create folders and files

Procedures: Publish a document

Procedure 1: Create folders

1. Sign in to IBM FileNet Workplace as p8admin.
2. Locate the following folders on the Development object store:
   - PublishTemplates
   - PublishedDocuments
3. If these folders do not exist, create them. (They are used in this lab.)

Procedure 2: Create the publish template

1. Open Workplace and sign in as p8admin.
2. Open Publishing Designer Wizard by clicking Author > Advanced Tools in the left pane and then clicking Publishing Designer link in the right pane.
3. If a Warning - Security window is displayed, click Run to close it.
4. On the Publish Instructions tab, select Development from the list for the “Object Store to host publish operations using this template” field.
5. Specify the location to add the published documents.
   a. Click Browse next to the “Destination Folder” field.
   b. Select PublishedDocuments and click Select.
6. Accept the default values for all other tabs.

7. Save the publish template to the Development object store:
   a. Click File > Add New.
   b. In the ‘Add a new template” window, click Browse and select the PublishTemplates folder to store the template.
   c. Click Next.
   d. Type `APIPublishTemplate` in the Document Title field for the title of the publish template.
   e. Click Next.
   f. In the Security Settings window, select the P8Admin user, and click Modify.
   g. Select the Allow - Owner Control option and click OK.
   h. Click Finish.

Important

P8Admin user need to have Allow - Owner Control on the publish template to be able to execute the code in this unit.

8. Close Publishing Designer by clicking File > Exit.

Procedure 3: Create a document to test the publish template

1. Open the Workplace and verify the following document exists.
2. If not, add a new document to the object store using the “Add Document” wizard and the following data table.

Item	Value
Object store name	`"Development"`
Document class	`Document`
Document title	`APIPublish.jpg`
Location of the file for the content	`C:\CMJavaAPIProg\SampleDocuments`
Folder name to file in	`APIFolder`
Add as Major Version	`Yes`

Note

You can also use an existing document from the object store for publishing.

Publish a document

Procedure 1: Create a Java class that extends CEConnectionEDU

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the “New Java Class” window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type `PublishEDU` as the class name in the Name field.
5. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
6. Select `public static void main(String[] args)` for “Which method stubs would you like to create?” option.
   a. Clear other options.
7. Click Finish to add the class and close the window.
   a. Verify that the new class is listed under your package in the Project Explorer.
8. Write code inside the class that you just added.
9. Import the following packages before the declaration of the class:

```
java.util.*
com.filenet.api.core.*
com.filenet.api.constants.*
com.filenet.api.collection.*
com.filenet.api.property.*
com.filenet.api.query.*
com.filenet.api.publishing.*
```

10. Inside the main method, write a `try-catch` block.
    a. Call `exception.printStackTrace()` in the catch block to display the exceptions thrown.
11. Add custom methods and call it from the main method as described in the following steps.

Procedure 2: Write a method to retrieve a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getDocumentEDU`
   - Parameters:
     - `ObjectStore` object
     - `String` for the folder path
   - Returns: `Document` Object
2. Create a new instance of `PropertyFilter` object.
   a. Assign the return value to a variable of type `PropertyFilter` object.
3. Call `PropertyFilter.addIncludeProperty(...)`.
   - Parameters:
     - `0`
     - `null`
     - `null`
     - `PropertyNames.NAME`
     - `1`
4. Repeat step 3 to include the following properties to the property filter:

   `PropertyNames.DEPENDENT_DOCUMENTS`
   `PropertyNames.DESTINATION_DOCUMENTS`
5. Call `Factory.Document.fetchInstance(...)`.
   - Parameters:
     - `Objectstore` object that is passed into this method
     - Folder path that is passed into this method
     - `PropertyFilter` object from step 2
6. Get and display the document name by calling `System.out.println(…)`.
   - Parameter: `document.get_Name()`
7. Return the `Document` object.

Procedure 3: Write a method to publish a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `publishDocumentEDU`
   - Parameters:
     - `Document` object
     - `ObjectStore` object
     - `String` for folder path with publish template name

2. Call `objectStore.fetchObject(...)`.
   - Parameters:
     - `ClassNames.PUBLISH_TEMPLATE`
     - `String` value for folder path with publish template name
     - `null`

   a. Typecast the return value into a `PublishTemplate` type and assign it to a variable.
3. Create an XML string for publication name like the one in the following sample:

   ```
   "<publishoptions><publicationname>" + document.get_Name() +_
   "_Publication + </publicationname></publishoptions>"
   ```

   a. Assign the return value to a variable of type `String`.
4. Call `document.publish(...)`.
   - Parameters:
     - `PublishTemplate` object from step 2
     - `String` value with the publish name from step 3

   a. Assign the return value to a variable of type `PublishRequest`.
5. Create a new instance a `PropertyFilter` object.

   a. Assign the return value to a variable of type `PropertyFilter` object.
6. Call `PropertyFilter.addIncludeProperty(...)`.
   - Parameters:
     - `0`
     - `null`
     - `null`
     - `PropertyNames.PUBLISHING_STATUS`
     - `1`
7. Call `publishRequest.save(...)`.
   - Parameters:
     - `RefreshMode.REFRESH`
     - `PropertyFilter` value from step 5
8. Retrieve the publishing status by calling `publishRequest.get_PublishingStatus().toString()`.

   a. Display the publishing status by calling `System.out.println(...)`.

Procedure 4: Instantiate the class and call the methods inside the main() method

1. Inside the `try` block of the main() method, create an instance of the class.

   Example: `PublishEDU publishInstance = new PublishEDU();`

2. Call the getCEConnectionEDU(...) method of the CEConnectionEDU class:
   - Parameters:
     - "p8admin"
     - "IBMFileNetP8"

   a. Assign the returned Connection object to a variable.
3. Call the getDomainEDU(…)method of the CEConnectionEDU class.
   - Parameter: Connection object variable from step 2

   a. Assign the returned Domain object to a variable.
4. Call the getObjectStoreEDU(…) method of the CEConnectionEDU class:
   - Parameters:
     - Domain object variable from step 3
     - "Development"

   a. Assign the returned ObjectStore object to a variable
5. Call the getDocumentEDU(...) method.
   - Parameters:
     - ObjectStore object from step 4
     - "/APIFolder/APIPublish.jpg"

   a. Assign the return value to a variable of type Document.
6. Call the publishDocumentEDU(...) method.
   - Parameters:
     - Objectstore variable from step 5
     - Document object from step 4
     - "/PublishTemplates/APIPublishTemplate"

Procedure 5: Run the program and verify the results

1. Run the program using Eclipse or the Command Prompt as instructed in the "Communication with the Content Engine Server" unit.
2. Verify the results in the Development object store using Content Engine Enterprise Manager or Workplace:
   a. Browse to the PublishedDocuments folder.
   b. Verify that the document is published with the name that you specified.

Note

Sometimes, the published document may not be listed. Refresh the browser if you do not see your document published.

Sample output

Refer to the sample output in Publish a document: Challenge, page 15-7

Lesson 15.2. Republish a document

Overview

Why is this lesson important to you?

The Marketing department has revised a document that describes their latest product update. Now they need to publish the document again to the company web site. The document must be filed in the same folder and must have the same security applied as the previous publication. As the programmer, you are going to write code to republish the document and use the same publish template.

Activities

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Republish a document: Challenge

Challenge

Use the same Java class that you created in the previous lesson and write code to republish a document.

Data

Item	Value
Object store name	`"Development"`
Folder path for the document	`"/APIFolder/APIPublish.jpg"`
Sample XML String	`"<publishoptions><publicationname>"+ _ document.get_Name()+"_Republished_ </publicationname></publishoptions>"`
Publications property of the document	`APIPublish.jpg_Publication` `APIPublish.jpg_Republished`

Verification

Run the program using Eclipse or the Command Prompt and verify the results using Workplace:

- Locate the PublishedDocuments folder.
- Verify that the document is republished with the name that you specified.
- Use Content Engine Enterprise Manager to locate the following file and view its system properties:
  APIFolder > APIPublish.jpg
- Verify that Publications property has the following values for the publication and republication document names.
  - APIPublish.jpg_Publication
  - APIPublish.jpg_Republished

Note

The Content Engine Enterprise Manager displays the containment name, so republication is displayed as APIPublish.jpg_Publication instead of APIPublish.jpg_Republished. You can check the title in the Properties page)

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIPublish.jpg  Document is retrieved
publication used = APIPublish.jpg_Publication
PUBLISHING_IN_QUEUE
The document is republished
```


Republish a document: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to republish a document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `rePublishDocumentEDU`
   - Parameters:
     - `Document` object
     - `ObjectStore` object
     - `String` for publication name
2. Retrieve the publications for the given document by calling `document.get_DestinationDocuments()`.
   a. Assign the return value to a variable of type `DocumentSet`.
3. Call `documentSet.iterator()`.
   a. Assign the returned value to a variable of the type `Iterator`..
4. Write a `while` loop to retrieve each publication from the collection.
5. Call `iterator.next()`within the while loop.
   a. Typecast the return value into a `Document` type and assign it to a variable.
6. Inside the `while` loop, use an `if` statement to check for the publication name that you are interested in.
   a. Write a second `if` statement to ensure that there is a publication.
7. Inside the `if` statement, display the publication name that is used to republish by calling `System.out.println(…)`.
   - Parameter: `document.get_Name()`
8. Create an XML string for the republication name as shown in the following sample:

   ```
   "<publishoptions><publicationname>" + document.get_Name() +_
   “_Republished + </publicationname></publishoptions>"
   ```

   a. Assign the return value to a variable of type `String`.

9. Call `document.republish(...)`.
   - Parameters:
     - `Document` object from step 6 (publication)
     - `String` value from step 8
   a. Assign the return value to a variable of type `PublishRequest`.
10. Create a new instance a `PropertyFilter` object.
    a. Assign the return value to a variable of type `PropertyFilter` object.
11. Call `PropertyFilter.addIncludeProperty(...)`.
    - Parameters:
      - `0`
      - `null`
      - `null`
      - `PropertyNames.PUBLISHING_STATUS`
      - `1`
12. Call `publishRequest.save(...)`.
    - Parameters:
      - `RefreshMode.REFRESH`
      - `PropertyFilter` value from steps 7 and 8
13. Retrieve the publishing status from the `publishRequest.get_PublishingStatus().toString()`.
    a. Display the publishing status by calling `System.out.println(…)`.

Procedure 2: Call the method inside the main() method

1. Call the `rePublishDocumentEDU(...)` method inside the main() method.
   - Parameters:
     - `Document` object that you retrieved in the previous activity
     - `"APIPublish.jpg_Publication"` (the name that you used for publishing the document in the previous activity)
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results:

1. Run the program and verify the results using the verification steps in Republish a document: Challenge, page 15-19

Sample output

Refer to the sample output in Republish a document: Challenge, page 15-19

Optional: Retrieve publish templates: Challenge

Challenge

Use the same Java class that you created in the previous lesson and the data in the following tables and write code to retrieve the publish templates from a given object store.

Data

Item	Value
Object store name	"Development"

SQL statement for retrieving the publish templates

```
sql = "SELECT pt.Name, pt.Id, pt.DateCreated, pt.StyleTemplate”
sql += "FROM PublishTemplate AS pt "
sql += "WITH INCLUDESUBCLASSES LEFT OUTER JOIN PublishStyleTemplate AS st"
sql += " WITH INCLUDESUBCLASSES ON pt.StyleTemplate = st.This"
sql += " WHERE (pt.IsCurrentVersion = true) ORDER BY pt.DateCreated"
```

Verification

- Run the program using Eclipse or the Command Prompt.
- Verify that a list of publish templates names is displayed in the Command prompt window or in the console of Eclipse.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
publish template = APIPublishTemplate
publish template = SamplePublishTemplate
```


Optional: Retrieve publish templates: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to retrieve publish templates

This method retrieves the publish templates from a given object store.

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getPublishTemplatesEDU`
   - Parameter: `ObjectStore` object
2. Build an SQL statement using the data in the table below and assign it to a `String`.

SQL statement for retrieving the publish templates

```
sql = "SELECT pt.Name, pt.Id, pt.DateCreated, pt.StyleTemplate”
sql += "FROM PublishTemplate AS pt "
sql += " WITH INCLUDESUBCLASSES LEFT OUTER JOIN PublishStyleTemplate AS st "
sql += " WITH INCLUDESUBCLASSES ON pt.StyleTemplate = st.This "
sql += " WHERE (pt.IsCurrentVersion = true) ORDER BY pt.DateCreated"
```

3. Create an Instance of SearchScope class:

   `new SearchScope(...)`

   - Parameter: `ObjectStore` object

   a. Assign the return value to a variable of type `SearchScope`.
4. Create an Instance of SearchSQL class:

   `new SearchSQL(...)`

   - Parameter: `String` value for SQL statement from step 2.

   a. Assign the return value to a variable of type `SearchSQL`.

5. Call SearchScope.fetchObjects(...).
   - Parameters:
     - SearchSQL object from step 4
     - Integer.valueOf("50")
     - null
     - Boolean.valueOf(true)

   a. Assign the return value to a variable of type DocumentSet.
6. Call documentSet.iterator().
   a. Assign the returned value to a variable of the type Iterator..
7. Write a while loop to retrieve each publish template from the collection.

   ```
   while (iterator.hasNext())
   ```
8. Call iterator.next()within the while loop.
   a. Typecast the return value into a Document type and assign it to a variable.
9. Inside the loop, retrieve the name of each publish template and display the name by calling System.out.println(…).
   - Parameter: document.get_Name()

Procedure 2: Call the method inside the main() method

1. Call the getPublishTemplatesEDU(...) method.
   - Parameters:
     - ObjectStore object that you retrieved in the previous activity
2. Comment out the method calls that are not tested in this activity.

Procedure 3: Run the program and verify the results

1. Run the program and verify that a list of publish templates names is displayed in the output window.

Sample output

```
Got the connection
Name of the Domain: P8Domain
Objectstore is: Development
publish template = APIPublishTemplate
publish template = SamplePublishTemplate
```


Solution code for PublishEDU.java

```
package com.ibm.filenet.edu;
import java.util.*;
import com.filenet.api.constants.*;
import com.filenet.api.core.*;
import com.filenet.api.property.*;
import com.filenet.api.publishing.*;
import com.filenet.api.query.*;
import com.filenet.api.collection.*;

public class PublishEDU extends CEConnectionEDU {
   public void publishDocumentEDU(Document document, ObjectStore os, String
   publishTemplatePath){
      PublishTemplate publishTemplate =(PublishTemplate)
      os.fetchObject(ClassNames.PUBLISH_TEMPLATE,publishTemplatePath,
      null);
      String option = "<publishoptions><publicationname>"+
      document.get_Name()+
      "_Publication</publicationname></publishoptions>";
      PublishRequest pr = document.publish(publishTemplate, option);
      PropertyFilter pf = new PropertyFilter();
      pf.addIncludeProperty(0, null, null,
      PropertyNames.PUBLISHING_STATUS,1);
      pr.save(RefreshMode.REFRESH, pf);
      System.out.println(pr.get_PublishingStatus().toString());
      System.out.println("The document is published");
   }
   public void rePublishDocumentEDU(Document document, ObjectStore os,
   String publicationName) throws Exception{
      DocumentSet publications = document.get_DestinationDocuments();
      Iterator it = publications.iterator();
      Document publication = null;
      while (it.hasNext())
      {
         publication = (Document)it.next();
         if (publicationName.equalsIgnoreCase(publication.get_Name()))
         break;
      }
      if(publication != null){
         System.out.println("publication used = " + publication.get_Name());
         String option = "<publishoptions><publicationname>" +
         document.get_Name()+ "_Republished
         </publicationname></publishoptions>";
```

```
         PropertyFilter pf = new PropertyFilter();
         pf.addIncludeProperty(0, null, null,
         PropertyNames.PUBLISHING_STATUS,1);
         pr.save(RefreshMode.REFRESH, pf);
         System.out.println(pr.get_PublishingStatus().toString());
      }
      System.out.println("The document is republished");
   }

public void getPublishTemplatesEDU(ObjectStore os)
{
      String sql = "SELECT pt.Name, pt.Id, pt.DateCreated, pt.StyleTemplate
      FROM PublishTemplate AS pt ";
      sql += " WITH INCLUDESUBCLASSES LEFT OUTER JOIN PublishStyleTemplate
      AS st " ;
      sql += " WITH INCLUDESUBCLASSES ON pt.StyleTemplate = st.This ";
      sql += " WHERE (pt.IsCurrentVersion = true) ORDER BY pt.DateCreated";
      SearchScope search = new SearchScope(os);
      SearchSQL searchSQL = new SearchSQL(sql);
      DocumentSet documents =
      (DocumentSet)search.fetchObjects(searchSQL,Integer.valueOf("50"),
      null, Boolean.valueOf(true));
      com.filenet.api.core.Document doc;
      Iterator it = documents.iterator();
      while (it.hasNext()){
         doc = (com.filenet.api.core.Document)it.next();
         System.out.println("publish template = "+ doc.get_Name());
      }
}

public Document getDocumentEDU(ObjectStore store, String folderPath)
{
      PropertyFilter pf = new PropertyFilter();
      pf.addIncludeProperty(0, null, null, PropertyNames.NAME,1);
      pf.addIncludeProperty(0, null, null,
      PropertyNames.DEPENDENT_DOCUMENTS,1);
      pf.addIncludeProperty(0, null, null,
      PropertyNames.DESTINATION_DOCUMENTS,1);
      Document document = Factory.Document.fetchInstance(store,folderPath,
      pf);
      String documentName = document.get_Name();
      System.out.println(documentName + "  Document is retrieved");
      return document;
}
```



```
public static void main(String[] args)
{
   try {
      PublishEDU myInstance = new PublishEDU();
      Connection conn =
      myInstance.getCEConnectionEDU("p8admin","IBMFileNetP8");
      Domain domain = myInstance.getDomainEDU(conn);
      ObjectStore store = myInstance.getObjectStoreEDU
      (domain,"Development");
      Document document = myInstance.getDocumentEDU(store,
      "/APIFolder/APIPublish.jpg");
      myInstance.getPublishTemplatesEDU(store);
      myInstance.publishDocumentEDU(document,
      store,"/PublishTemplates/APIPublishTemplate");
      myInstance.rePublishDocumentEDU(document, store, document.get_Name() +
      "_Publication");
   }
   catch (Exception e)
   {
      e.printStackTrace();
   }
}
}
```


Unit 16. Compound Documents

Unit overview

This unit contains the following lessons.

Lessons

Skill levels

- Challenge: Minimal guidance
- Walkthrough: More guidance, with step-by-step directions

Requirements

The activities in this unit assume that you have access to the student system configured for these activities.

Unit dependencies

- The “Set Up Eclipse IDE” unit is needed for working with Eclipse.
- The Java class in this unit extends the CEConnectionEDU class from the “Communication with the Content Engine” unit.
- Concepts and some parts of the code from Documents unit is used.

Working with Eclipse IDE

You can write the code using Eclipse, Notepad, or any other text editor. You can run the code using Eclipse or the Command Prompt window.

- The “Set Up Eclipse IDE” unit has the procedures to create a project and other details needed to do the activities.
- This unit provides instructions for both Eclipse and the Command Prompt.

System check

1. Verify that the WebSphere is running:
   a. In your client system browser, go to the following web page: https://ccv01135:9043/ibm/console/logon.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The page displays the Integrated Solution Console.
   c. Log out of the console.
2. Verify that the Content Engine running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/FileNet/Engine

      The page displays contents similar to the following.

Content Engine Startup Context (Ping Page)	
Product Name	P8 Content Engine - 5.0.0
Build Version	dap452.227
Operating System	Linux 2.6.18-164.el5

3. Verify that the Workplace is running:
   a. In your client system browser, go to the following web page: http://ccv01135:9080/Workplace/Browse.jsp
   b. Log in as `p8admin` user with `IBMFileNetP8` as the password.

      The Browse page of the Workplace opens. The page displays a list of Object Stores.
   c. Log out of the Workplace and close the browser.
4. If the services are not running, start the services:
   a. Right-click the desktop on your linux server system (`ccv01135`) and click Open Terminal.
   b. In the terminal, type `sudo su -` to log in as root.
   c. At the password prompt, type `filenet`.
   d. Type `./Startup-Services.sh` to run the shell script that starts the services.
   e. Wait until the terminal displays that all the services are started.
   f. Repeat the steps 1 through 3 to make sure the services are running.

Supporting files

The supporting files for these activities are located in C:\CMJavaAPIProg\Source. (These files include the make.bat file for

compiling code, the run.bat file for running code using the Windows Command Prompt, and the VMArgumentsforEclipse.txt file for Eclipse IDE.)

Solution code

- The solution code for this unit is provided in the C:\CMJavaAPIProg\Solution folder.
- A hard copy of the solution code for this unit is provided at the end of these instructions.
- An Eclipse solution project named CMJavaAPISolution is provided in the C:\CMJavaAPIProg folder.

Developer help

For details on the IBM FileNet P8 Content Engine Java API classes, refer to IBM FileNet P8 Version 5.0 information center > Developing IBM FileNet applications > Content Engine Development > Java API Reference

Important

If you are starting this unit with a fresh student system or have lost your work from the previous unit, do one of the following:

- Use the Eclipse project named JavaSampleProject that is provided in the C:\CMJavaAPIProg folder.
- Create a new project using the instructions in the “Set up Eclipse IDE” unit.
  - Add the `CEConnectionEDU.java` file to the project from the C:\CMJavaAPIProg\Solution folder.

Lesson 16.1. Create compound documents

Overview

Why is this lesson important to you?

Your company wants to hierarchically organize several files containing product information so that they can be assembled together to form a single document. You, as the programmer, are going to write code to create compound documents on the Content Engine.

Activities

- Create compound documents: Challenge, page 16-7
- Create compound documents: Walkthrough, page 16-9

User accounts

Type	User ID	Password
IBM FileNet Workplace	p8admin	IBMFileNetP8
Content Engine Enterprise Manager	p8admin	IBMFileNetP8
Custom application	p8admin	IBMFileNetP8

Create compound documents: Challenge

Challenge

Use the data in the table to create a Java class that extends CEConnectionEDU and write code to do the following:

- Create a parent document (content is optional).
- Retrieve a folder and file the document in the specified folder.
- Set the state of the document to compound document.
- Retrieve children documents and set component relationships with the parent document.

Data

Item	Value

Packages to import	`java.util.*`
	`com.filenet.api.core.*`
	`com.filenet.api.constants.*`
	`com.filenet.api.collection.*`
	`com.filenet.api.property.*`

Object store name	`"Development"`
Document class name	`"Document"`
Document title	`"APIParent"`
Folder path to file the document	`"/APIFolder"`
Folder path for the child document -1	`"/APIFolder/APIChild1.txt"`
Component relationship type for child document -1	`ComponentRelationshipType.STATIC_CR`
Folder path for child document - 2	`"/APIFolder/APIChild2.txt"`
Component relationship type for child document - 2	`ComponentRelationshipType.DYNAMIC_CR`

Verification

Run the program using Eclipse or the Command Prompt and verify the results:

- Sign in to Workplace and locate the Development object store > APIFolder.
- Verify that the compound document has been created with the title that you specified. It has a compound document icon in front of the document name.
- Click the information icon next to the document and select the Child Documents link in the left pane.
- Verify that the two child documents that you added are displayed.

Optional Verification

- Check out and check in the child documents to create more versions.
- Verify that the Link To and Major and Minor Version properties have different values for each child document depending on the ComponentRelationshipType and VersionBindType that you assigned for these child documents.

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIFolder  folder is retrieved
Parent document is created
Compound document state is set
(1) static child document is added - APIChild1.txt
(2) Dynamic child document is added - APIChild2.txt
Compound document is successfully created
```


Create compound documents: Walkthrough

Procedures

Procedure 1: Create a Java class that extends CEConnectionEDU

1. Start Eclipse and open the existing project CMJavaAPI that you created.
2. Expand the project from the Project Explorer pane, right-click the `com.ibm.filenet.edu` package, and click New > Class.
3. In the “New Java Class” window, make sure that the values for Source folder, Package and Modifiers are selected correctly:
   - Source folder: `CMJavaAPI/src`
   - Package: `com.ibm.filenet.edu`
   - Modifiers: `public`
4. Type `CompoundDocumentsEDU` as the class name in the Name field.
5. Type `com.ibm.filenet.edu.CEConnectionEDU` in the Superclass field.
6. Select `public static void main(String[] args)` for “Which method stubs would you like to create?” option.
   a. Clear other options.
7. Click Finish to add the class and close the window.
   a. Verify that the new class is listed under your package in the Project Explorer.
8. Write code inside the class that you just added.

9. Import the following packages before the declaration of the class:

```
java.util.*
com.filenet.api.core.*
com.filenet.api.constants.*
com.filenet.api.collection.*
com.filenet.api.property.*
```

10. Inside the main method, write a `try-catch` block.
    a. Call `exception.printStackTrace()` in the catch block to display the exceptions thrown.
11. Add custom methods and call it from the main method as described in the following steps.

Procedure 2: Write a method to create Document object and file it to a folder

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `createDocumentEDU`
   - Parameters:
     - `ObjectStore` object
     - `Folder` object
   - Returns: `Document`

Note

This method creates a Document object (without content) to be used as a parent document. You can copy and paste the code for creating a document, setting properties, and filing it in a folder from the `DocumentsEDU.java` file that you wrote in "Create Documents objects" lesson of the "Documents" unit. Optionally, you can add content.

2. Call `Factory.Document.createInstance(...)`.
   - Parameters:
     - `ObjectStore` object
     - `"Document"`

   a. Assign the return value to a variable of type `Document`.
3. Call `document.getProperties()`.
   a. Assign the return value to a variable of type `com.filenet.api.property.Properties`.

4. Call properties.putValue(...) to set value to the DocumentTitle property.
   - Parameters:
     - "DocumentTitle"
     - "APIParent"
5. Call document.checkin(...) to check in the document.
   - Parameters:
     - AutoClassify.DO_NOT_AUTO_CLASSIFY
     - CheckinType.MAJOR_VERSION
6. Call document.save(...) to save the changes to the document.
   - Parameter: RefreshMode.REFRESH
7. Call folder.file(...) on the folder object that is passed into this method.
   - Parameters:
     - Document object that you created in step 2
     - AutoUniqueName.AUTO_UNIQUE
     - "APIParent" for the containment name
     - DefineSecurityParentage.DO_NOT_DEFINE_SECURITY_PARENTAGE

   a. Assign the return value to a variable of type ReferentialContainmentRelationship.
8. Call referentialContainmentRelationship .save(...).
   - Parameter: RefreshMode.NO_REFRESH
9. Return the Document object.

Procedure 3: Write a method to create a compound document

1. Define the method using the following signature:
   - Scope: public
   - Name: createCompoundDocumentEDU
   - Parameters:
     - ObjectStore object
     - Document object that is to be the parent document
2. Set the compound document state to the document by calling the document.set_CompoundDocumentState(...)method.
   - Parameter: CompoundDocumentState.COMPOUND_DOCUMENT
3. Call document.save(...) to save the changes to the document.
   - Parameter: RefreshMode.REFRESH

Procedure 4: Retrieve a child document and set component relationship with the parent document

Continue to write the code in the same method from procedure 3.

1. Call `Factory.Document.fetchInstance(...)`to get the document that is used as a child document 1.
   - Parameters:
     - `ObjectStore object`
     - `"/APIFolder/APIChild1.txt"`
     - `null`
   a. Assign the return value to a variable of type `Document`.

Information

Optional steps:

If steps 2 and 3 are not done, the code adds the current version of the document at the time of component relation creation.

2. Retrieve the versions for the child from `document.get_Versions()` on the object from step 1.
   a. Assign the return value to a variable of type `VersionableSet`.
3. Write a `while` loop to retrieve each version and to display the version numbers.
   a. Choose a version of the child to be added to the parent.

4. Call `Factory.ComponentRelationship.createInstance(...)`to create a component relationship object.
   - Parameters:
     - `ObjectStore` object
     - `null`
     - `ComponentRelationshipType.STATIC_CR`
     - `Document` object that is passed into this method(parent)
     - `“ComponentRelation1”`
   a. Assign the return value to a variable of type `ComponentRelationship`.
5. Set value to the child component by calling the `componentRelationship.set_ChildComponent(...)` on the object from step 4.
   - Parameter: `Document` object from step 1
6. Call `componentRelationship.save(...)` to save the changes to the object.
   - Parameter: `RefreshMode.REFRESH`

7. Display the message `A Static child document has been added` by calling `System.out.println(…)`.

Procedure 5: Retrieve a second document and set component relationship with the parent document

Continue the code inside the same method from procedure 3 and use the data in the following table to retrieve the child document and create a component relationship.

1. Repeat procedure 4 with modifications as shown in the following steps to add the second child document to the parent document.

   This second child document will have a different component relationship.

2. For child document 2, in addition to all the steps in procedure 4, set the value to version bind type before saving the component relation object:

   a. Call `componentRelationship.set_VersionBindType(...)`.

      - Parameter: `VersionBindType.LATEST_MAJOR_VERSION`

Item	Value
Folder path for the child document - 2	`"/APIFolder/APIChild2.txt"`
Component relationship type for child document - 2	`ComponentRelationshipType.DYNAMIC_CR`
Name for the ComponentRelationship-2	`ComponentRelation2`
VersionBindType for child document - 2	`VersionBindType.LATEST_MAJOR_VERSION`

Procedure 6: Instantiate the class and call the methods that you wrote

1. Inside the `try` block of the main() method, create an instance of the class and assign the return value to a variable.

   Example:

   ```
   CompoundDocumentsEDU compoundDocInstance=newCompoundDocumentsEDU();
   ```

2. Call the `getCEConnectionEDU(...)` method of the CEConnectionEDU class:

   - Parameters:
     - `"p8admin"`
     - `"IBMFileNetP8"`

   a. Assign the returned `Connection` object to a variable.

3. Call the `getDomainEDU(…)`method of the `CEConnectionEDU` class.

   - Parameter: `Connection` object variable from step 2

   a. Assign the returned `Domain` object to a variable.

4. Call the `getObjectStoreEDU(…)` method of the `CEConnectionEDU` class:
   - Parameters:
     - `Domain` object variable from step 3
     - `"Development"`

   a. Assign the returned `ObjectStore` object to a variable.
5. Call the `getFolderEDU(...)` method of the `CEConnectionEDU` class.
   - Parameters:
     - `Objectstore` variable from step 4
     - `"/API Folder"`

   a. Assign the return value to a variable of type `Folder`.
6. Call the `createDocumentEDU(...)` method.
   - Parameters:
     - `Objectstore` variable from step 4
     - `Folder` variable from step 5

   a. Assign the return value to a variable of type `Document`.
7. Call the `createCompoundDocumentEDU(...)` method.
   - Parameters:
     - `Objectstore` variable from step 4
     - `Document` variable from step 6

Procedure 7: Run the program and verify the results

Refer to the verification steps in Create compound documents: Challenge, page 16-7

Sample output

Refer to the sample output in Create compound documents: Challenge, page 16-7

Lesson 16.2. Retrieve compound documents

Overview

Why is this lesson important to you?

Your company employees access compound documents to review their content. As their programmer, you are going to write code to retrieve a compound document and its child documents when an employee accesses it.

Activities

Retrieve compound documents: Challenge

Challenge

Use the same Java class that you created in the previous lesson and write code to do the following:

- Retrieve a compound document with the properties specified in the data table.
- Retrieve component relationships and display their names and version bind types.
- Retrieve child documents and display their names, and major and minor version numbers.

Data

Item	Value
Object store name	`"Development"`
Folder path for compound document	`"/APIFolder/APIParent"`

Document properties to retrieve	`PropertyNames.NAME`
	`PropertyNames.COMPOUND_DOCUMENT_STATE`
	`PropertyNames.CHILD_RELATIONSHIPS`
	`COMPONENT_RELATIONSHIP_TYPE`
	`PropertyNames.CHILD_DOCUMENTS`

Verification

Run the program using Eclipse or the Command Prompt and verify that the following are displayed in the Command prompt window or in the console of Eclipse:

- A list of names, relationship type, and version bind types of component relationships
- A list of names and major and minor version numbers of child documents

Sample output

```
Got the connection
Name of the domain: P8Domain
Name of the objectstore: Development
APIParent document is retrieved
=======================================
List of component relationships (CR):
CR - 1 Name =  ComponentRelation1
     Relationship type =  STATIC_CR
     Version bind type = null
CR - 2 Name =  ComponentRelation2
     Relationship type =  DYNAMIC_CR
     Version bind type = LATEST_MAJOR_VERSION
=======================================
List of child documents
1. Child document= APIChild1.doc
      Major version= 1
      Minor version= 0
2. Child document= APIChild2.doc
      Major version= 1
      Minor version= 0
```


Retrieve compound documents: Walkthrough

Use the same Java class that you created in the previous lesson.

Procedures

Procedure 1: Write a method to retrieve a compound document

1. Define the method using the following signature:
   - Scope: `public`
   - Name: `getCompoundDocumentEDU`
   - Parameters:
     - `ObjectStore` object
     - `String` for the folder path of the document
2. Create a new instance a `PropertyFilter` object.
   a. Assign the return value to a variable of type `PropertyFilter` object.
3. Call `PropertyFilter.addIncludeProperty(...)`.
   - Parameters:
     - `0`
     - `null`
     - `null`
     - `PropertyNames.NAME`
     - `1`
4. Repeat step 3 to include the following properties to the property filter:
   - `PropertyNames.COMPOUND_DOCUMENT_STATE`
   - `PropertyNames.CHILD_RELATIONSHIPS`
   - `PropertyNames.COMPONENT_RELATIONSHIP_TYPE`
   - `PropertyNames.CHILD_DOCUMENTS`

5. Call `Factory.Document.fetchInstance(...)`.
   - Parameters:
     - `Objectstore` object
     - Folder path that are passed into this method
     - `PropertyFilter` object that you got in steps 2 through 4
6. Retrieve and display the document name by calling `System.print.out(…)`.
   - Parameter: `document.get_Name()`

Procedure 2: Retrieve component relationships from a compound document

Continue the code inside the same method from procedure 1.

1. Verify that the document is a compound document:
   a. Call `document.get_CompoundDocumentState()` on the parent document object from procedure 1.
   b. Assign the return value to a variable of type `CompoundDocumentState`.
   c. Write an `if` statement to see whether the value from step 1b is equal to `CompoundDocumentState.COMPOUND_DOCUMENT_AS_INT`.
   d. Write steps 2 through 4 within the `if` statement.
2. Retrieve the child relationships by calling `document.get_ChildRelationships()`.
   a. Assign the return value to a variable of type `ComponentRelationshipSet`.
3. Call `componentRelationshipSet.iterator()`.
   a. Assign the returned value to a variable of the type `Iterator`.
4. Write a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```

   a. Call `iterator.next()`within the `while` loop.
   b. Typecast the return value into a `ComponentRelationship` type and assign it to a variable.
   c. Retrieve and display the name of each component relationship by calling `System.out.println(…)`.
      - Parameter: `componentRelationship.get_Name()`
   d. Repeat step 4c to retrieve and display the relationship type for each component relationship using the following parameter:

      `componentRelationship.get_ComponentRelationshipType().toString()`

e. Repeat step 4c to retrieve and display the version bind type for each component relationship using the following parameter:

   `componentRelationship.get_VersionBindType()`

Procedure 3: Retrieve child documents from a compound document

Continue the code inside the same method from procedure 1.

1. Retrieve the child documents by calling `document.get_ChildDocuments()`on the parent document object from procedure 1.
   a. Assign the return value to a variable of type `DocumentSet`.
2. Call `documentSet.iterator()`.
   a. Assign the returned value to a variable of the type `Iterator`.
3. Write a `while` loop and iterate through the collection.

   ```
   while (iterator.hasNext())
   ```

   a. Call `iterator.next()`within the while loop.
   b. Typecast the return value into a `Document` type and assign it to a variable.
   c. Call `document.get_Name()` on the object from step b to retrieve the name of child document.
   d. Assign the return value to a variable of `String` type.
   e. Retrieve major and minor version numbers for the child document by calling the following methods, respectively.
      - `document.get_MajorVersionNumber()`
      - `document.get_MinorVersionNumber()`
   f. Assign each of the return values to a variable of `Integer` type.
   g. Display each document name, major and minor version numbers of the documents that are retrieved in steps 3d and 3f by calling `System.out.println(…)`.

Procedure 4: Call the method inside the main() method

1. Call the `getCompoundDocumentEDU(...)` method that you wrote inside the main() method.
   - Parameters:
     - `Objectstore` variable that you got in the previous lesson.
     - `"/APIFolder/APIParent"` (folder path for the document)
2. Comment out the method calls that are not tested in this activity.

Procedure 5: Run the program and verify the results

1. Run the program using the Eclipse or the Command Prompt and verify that the following are displayed in the Command prompt window or in the console of Eclipse:
   - A list of names, relationship type, and version bind types of component relationships
   - A list of names and major and minor version numbers of child documents

Sample output

Refer to the sample output in Retrieve compound documents: Challenge, page 16-17

Solution code for CompoundDocumentsEDU.java

```
   package com.ibm.filenet.edu;
   import java.util.*;
   import com.filenet.api.core.*;
   import com.filenet.api.constants.*;
   import com.filenet.api.collection.*;
   import com.filenet.api.property.*;


public class CompoundDocumentsEDU extends CEConnectionEDU {

      public Document createDocumentEDU(ObjectStore os, Folder folder)
      {
      Document myDoc = Factory.Document.createInstance(os,"Document");
      com.filenet.api.property.Properties properties =
      myDoc.getProperties();
      properties.putValue("DocumentTitle", "APIParent");
      myDoc.checkin(AutoClassify.DO_NOT_AUTO_CLASSIFY,
      CheckinType.MAJOR_VERSION);
      myDoc.save(RefreshMode.REFRESH);
      ReferentialContainmentRelationship rel = folder.file(myDoc,
      AutoUniqueName.AUTO_UNIQUE, "APIParent",
      DefineSecurityParentage.DO_NOT_DEFINE_SECURITY_PARENTAGE);
      rel.save(RefreshMode.NO_REFRESH);
      System.out.println("Parent document is created");
      return myDoc;
   }
```

```
public void createCompoundDocumentEDU(ObjectStore os, Document parentDoc)
{
   // set the compound- state to the document to be a parent
   parentDoc.set_CompoundDocumentState(CompoundDocumentState.COMPOUND_DO
   CUMENT);
   parentDoc.save(RefreshMode.REFRESH);
   System.out.println("Compound document state is set");
   // Get child doc
   Document staticChild=
   Factory.Document.fetchInstance(os,"/APIFolder/APIChild1.doc", null);
   Document dynamicChild =
   Factory.Document.fetchInstance(os,"/APIFolder/APIChild2.doc", null);
   // Setup component relationship
   ComponentRelationship componentRelationship1 =
   Factory.ComponentRelationship.createInstance(os,null,ComponentRelatio
   nshipType.STATIC_CR,parentDoc,"ComponentRelation1");
   componentRelationship1.set_ChildComponent(staticChild);
   componentRelationship1.save(RefreshMode.REFRESH);
   System.out.println("(1) static child document is added - " +
   staticChild.get_Name());

   // Setup component relationship - 2
   ComponentRelationship componentRelationship2 =
   Factory.ComponentRelationship.createInstance(os,null,ComponentRelatio
   nshipType.DYNAMIC_CR,parentDoc,"ComponentRelation2");
   componentRelationship2.set_ChildComponent(dynamicChild);
   //componentRelationship2.set_VersionBindType(VersionBindType.LATEST_V
   ERSION);
   componentRelationship2.set_VersionBindType(VersionBindType.LATEST_MAJ
   OR_VERSION);
   componentRelationship2.save(RefreshMode.REFRESH);
   System.out.println("(2) Dynamic child document is added - " +
   dynamicChild.get_Name());
   System.out.println("Compound document is successfully created");
}


public void getCompoundDocumentEDU(ObjectStore os)
{
   PropertyFilter propFilter = new PropertyFilter();
   propFilter.addIncludeProperty(0, null, null, PropertyNames.NAME,1);
   propFilter.addIncludeProperty(0, null, null,
   PropertyNames.COMPOUND_DOCUMENT_STATE,1);
```



```
propFilter.addIncludeProperty(0, null, null,
PropertyNames.COMPONENT_RELATIONSHIP_TYPE,1);
propFilter.addIncludeProperty(0, null, null,
PropertyNames.CHILD_RELATIONSHIPS,1);
propFilter.addIncludeProperty(0, null, null,
PropertyNames.CHILD_DOCUMENTS,1);
Document doc = Factory.Document.fetchInstance
(os,"/APIFolder/APIParent",propFilter);
CompoundDocumentState compDocState = doc.get_CompoundDocumentState();
System.out.println(doc.get_Name() + " document is retrieved");
int compDocStatevalue = compDocState.getValue();

if (compDocStatevalue ==
CompoundDocumentState.COMPOUND_DOCUMENT_AS_INT)
{
ComponentRelationshipSet comRelationshipSet =
doc.get_ChildRelationships();
Iterator it = comRelationshipSet.iterator();
ComponentRelationship comRelation;
int counts = 1;
System.out.println("========================================");
System.out.println("List of component relationships (CR): ");
while (it.hasNext())
{
   comRelation = (ComponentRelationship)it.next();
   String comName = comRelation.get_Name();
   String comRelType =
   comRelation.get_ComponentRelationshipType().toString();
   VersionBindType vbType = comRelation.get_VersionBindType();
   System.out.println("CR - " + counts + " Name =  " + comName);
   System.out.println("     Relationship type =  " + comRelType);
   System.out.println("     Version bind type = " + vbType);
   counts = counts + 1;
}
DocumentSet childDocs = doc.get_ChildDocuments();
Iterator childIt = childDocs.iterator();
Document childDoc;
System.out.println("========================================");
System.out.println("List of child documents");
int count = 1;
while (childIt.hasNext())
{
   childDoc= (Document)childIt.next();
   String cDocName = childDoc.get_Name();
   Integer majorVerNum = childDoc.get_MajorVersionNumber();
```

```
         Integer minorVerNum = childDoc.get_MinorVersionNumber();
         System.out.println(count + ". Child document= " + cDocName);
         System.out.println("      Major version= " + majorVerNum);
         System.out.println("      Minor version= " + minorVerNum);
         count = count + 1;
      }
      }//if
   }//get method

   public static void main(String[] args) {
      try {
         CompoundDocumentsEDU compoundDocInstance = new
         CompoundDocumentsEDU();
         Connection conn =
         compoundDocInstance.getCEConnectionEDU("p8admin","IBMFileNetP8");
         Domain domain = compoundDocInstance.getDomainEDU(conn);
         ObjectStore store =
         compoundDocInstance.getObjectStoreEDU(domain,"Development");
         Folder folder = compoundDocInstance.getFolderEDU(store,
         "/APIFolder");
         Document document =
         compoundDocInstance.createDocumentEDU(store,folder);
         compoundDocInstance.createCompoundDocumentEDU(store, document);
         compoundDocInstance.getCompoundDocumentEDU(store);
      }
      catch (Exception e) {
         e.printStackTrace();
      }
   }//main
}//class
```