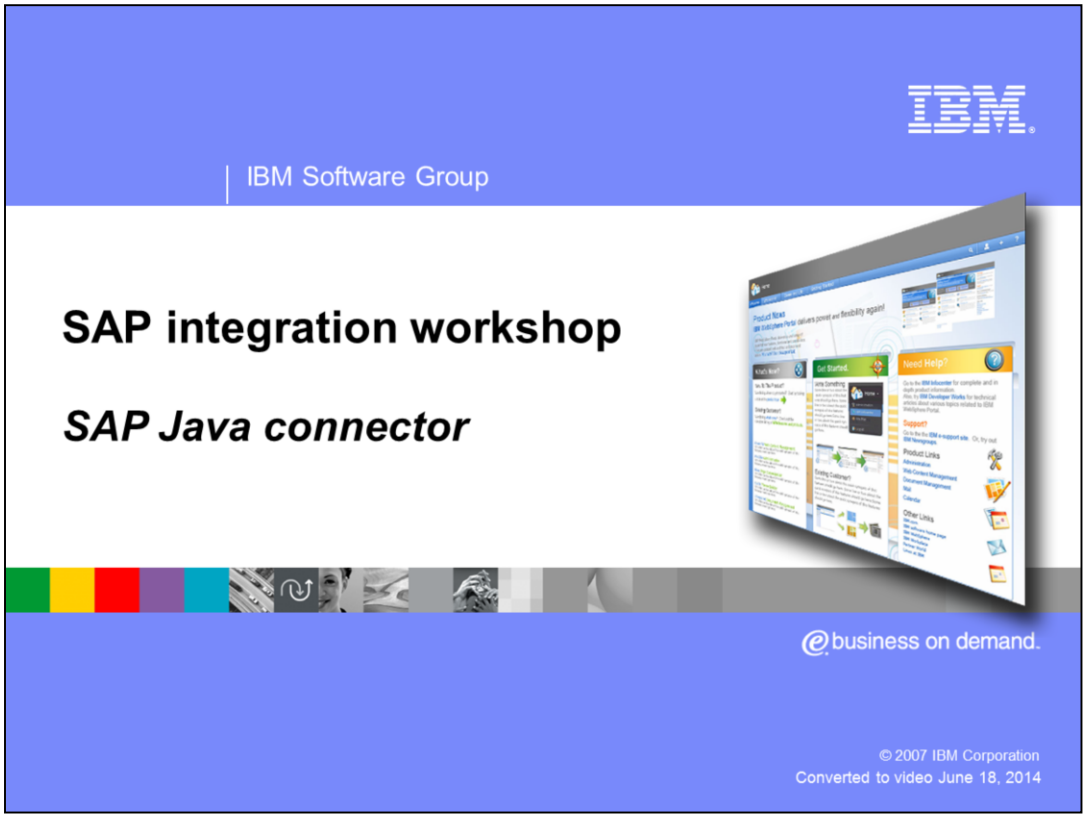
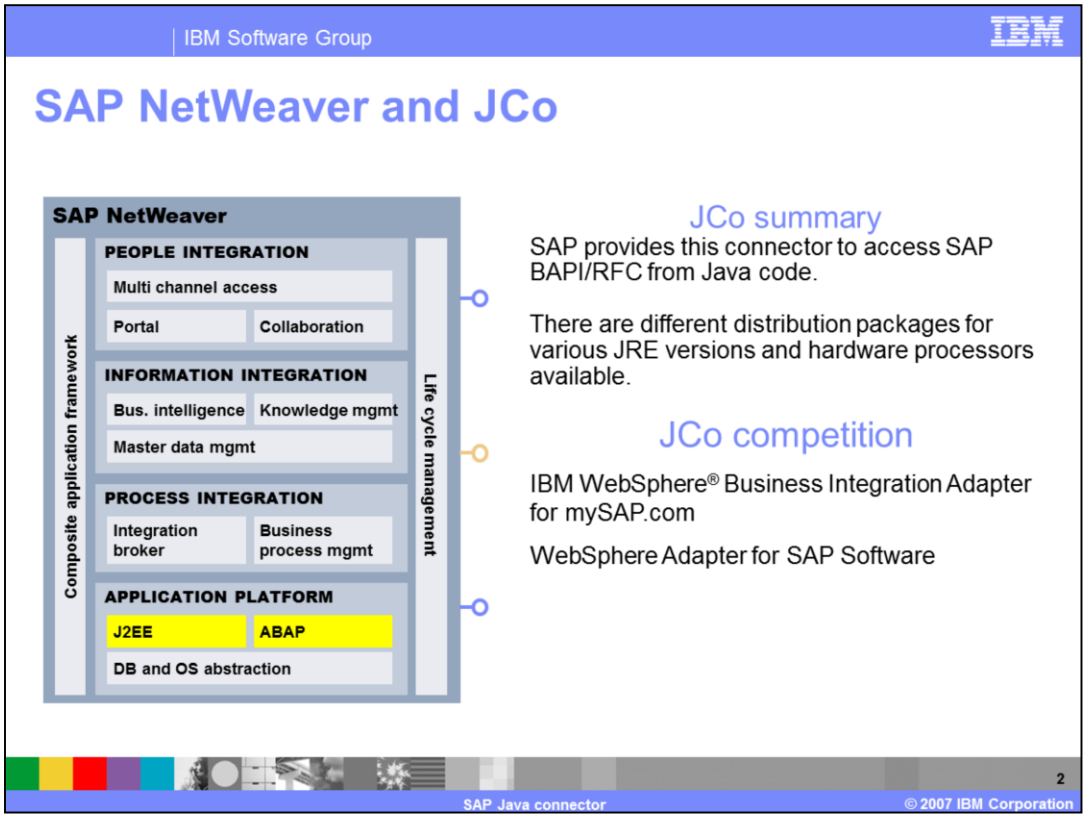


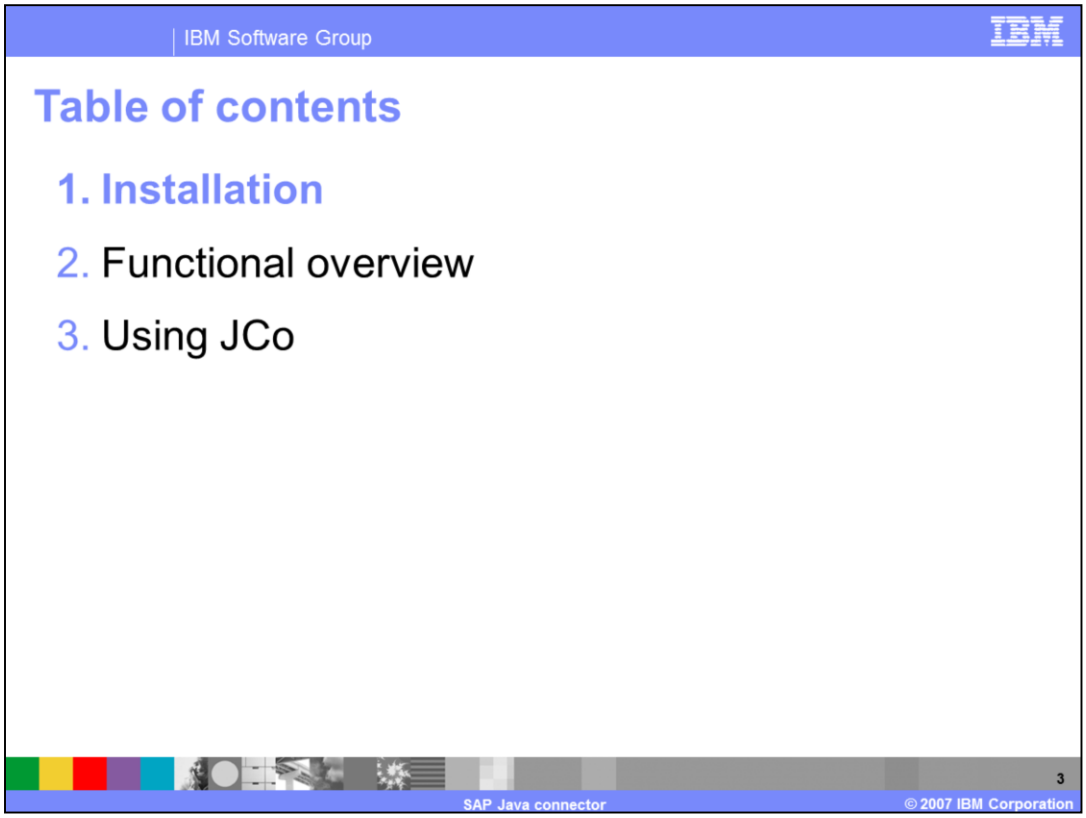
This presentation will describe the SAP Java™ connector, provide information on how to install it, give a functional overview of the connector, and show how to use it in a Java client.



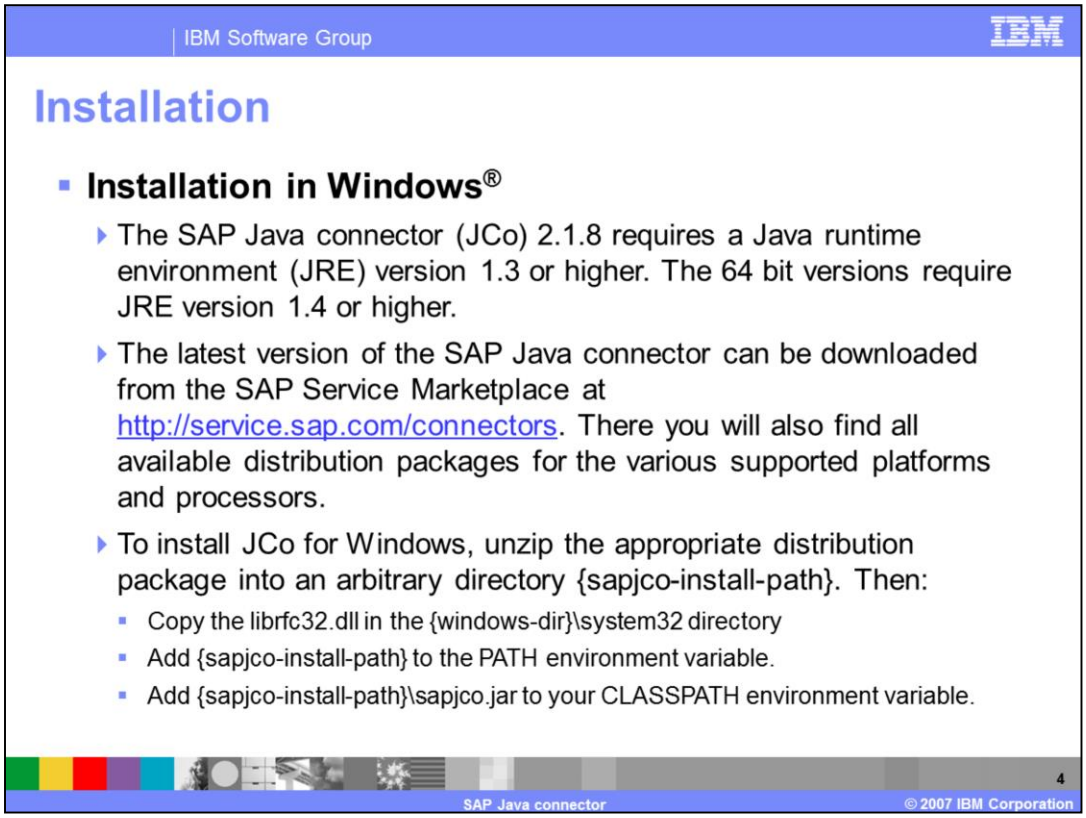
The graphic on this slide shows the SAP NetWeaver components. Notice that the application platform consists of both Java and ABAP components. The need for something to enable Java to call ABAP and vice versa is readily apparent. SAP Java connector, also called “JCo”, is provided by SAP as a middleware component to do exactly that.

There are different distribution packages for various JRE versions and hardware processors available. They are available on the on the SAP Service Marketplace Web site.

The IBM WebSphere Business Integration Adapter for mySAP.com, and the WebSphere Adapter for SAP Software utilize JCo, but they provide functionality that fits with the J2EE programming model, the WebSphere integration products and the supporting development tools. Using JCo directly is possible but probably less productive in that environment.



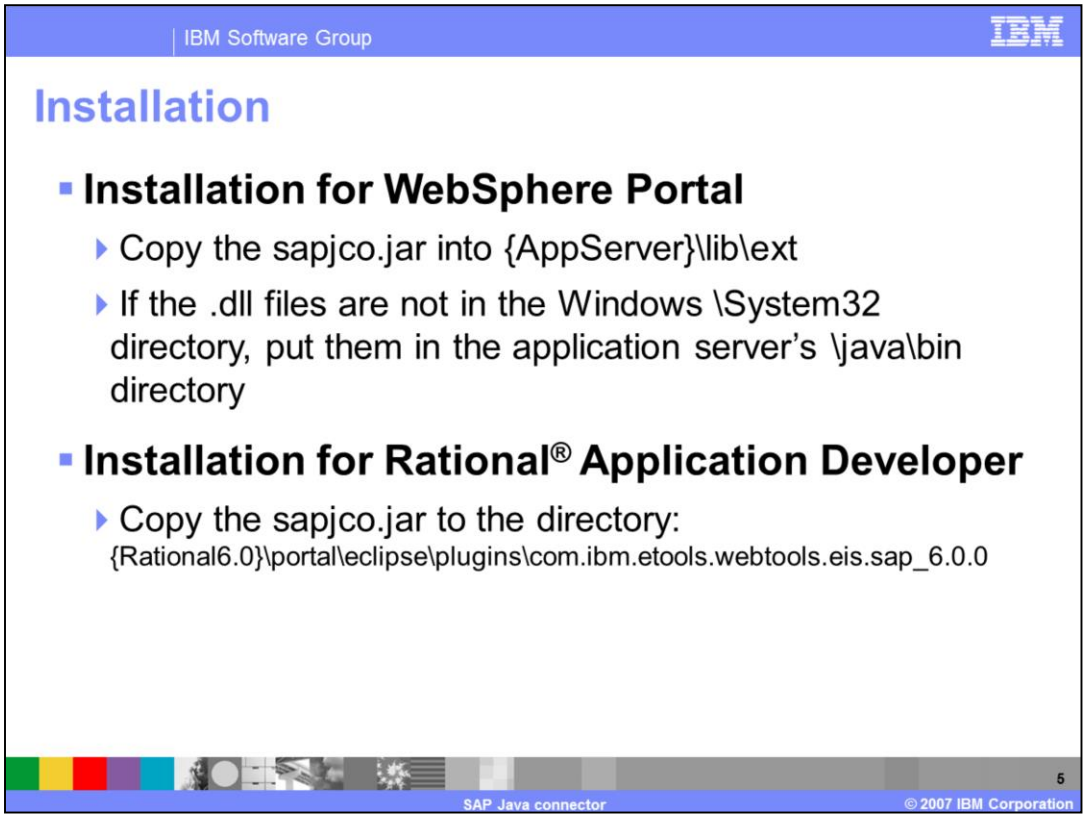
The next part of the presentation describes the installation of the SAP Java connector. At the time this presentation was written, the supported version is 2.1.8.



On Windows the SAP Java Connector version 2.1.8 requires a Java runtime environment, or JRE, of version 1.3 or higher. The 64 bit versions require JRE version 1.4 or higher.

To install JCo for Windows, unzip the appropriate distribution package into an arbitrary directory. Then copy the librfc32.dll in the {windows-dir}\system32 directory, add the installation path to the PATH environment variable, and add the archive sapjco.jar to your CLASSPATH environment variable. Different products may have their own directory requirements.

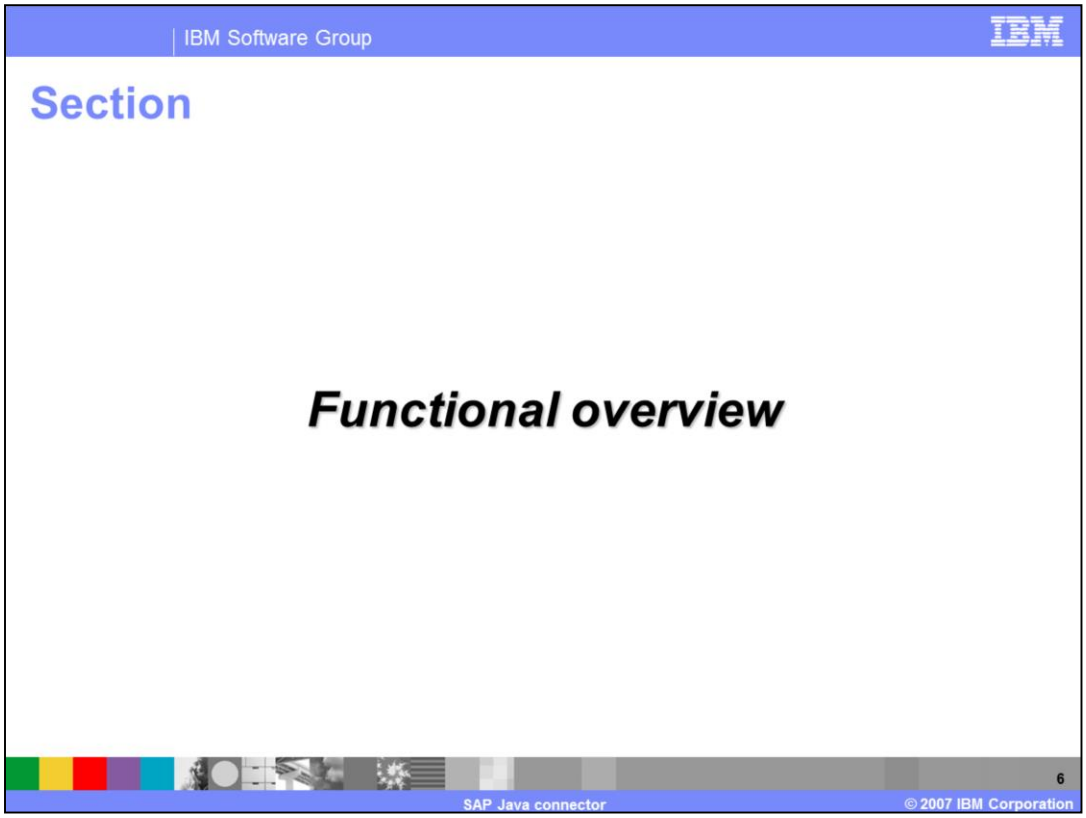
The latest version of the SAP Java connector can be downloaded from the SAP Service Marketplace if you have a valid SAP Marketplace user ID. There you will also find all of the available distribution packages for the various supported platforms and processors, and accompanying installation instructions.



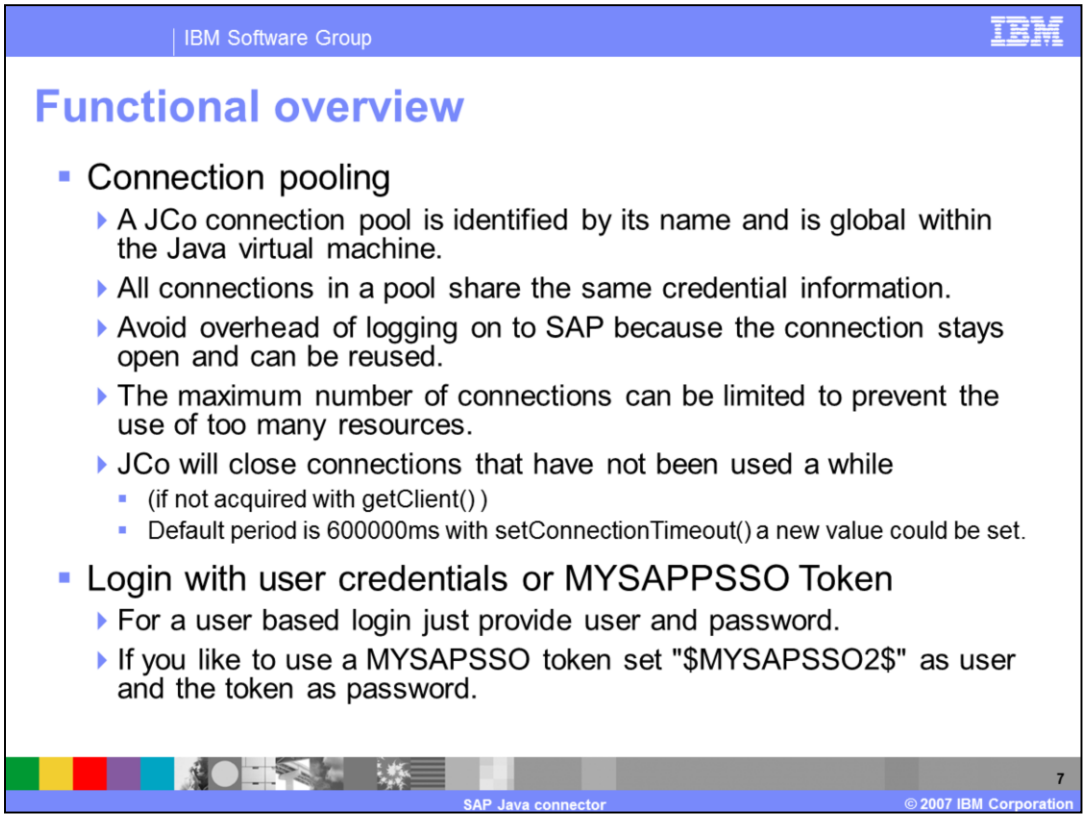
Because IBM cannot ship SAP code, it is necessary to enable the IBM products manually to work with JCo.

To install the SAP Java connector in the WebSphere Portal runtime environment, copy the sapjco.jar file into the {AppServer}\lib\ext directory to get it into the application server's CLASSPATH. If all the DLLs from the installation path are not in the System32 directory, place them in the {AppServer}\ java\bin directory.

To install the SAP Java connector in Rational Application Developer, copy the sapjco.jar to the directory shown here.



The next part of the presentation gives the functional overview of the connector.

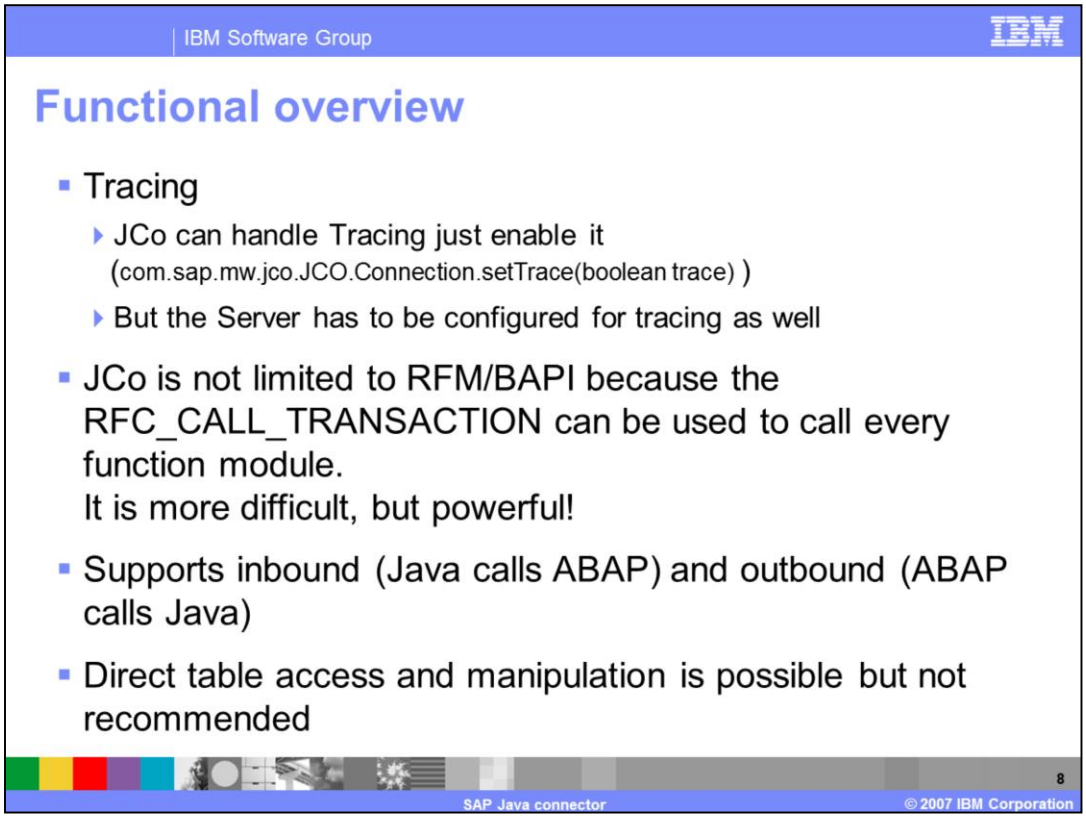


JCo is a high-performance, JNI-based middleware for SAP's remote function call protocol.

It has the ability to use connection pooling. Here are some important points to keep in mind while using connection pooling. A JCo connection pool is identified by its name and is global within the Java virtual machine. All connections in a pool share the same credential information. Using connection pool avoids the overhead of logging on to SAP because the connection stays open and can be reused. The maximum number of connections can be limited to prevent the use of too many resources. JCo will close connections that have not been used in a while, that is if they are not acquired with a getClient() call. The default connection expiration period is ten minutes. New value can be set with call to setConnectionTimeout().

New connection pool needs to have the logon properties specified – there are several ways to do that. For a user-based login, provide the user ID and password in the properties file.

Depending on the SAP system release, logins using Single-Sign-On or X509 certificates are supported. For SSO, specify the user to be $MYSAPSSO2$ and pass the base64 encoded ticket as the password parameter. For X509 specify the user to be $X509CERT$ and pass the base64 encoded certificate as the password parameter.

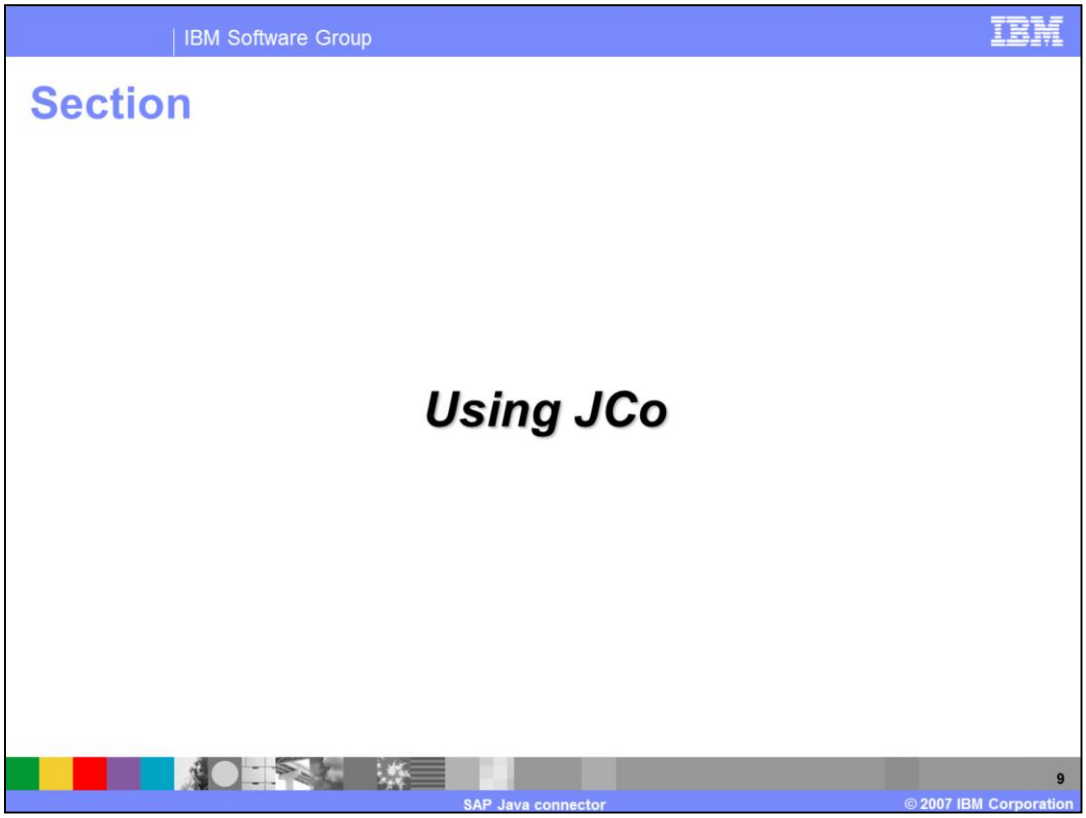


Another important function is tracing. JCo can handle tracing - just enable it by calling the method listed on this slide. But the server has to be configured for tracing as well.

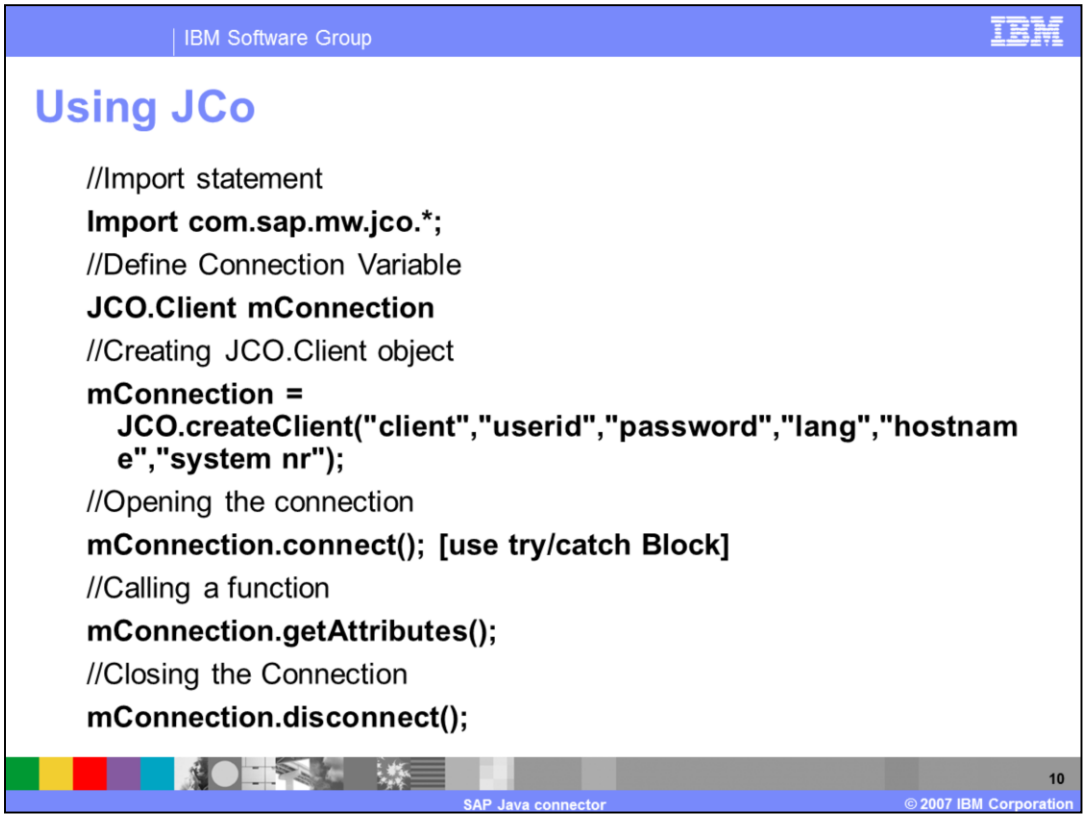
JCo is not limited to RFM / BAPI because the RFC_CALL_TRANSACTION can be used to call every function module. It is more difficult, but powerful!

JCo supports both directions – inbound, when Java calls ABAP, and outbound, when ABAP calls Java.

With JCo, direct table access and manipulation is possible, but not recommended.

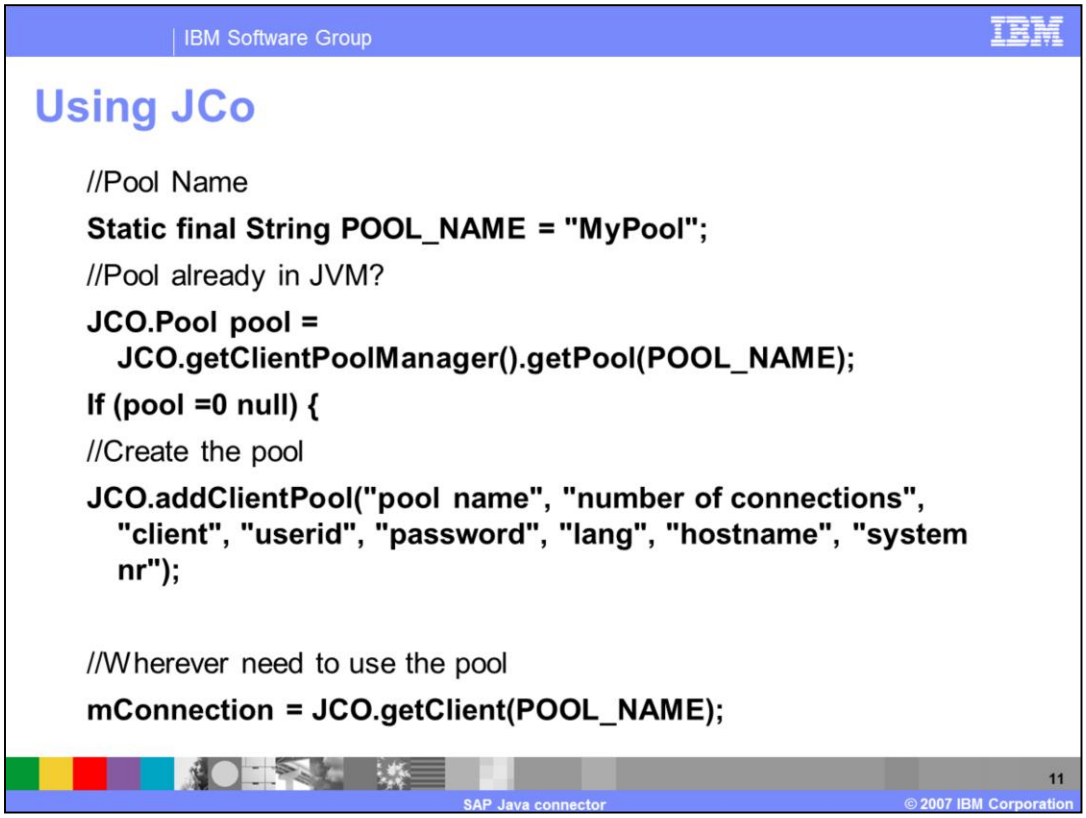


The last part of the presentation shows how to write code for the SAP Java connector client.

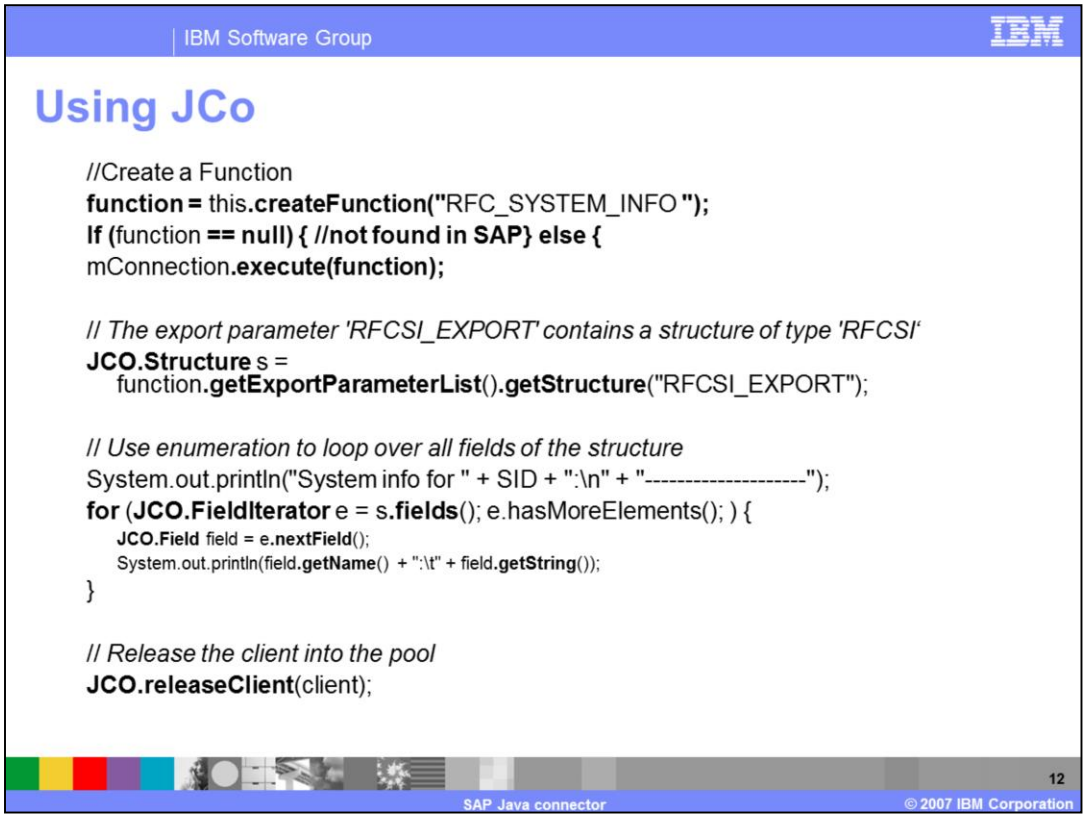


This slide shows code to use JCo to create a non-pooled connection, get the attributes object for this connection, and close it.

First import all JCO packages, define and create the JCO.Client object which holds all the information necessary to establish a connection to a remote JCO server (such as an SAP system). Open the connection using the connect() method. Be aware that the method can throw an exception. There are number of methods that can be called with an open connection, this slide show the method getAttributes(). Finally close the connection with call to disconnect().



This slide shows code to use JCo to create connection pool with name "MyPool", and instead of creating a connection as in the previous example, use the JCO.getClient() method to return a client connection from that pool. Note the call to JCO.PoolManager to check for existence of a pool with the same name before creating it.



Here is an example of how to create a JCO.Function object that represents function "RFC_SYSTEMS_INFO", call the function, use the method JCO.Function.getExportParameterList() to access the return parameters of the RFC_SYSTEMS_INFO, retrieve from it the structure "RFCSI_EXPORT", and iterate through it. In the end, the client connection is released to the pool. For the example to work it would require that the class implements a utility method createFunction() which uses *IFunctionTemplate* to create the actual object from repository.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM Rational WebSphere

Rational is a trademark of International Business Machines Corporation and Rational Software Corporation in the United States, Other Countries, or both.

Windows, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

J2EE, Java, Java runtime environment, JRE, JVM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.