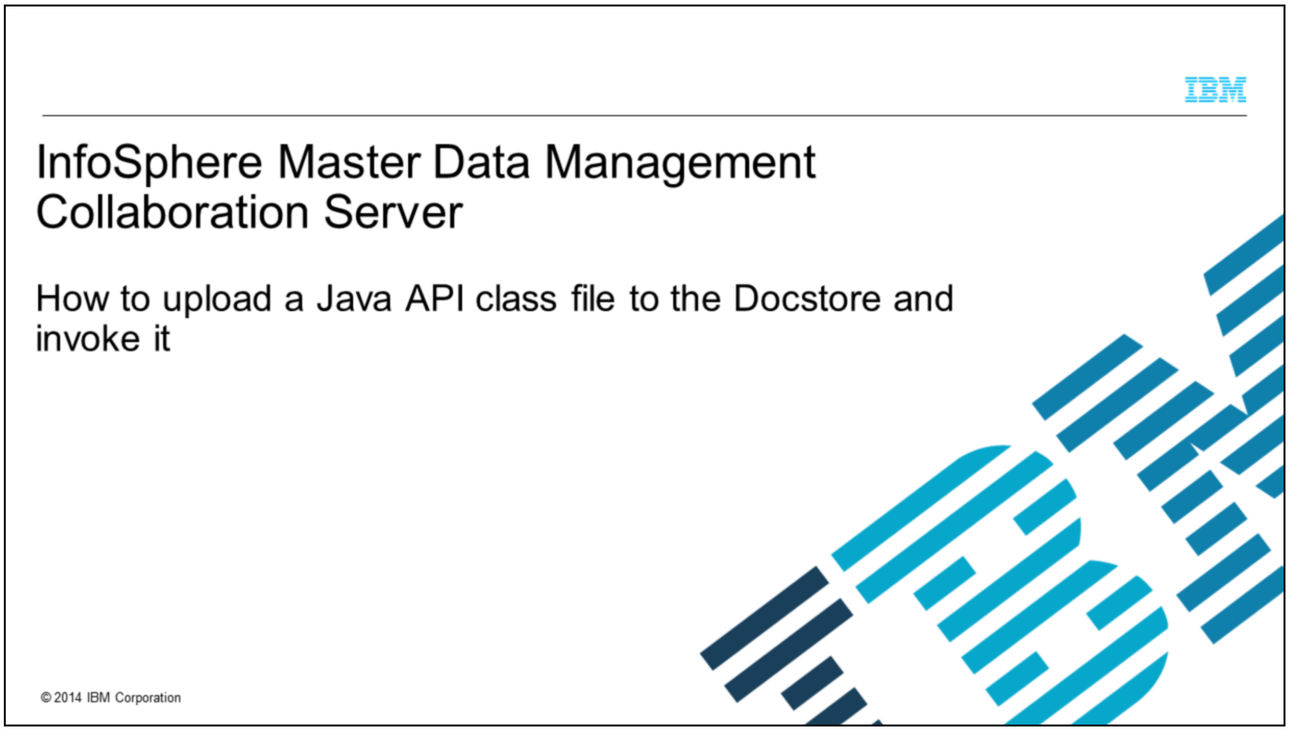
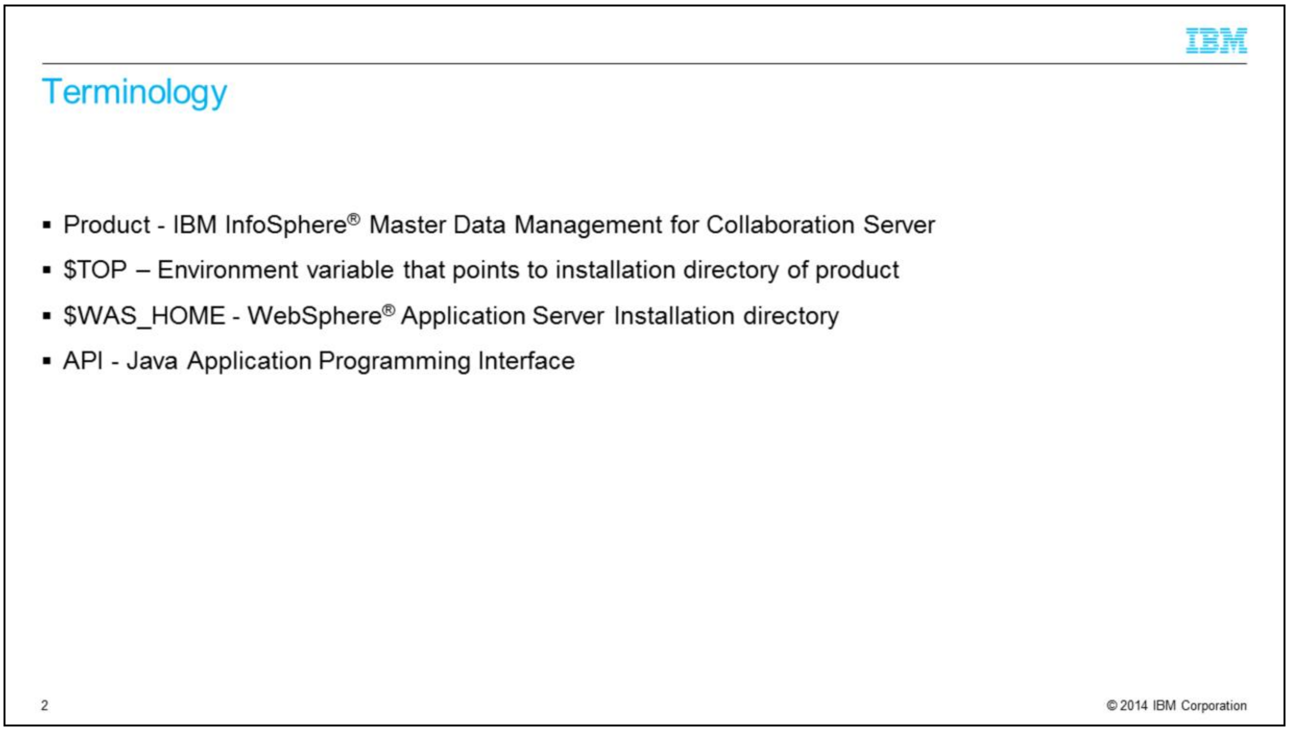


InfoSphere Master Data Management Collaboration Server

How to upload a Java API class file to the Docstore and invoke it

This presentation covers the topic of how to upload directly to the docstore and use a class file that has a Java™ API implementation. In this method, you do not need Integrated development environments such as Rational® Software Architect to develop your application. Also note that this method is to be used for testing purposes only as in production environments there are typically multiple extension point implementations that need to be deployed and that is done best through a single JAR file with all the bundled classes.



Terminology

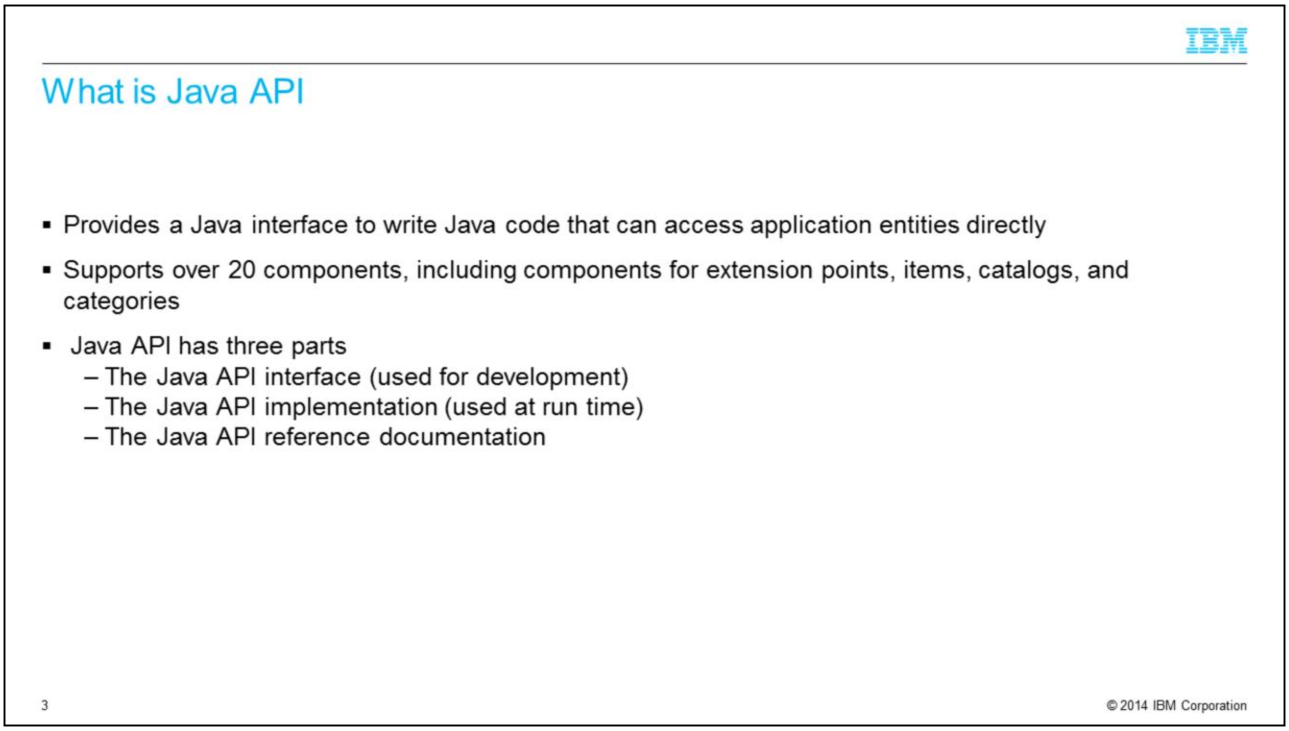
- Product - IBM InfoSphere® Master Data Management for Collaboration Server
- $TOP – Environment variable that points to installation directory of product
- $WAS_HOME - WebSphere® Application Server Installation directory
- API - Java Application Programming Interface

2

Before going into details, there is some terminology that you need to be aware of. The official name of the product or application that is referenced in this presentation is IBM InfoSphere Master Data Management for Collaboration Server and is referred to as InfoSphere MDMCS.

The term $TOP is an environment variable that points to the installation directory of the product.

WAS_HOME is the directory where the WebSphere Application Server is installed and API refers to Java Application Programming Interface.



What is Java API

- Provides a Java interface to write Java code that can access application entities directly
- Supports over 20 components, including components for extension points, items, catalogs, and categories
- Java API has three parts
  - The Java API interface (used for development)
  - The Java API implementation (used at run time)
  - The Java API reference documentation

3 © 2014 IBM Corporation

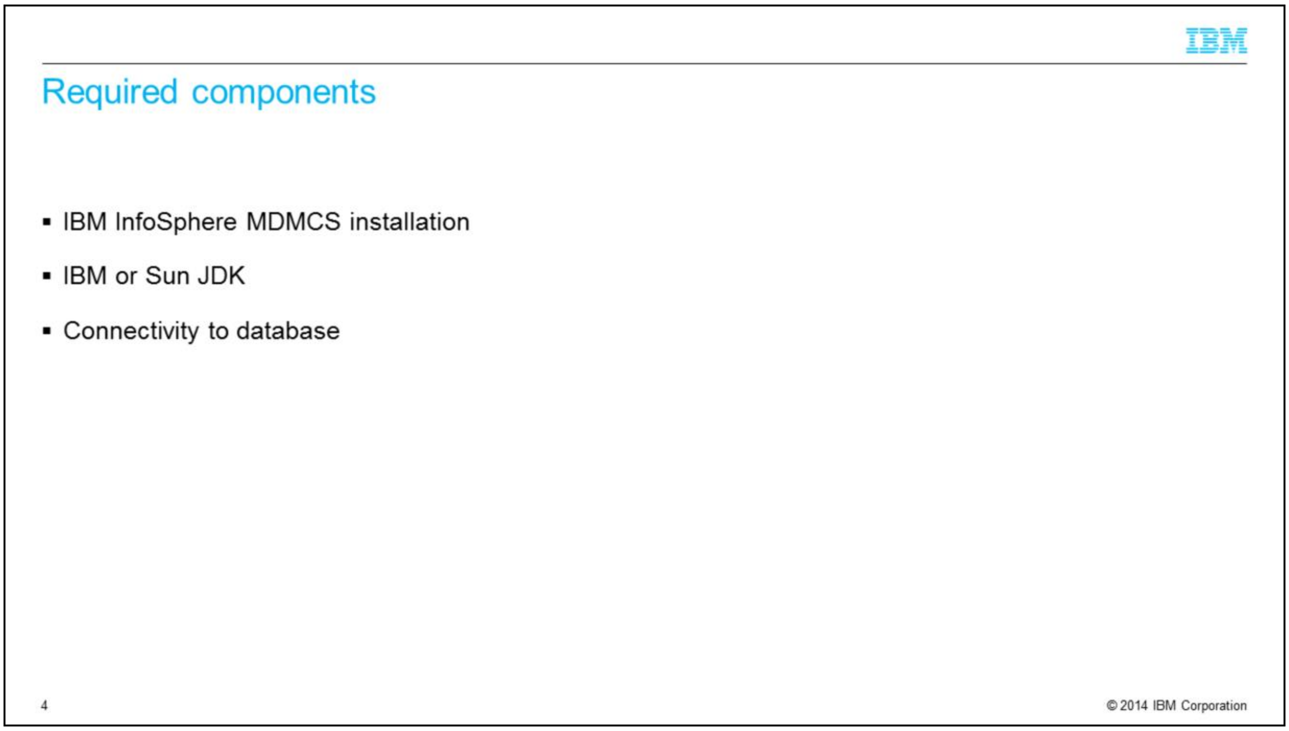
The Java API provides a Java interface that exposes a set of classes and methods. You can use these to write Java code that can access application entities directly, without the need for custom scripts. There are over 20 components for various entities such as extension points, items, catalogs, and categories that are supported.

The Java API has three parts. The Java API interface, which is used for development, the Java API implementation, which is used at run time, and the Java API reference documentation.

The Java API interface is a set of Java interfaces that document all the classes and methods that are available to you. This API is shipped as a .jar file called ccd_javaapi2.jar in the javaapi folder of the InfoSphere MDMCS installation directory.

The Java API implementation refers to the application internal code that provides the functionality that is documented in the Java API interface. A Java class that uses the Java API interface automatically uses this internal code when deployed.

The Java API reference documentation provides explanations for some common classes and methods that are available in the Java API interface. For a complete technical reference with detailed explanations of the classes and methods that are available in your InfoSphere MDMCS instance, see the InfoSphere MDMCS Javadoc. This document is generated from the InfoSphere MDMCS code and is shipped as a compressed file called ccd_javaapi2doc.zip in the javaapi folder of the InfoSphere MDMCS installation directory. To read the documentation, extract this file to a directory and open the index.html file from that directory in a web browser.


Required components

- IBM InfoSphere MDMCS installation
- IBM or Sun JDK
- Connectivity to database

4

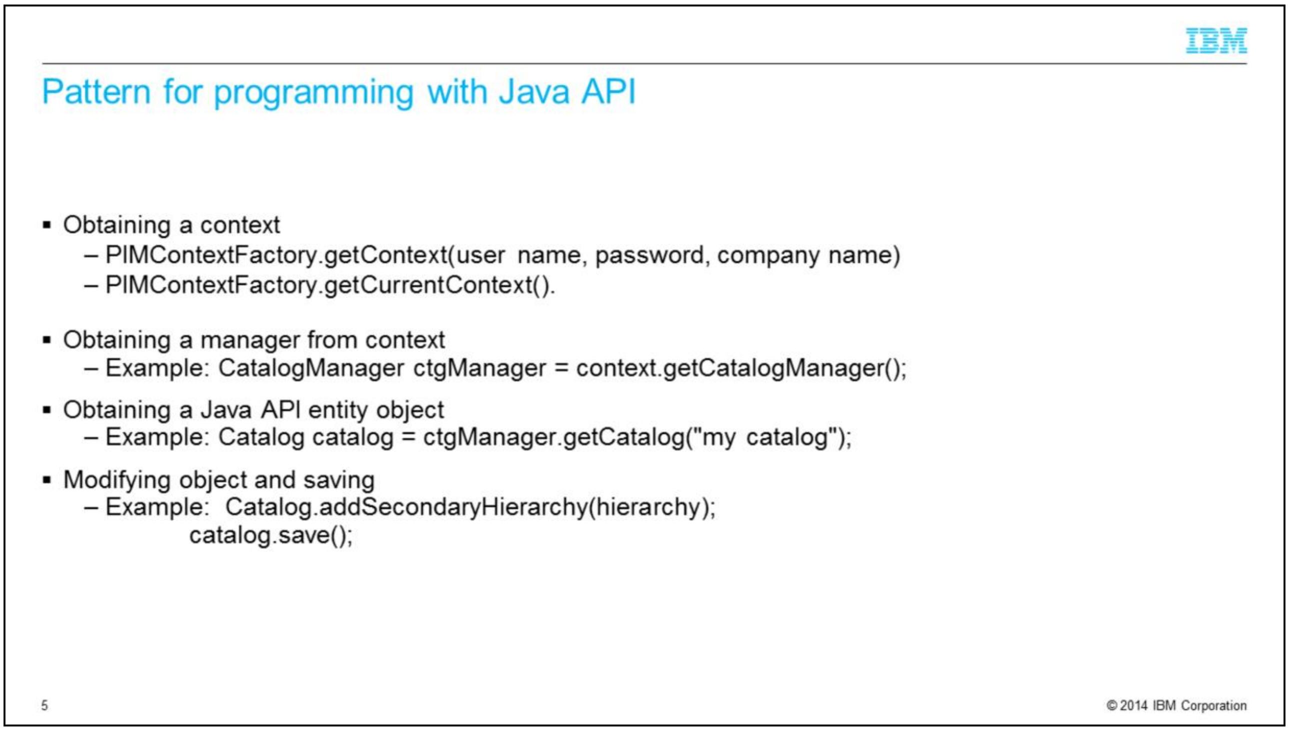
© 2014 IBM Corporation

There are components that are required for writing Java API-based code. You need access to a running IBM InfoSphere MDMCS installation.

You need IBM or Sun JDK and the version of the JDK depends on the version of the InfoSphere MDMCS being used. The JDK version to be used starts from 1.5.

The tools.jar file from JDK is required to be present in the class path of the product.

Client connectivity to the database, which is being used by the IBM InfoSphere MDMCS instance and database-specific client libraries. For Oracle, ojdbc5.jar or ojdbc6.jar is required on the class path of the project. For DB2®, db2jcc.jar and db2jcc_license_cu.jar are required on the class path.



Pattern for programming with Java API

- Obtaining a context
  - PIMContextFactory.getContext(user name, password, company name)
  - PIMContextFactory.getCurrentContext().
- Obtaining a manager from context
  - Example: CatalogManager ctgManager = context.getCatalogManager();
- Obtaining a Java API entity object
  - Example: Catalog catalog = ctgManager.getCatalog("my catalog");
- Modifying object and saving
  - Example: Catalog.addSecondaryHierarchy(hierarchy);
    catalog.save();

5

© 2014 IBM Corporation

When programming through the Java API, the pattern that is typically followed is obtaining a context, obtaining a manager from the context, obtaining a Java API entity object, and modifying the object and saving it.

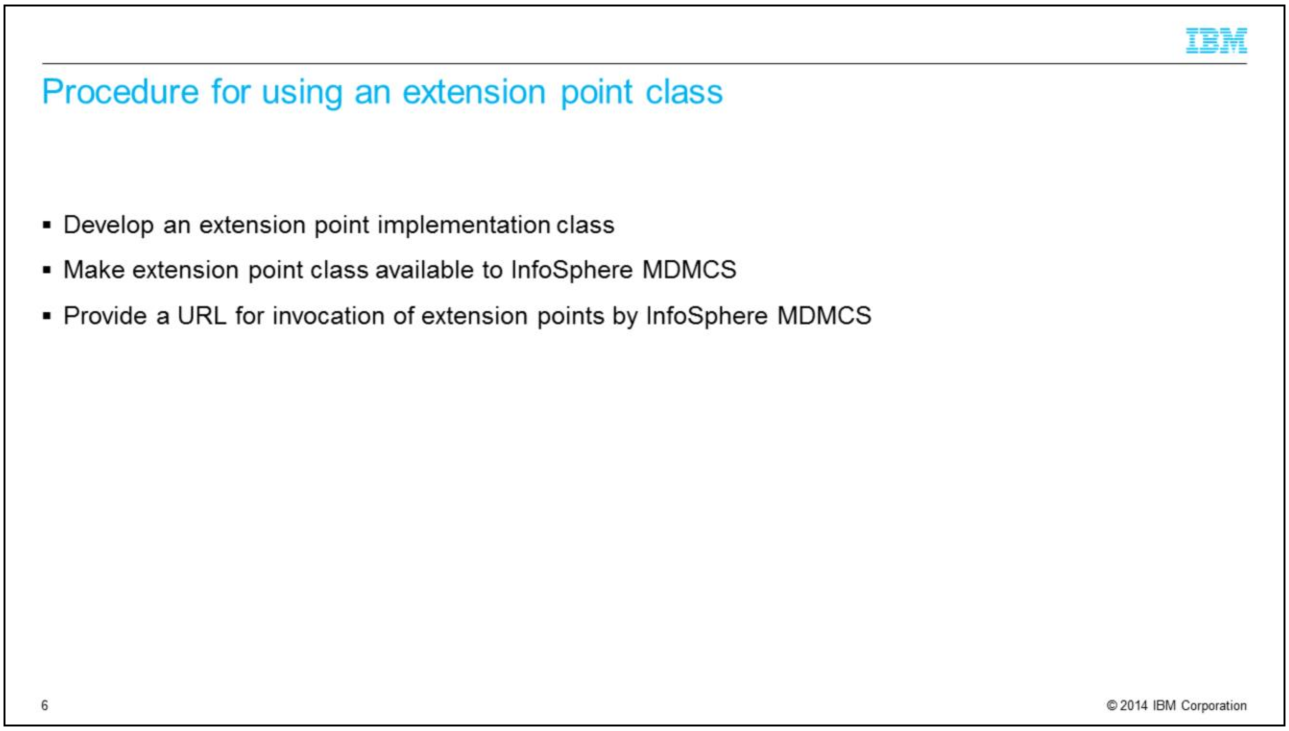
Regarding obtaining a context to access IBM InfoSphere MDMCS entities and methods, the Java code must first obtain a PIMContext. You can obtain a PIMContext in two ways. You can obtain a fresh context by providing the user name, password, and company information to the API. For example, PIMContextFactory.getContext. This method is used when you are writing stand-alone Java API applications or unsecured web services.

If the Java API code is running in an already authenticated context, for example, an extension point implementation class that runs within the InfoSphere MDMCS application or a secure web service, where authentication information was provided when the web service was started, you can obtain the existing context through the API, PIMContextFactory.getCurrentContext(). Using this approach removes the over head of creating extra contexts.

Regarding obtaining a manager from the context, after the context is available, you can retrieve from the context a Java API manager object that corresponds to the entity. For example, a manager for the catalog can be obtained by using the Java API. CatalogManager ctgManager = context.getCatalogManager();

Regarding obtaining a Java API entity object, after manager for an entity is obtained from the context, you can access the entity itself. For example, a catalog object can be obtained from the catalog manager by using the Java API. ctgManager.getCatalog with the catalog name as the input parameter.

Regarding modifying the object and saving it, the entity object that you obtained from the manager class can be modified by using the APIs that are available within the object itself. After the modifications are done, save them to the database with the save() method. For example, a catalog object that is obtained from the catalog manager can be modified and saved by using the Java API. addSecondaryHierarchy with hierarchy as a parameter.



Procedure for using an extension point class

- Develop an extension point implementation class
- Make extension point class available to InfoSphere MDMCS
- Provide a URL for invocation of extension points by InfoSphere MDMCS

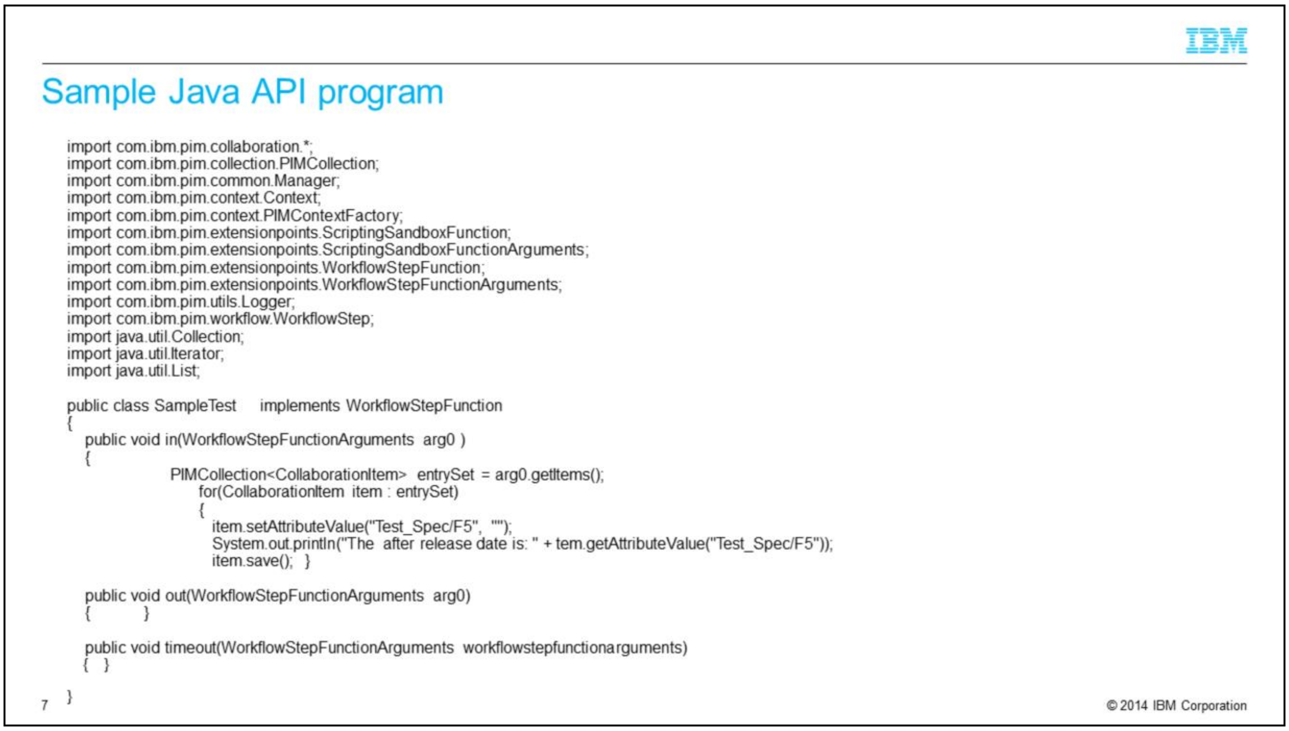
6

© 2014 IBM Corporation

Extension points are the various points within InfoSphere MDMCS where you can modify the behavior by running some user-defined business logic.

In order to implement the extension points using the Java API, you need to develop and extend a set of predefined interfaces that are supplied with InfoSphere MDMCS.

There are steps to be used when using Java API-based extension points. First, develop an extension point class to provide custom code for a particular method. Next, to make the extension point class available to InfoSphere MDMCS, load the .class file to the document store or make it available using a JAR file in the classpath of the InfoSphere MDMCS instance. Finally, before InfoSphere MDMCS can invoke a specific extension point, the Java-based extension point implementation class needs to be registered from the corresponding extension point of the InfoSphere MDMCS user interface. You provide a URL for invocation of extension points.

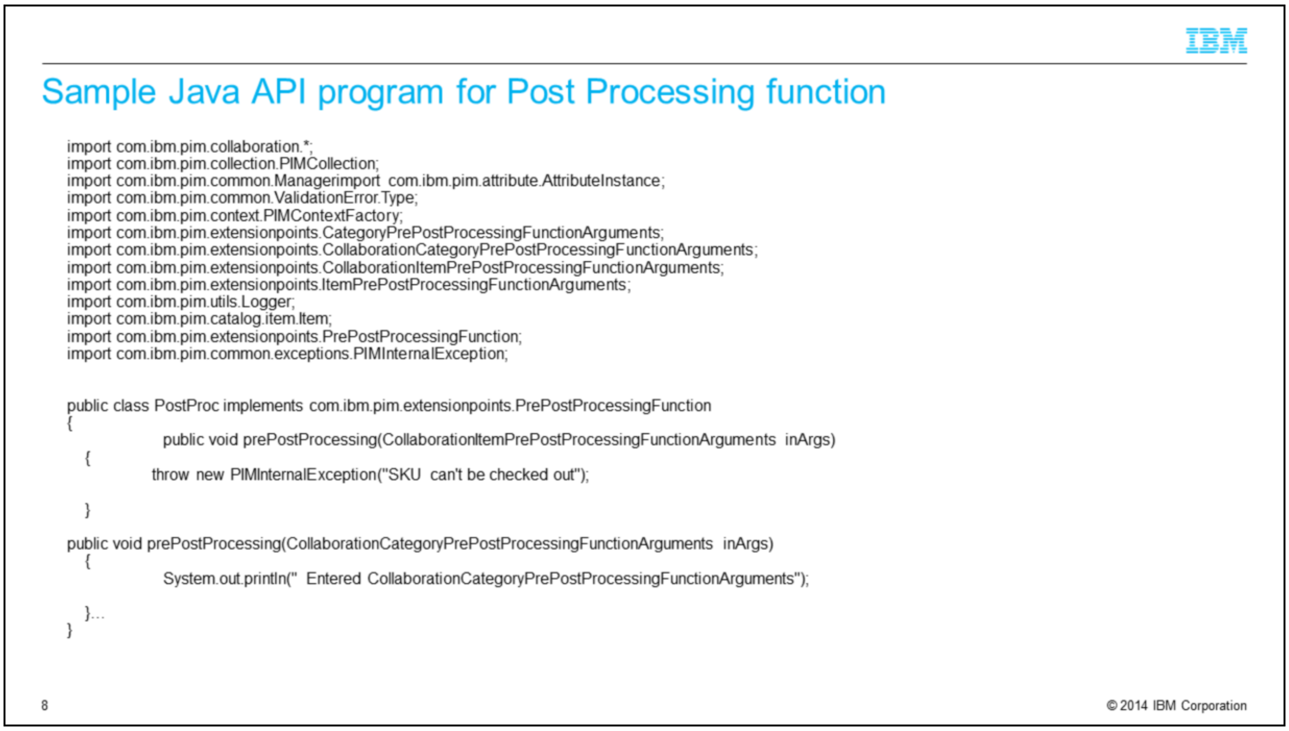


Sample Java API program

```
import com.ibm.pim.collaboration.*;
import com.ibm.pim.collection.PIMCollection;
import com.ibm.pim.common.Manager;
import com.ibm.pim.context.Context;
import com.ibm.pim.context.PIMContextFactory;
import com.ibm.pim.extensionpoints.ScriptingSandboxFunction;
import com.ibm.pim.extensionpoints.ScriptingSandboxFunctionArguments;
import com.ibm.pim.extensionpoints.WorkflowStepFunction;
import com.ibm.pim.extensionpoints.WorkflowStepFunctionArguments;
import com.ibm.pim.utils.Logger;
import com.ibm.pim.workflow.WorkflowStep;
import java.util.Collection;
import java.util.Iterator;
import java.util.List;

public class SampleTest     implements WorkflowStepFunction
{
   public void in(WorkflowStepFunctionArguments  arg0 )
   {
               PIMCollection<CollaborationItem>  entrySet = arg0.getItems();
                   for(CollaborationItem  item : entrySet)
                   {
                     item.setAttributeValue("Test_Spec/F5",  "");
                     System.out.println("The  after release date is: " + tem.getAttributeValue("Test_Spec/F5"));
                     item.save();  }

   public void out(WorkflowStepFunctionArguments  arg0)
   {         }

   public void timeout(WorkflowStepFunctionArguments  workflowstepfunctionarguments)
   {   }

}
```

7

This slide displays a simple example of a Java API program. This extension class is a workflow extension class and has custom logic in the IN function of the workflow wherein it is setting the value of one of the attributes to a blank.

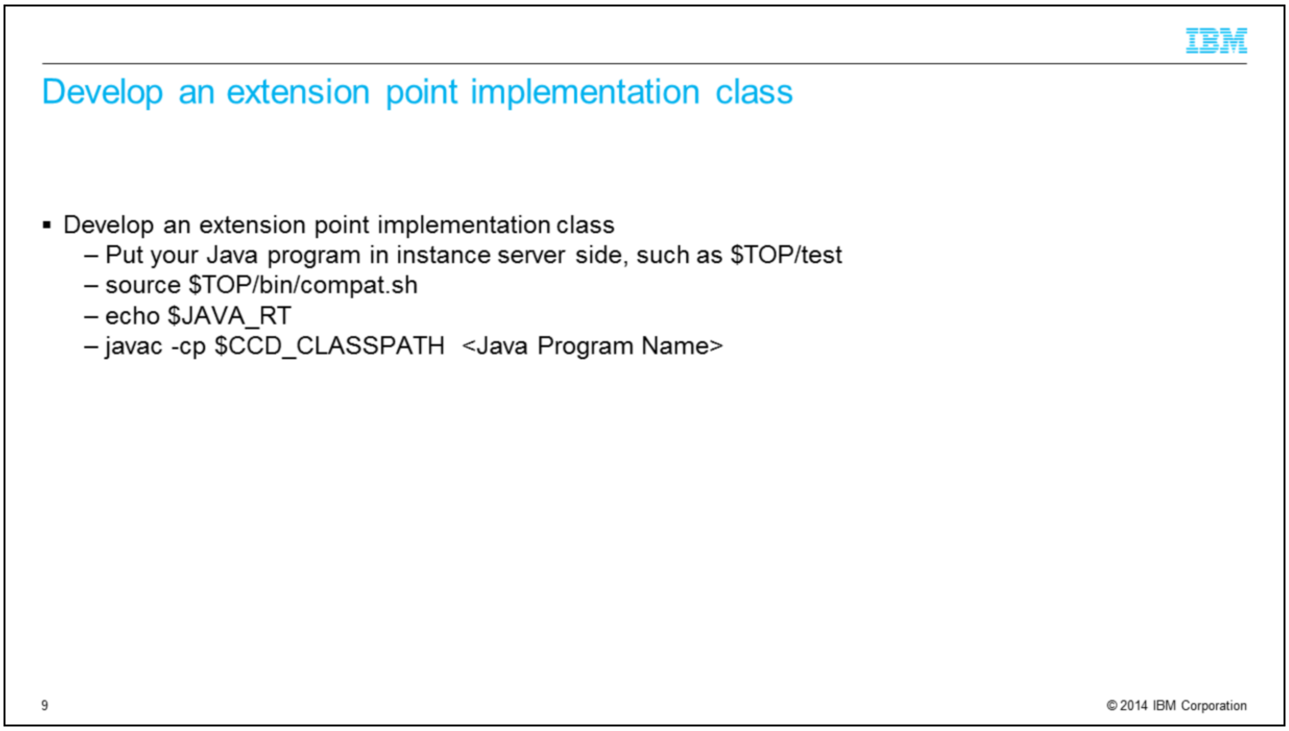


Sample Java API program for Post Processing function

```
import com.ibm.pim.collaboration.*;
import com.ibm.pim.collection.PIMCollection;
import com.ibm.pim.common.Managerimport  com.ibm.pim.attribute.AttributeInstance;
import com.ibm.pim.common.ValidationError.Type;
import com.ibm.pim.context.PIMContextFactory;
import com.ibm.pim.extensionpoints.CategoryPrePostProcessingFunctionArguments;
import com.ibm.pim.extensionpoints.CollaborationCategoryPrePostProcessingFunctionArguments;
import com.ibm.pim.extensionpoints.CollaborationItemPrePostProcessingFunctionArguments;
import com.ibm.pim.extensionpoints.ItemPrePostProcessingFunctionArguments;
import com.ibm.pim.utils.Logger;
import com.ibm.pim.catalog.item.Item;
import com.ibm.pim.extensionpoints.PrePostProcessingFunction;
import com.ibm.pim.common.exceptions.PIMInternalException;


public class PostProc implements com.ibm.pim.extensionpoints.PrePostProcessingFunction
{
                public void prePostProcessing(CollaborationItemPrePostProcessingFunctionArguments  inArgs)
   {
              throw  new  PIMInternalException("SKU  can't be checked out");

   }

public void prePostProcessing(CollaborationCategoryPrePostProcessingFunctionArguments  inArgs)
   {
                System.out.println("  Entered  CollaborationCategoryPrePostProcessingFunctionArguments");

   }...
}
```

8 © 2014 IBM Corporation

This is another simple example of a Java API program for a post processing function implementation. Here, a PIMInternalException is thrown when trying to check out an item from the catalog to a workflow.



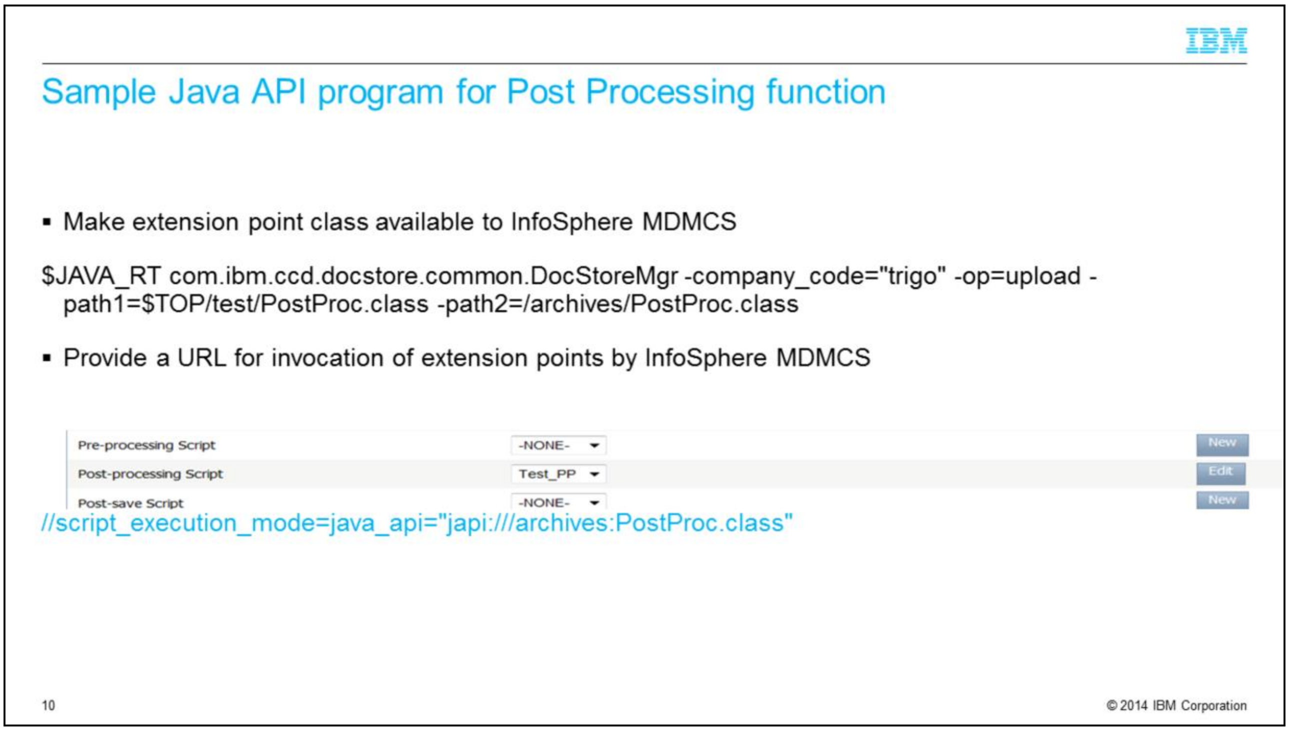
Develop an extension point implementation class

- Develop an extension point implementation class
  - Put your Java program in instance server side, such as $TOP/test
  - source $TOP/bin/compat.sh
  - echo $JAVA_RT
  - javac -cp $CCD_CLASSPATH <Java Program Name>

9

© 2014 IBM Corporation

There are more details on the first step, which is to write custom code for your extension point class. Write your Java program into a sample directory such as “test” within the product’s installation directory. For example, $TOP/test. It is a good idea to source the compat.sh so that all variables including some of the old environment variables such as $TOP, $CCD_DB, and $JAVA_RT are also added. Next, compile the program. This generates the class file.



Sample Java API program for Post Processing function

- Make extension point class available to InfoSphere MDMCS

```
$JAVA_RT com.ibm.ccd.docstore.common.DocStoreMgr -company_code="trigo" -op=upload -
  path1=$TOP/test/PostProc.class -path2=/archives/PostProc.class
```

- Provide a URL for invocation of extension points by InfoSphere MDMCS

Pre-processing Script	-NONE-	New
Post-processing Script	Test_PP	Edit
Post-save Script	-NONE-	New

```
//script_execution_mode=java_api="japi:///archives:PostProc.class"
```

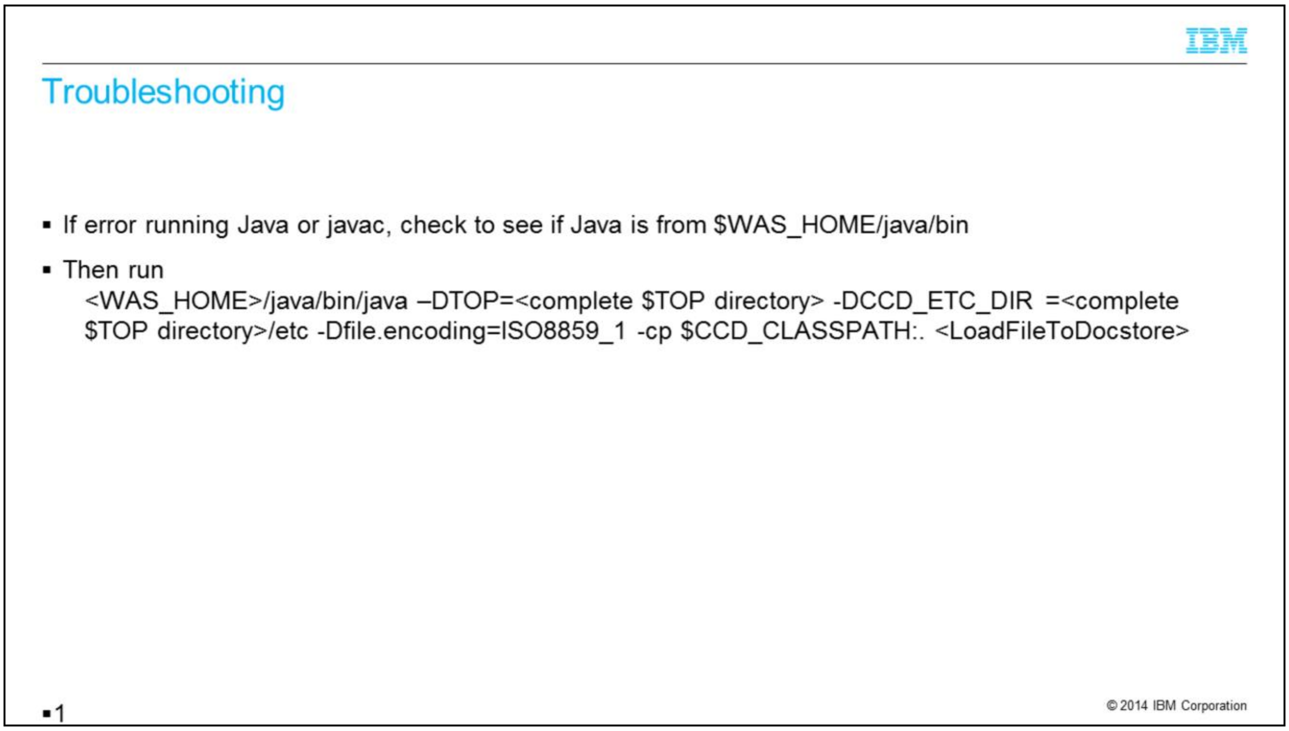
10 © 2014 IBM Corporation

Next, upload the generated class file from $TOP/test, that is path1 to the docstore path, that is path2. In this example, the class file is being uploaded to the company Trigo.

The last step is to create a Post Processing script on the catalog from the UI. In the screen capture that is displayed on this slide, you see the script being created with name as Test_PP. Put in this line to invoke the Java API.

This program ensures that whenever an item is checked out to a workflow from a catalog, the Java API implementation class within the Post Processing script creates an exception to the UI.

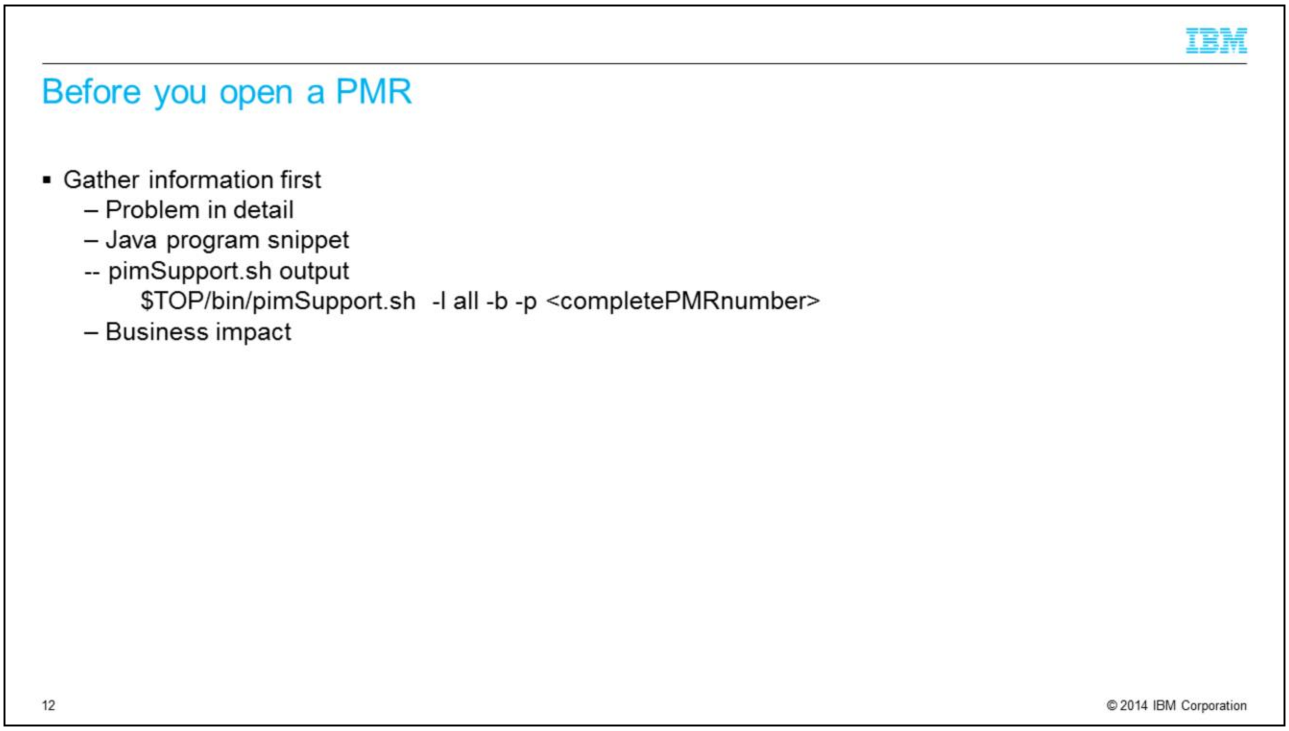
This is just a simple implementation. You can customize it further to work with your requirements.


Troubleshooting

- If error running Java or javac, check to see if Java is from $WAS_HOME/java/bin
- Then run
  <WAS_HOME>/java/bin/java –DTOP=<complete $TOP directory> -DCCD_ETC_DIR =<complete $TOP directory>/etc -Dfile.encoding=ISO8859_1 -cp $CCD_CLASSPATH:. <LoadFileToDocstore>

▪1

Sometimes Java needs to be loaded from the WebSphere Application Server installation directory. You can then compile the program with the command as displayed on this slide.



Before you open a PMR

- Gather information first
  - Problem in detail
  - Java program snippet
  - pimSupport.sh output

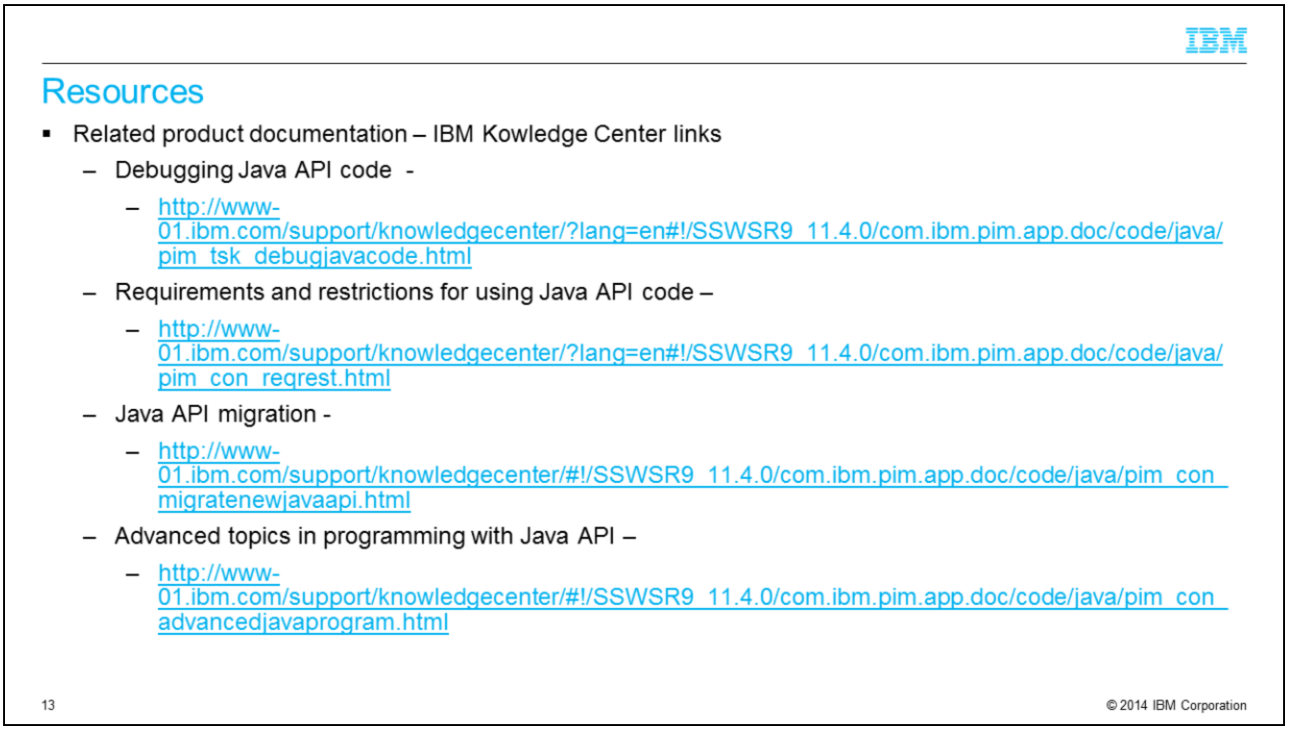
    ```
    $TOP/bin/pimSupport.sh  -l all -b -p <completePMRnumber>
    ```
  - Business impact

12

© 2014 IBM Corporation

Before opening a PMR and contacting product Support, have a detailed problem description including the use case, expected output, and the result that is seen now. Also, useful for further diagnosis is to upload the relevant code so that Support can also use the same code to reproduce the issue generically. Attach to the PMR the pimSupport output that includes the logging for when the issue was reproduced. The pimSupport script gives the product logs and configuration files. The command to run is displayed on this slide.

Finally, provide Support the business impact of this issue on your application or project.



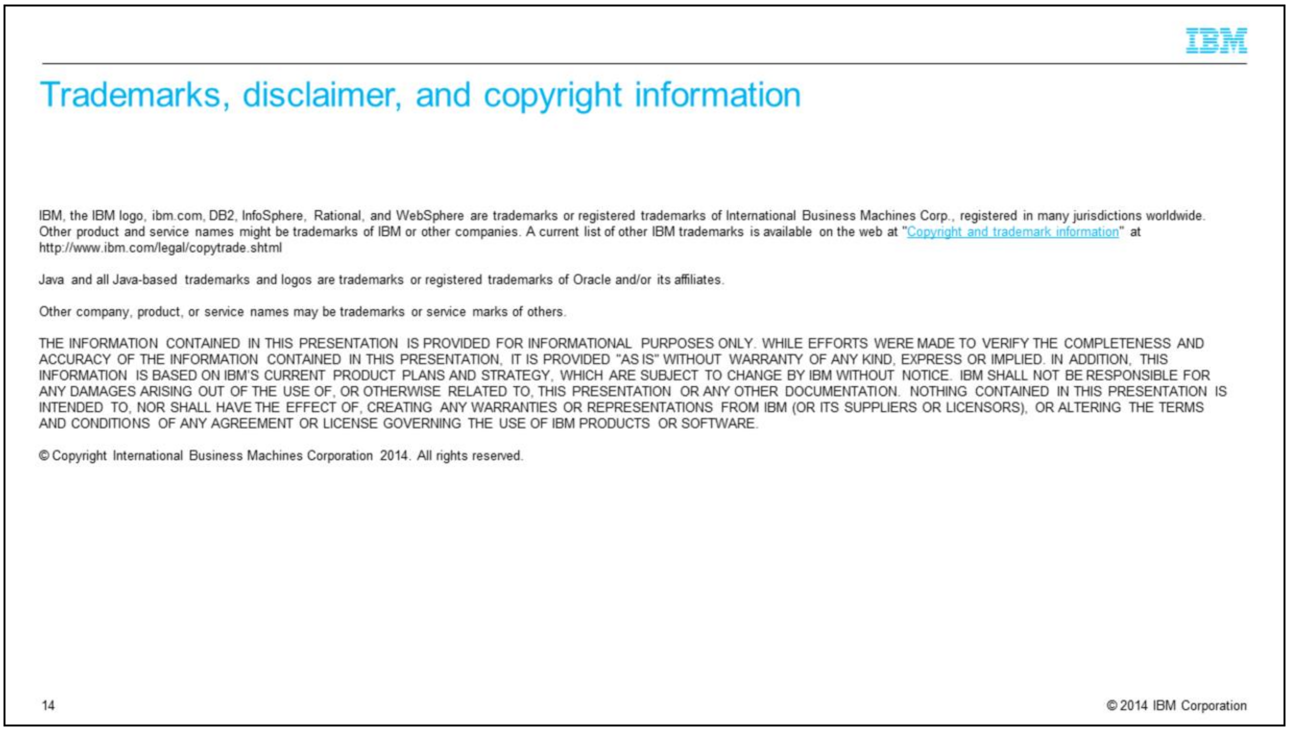
Resources

- Related product documentation – IBM Kowledge Center links
  - Debugging Java API code -
    - http://www-01.ibm.com/support/knowledgecenter/?lang=en#!/SSWSR9_11.4.0/com.ibm.pim.app.doc/code/java/pim_tsk_debugjavacode.html
  - Requirements and restrictions for using Java API code –
    - http://www-01.ibm.com/support/knowledgecenter/?lang=en#!/SSWSR9_11.4.0/com.ibm.pim.app.doc/code/java/pim_con_reqrest.html
  - Java API migration -
    - http://www-01.ibm.com/support/knowledgecenter/#!/SSWSR9_11.4.0/com.ibm.pim.app.doc/code/java/pim_con_migratenewjavaapi.html
  - Advanced topics in programming with Java API –
    - http://www-01.ibm.com/support/knowledgecenter/#!/SSWSR9_11.4.0/com.ibm.pim.app.doc/code/java/pim_con_advancedjavaprogram.html

13

© 2014 IBM Corporation

For reference, this slide displays links that you might find useful.



Trademarks, disclaimer, and copyright information