IBM VisualAge® for Java™, Version 3.5

Access Builder and Connector for SAP R/3

Note!
Before using this information and the product it supports, be sure to read the general information under **Notices**.

Edition notice

This edition applies to Version 3.5 of IBM VisualAge for Java and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Access Builder and Connector for SAP R/3 1
Access Builder for SAP R/3 1
Connector for SAP R/3 2

Chapter 2. Base knowledge 7
SAP interface technologies 7
Supported SAP interface technologies 9
BAPI transactions 10
Advantages of using proxy beans 12

Chapter 3. Access Builder for SAP R/3 15
Local repositories 16
Reference documentation 17
Proxy beans for Business Objects 18
Enterprise bean proxies for Business Objects . . . 19
Proxy beans for RFC modules 21

Chapter 4. Base Connector for SAP R/3 23
Common RFC Interface for Java 23
Run-time classes for SAP R/3 26
BOR Access Classes 28
Connection pooling 31

Chapter 5. CCF Connector for SAP R/3 35

Chapter 6. RFC Server for SAP R/3 . . 39

Chapter 7. Utilities 43
Logon bean 43
saplogon.ini 48

Chapter 8. Using the Access Builder 51
Starting the Access Builder for SAP R/3 51
Starting the Access Builder for SAP R/3 outside of VisualAge for Java 52
Navigating through a repository 52
Searching for information 53
Creating a local repository 55
Loading a local repository 57
Updating a local repository 58
Generating proxy beans for SAP Business Objects and RFC modules 60
Generating documentation for SAP Business Objects and RFC modules 61
Customizing the Access Builder for SAP R/3 . . . 61

Chapter 9. Developing an application 65
Connecting to the SAP R/3 system 65
Integrating the Logon bean into your application . . 67
Using the proxy beans to access SAP Business Objects 73
Calling RFC modules 77

Chapter 10. Using the CCF Connector for SAP R/3 79
Setting up the CCF infrastructure 79
Building EAB Commands using proxy beans for RFC modules 79
Using enterprise bean proxies for SAP Business Objects 81
Deploying EJB beans into WepSphere Application Server Enterprise Edition 3.02 83

Chapter 11. Running and deploying your application 89
Running your application 89
Deploying your application 90
Running applets with JNI code in AppletViewer . . 92
Running applets with JNI code in Netscape Communicator 93
Running applets with JNI code in Java Plug-In . . 96

Chapter 12. Developing an RFC Server application - overview 99
Creating meta information for an RFC Server module 99
Adding business logic 101
Creating an RFC Server module factory 101
Adding transaction management 102
Adding security management 102
Configuring an RFC Server 103
Creating an RFC Server object 104
Creating an idle event listener 104
Completing the RFC Server application 105

Chapter 13. Performing specific tasks 107
Generating proxies from command line 107
Tracing the SAP R/3 connection of your application 112
Configuring SAP R/3 to call an RFC Server application from ABAP/4 114
Creating an ABAP/4 Report 117
Using BAPI Helpvalues 120
Invoking BAPIs and RFCs dynamically 120
Supporting a national language 121
Developing SAP R/3 applications using the InfoBus technology 122

Chapter 14. Samples 129
Sample: View documentation of Business Objects in IDES 4.5a 129
Sample: Retrieve company information with the proxy bean 129
Sample: Retrieve company information without proxy bean 130
Sample: Retrieve company information with the RFC module 131

Sample: Create a sales order manually 132
Sample: Retrieve company information visually 133
Sample: Submit a sales order 134
Sample: Retrieve company information with EJB beans 135
Creating the EJB group 136
Deploying the EJB beans 137
Running the sample 141
Sample: Retrieve company information with EJB beans visually 144
Sample: Creating sales orders with EJB beans in WebSphere Application Server Enterprise Edition . 148
Creating the EJB-JAR file 149
Building the sample application 151
Loading the application into the System Manager 151
Creating and configuring a server group on which the application is to run 152
Running the application 153
Working with the client GUI 155
Sample: Process purchase requisitions with EJB beans 156
Business process 156
Application internals 157
Running the sample 161
Providing logon information 161
Creating or deleting purchase requisitions . . . 162
Showing information about purchase requisitions 163
Changing purchase requisitions 164
Releasing purchase requisitions 165
Sample: Receiving stock prices from web using an RFC Server 166

Chapter 15. Appendix 171
ABAP/4 and Java data types 171
Naming conventions for enterprise bean proxies 172
Naming conventions for proxy beans 173
Naming conventions for RFC module proxy beans 174

Notices 177

Programming interface information 179

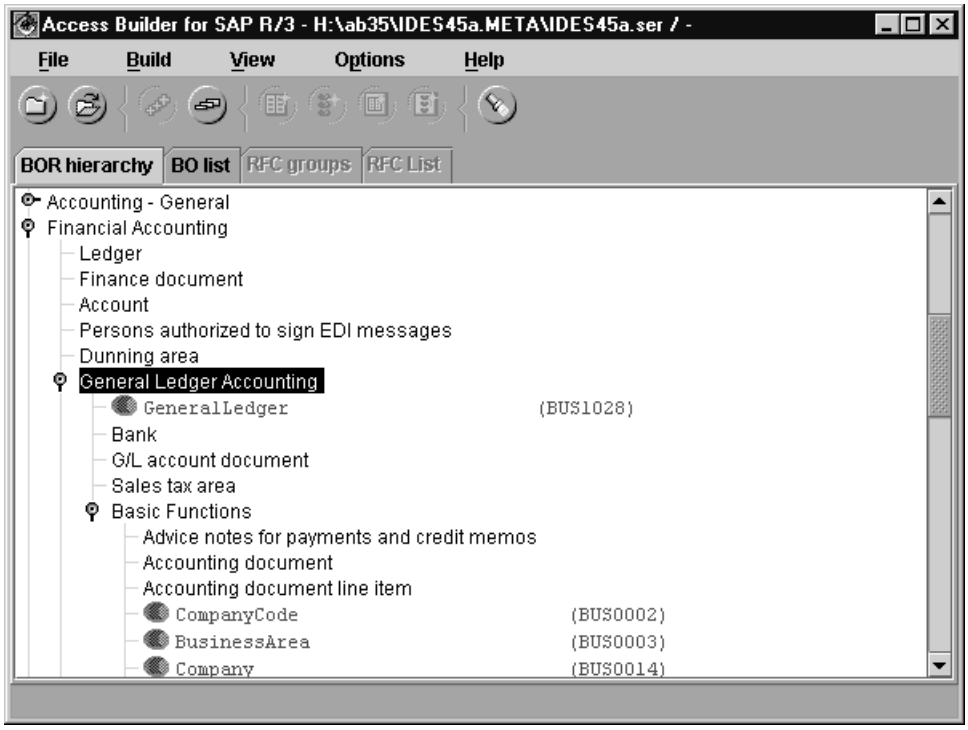
Trademarks and service marks 181

Chapter 1. Access Builder and Connector for SAP R/3

Access Builder and Connector for SAP R/3 enable you to develop Java applets, Java applications, Java servlets and EJB beans that access and manipulate data in SAP R/3 systems in an easy, productive and reusable way. As the product name indicates, it consists of two parts: the Access Builder and the Connector.

Access Builder for SAP R/3

Overview
The Access Builder for SAP R/3 tool is the developer frontend when working with the SAP R/3 system. The picture below shows the Access Builder displaying the SAP Business Object CompanyCode in the Business Object Repository (BOR) hierarchy.



Using SAP Business Objects, BAPIs, and RFCs
You use the Access Builder tool to browse meta information about SAP Business Objects and BAPIs of the SAP R/3 system. You can read the full reference of SAP Business Objects and BAPIs, disconnected from SAP R/3, with any Web browser.

The Access Builder allows you to do these things:

- Retrieve complete meta information from the Business Object Repository (BOR) within SAP R/3
- Keep meta information of multiple SAP R/3 systems
- Access meta information locally without a connection to SAP R/3

The Access Builder for SAP R/3 generates proxy beans for SAP Business Objects, their BAPIs, and RFC modules. You can use these beans to design your application visually or to manually create the application code.

You will use the Access Builder to:

- Generate Java proxy beans for SAP Business Objects, BAPIs, and RFC modules
- Generate EJB beans proxies for SAP Business Objects
- Generate HTML documentation for specific SAP Business Objects and RFC modules from the documentation stored in SAP R/3

Connector for SAP R/3

Overview

The Connector for SAP R/3 consists of three parts:

- The Base Connector for SAP R/3
- The CCF Connector for SAP R/3
- The RFC Server for SAP R/3

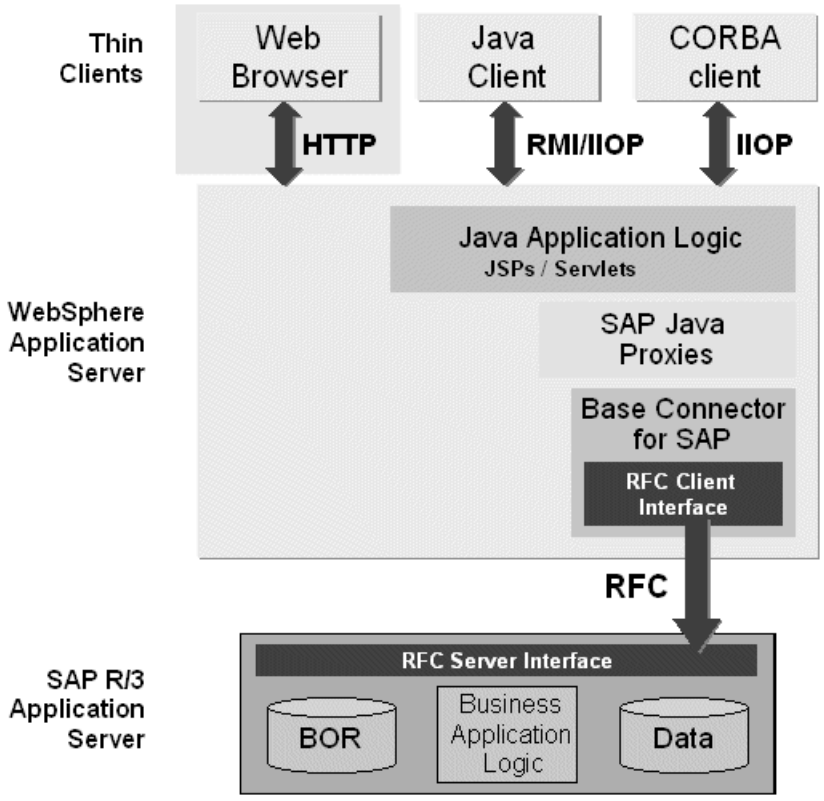
Each of those components is briefly discussed in the following.

Base Connector for SAP R/3

The Base Connector for SAP R/3 provides the runtime support for accessing SAP R/3 systems from Java applets, Java applications and Java servlets. It implements the client interface of the SAP RFC protocol according to the Common RFC Interface for Java. Through this RFC protocol it can send function calls from Java to SAP R/3.

The Common RFC Interface for Java defines a middleware-independent layer for accessing SAP R/3 from Java. You can leverage different ways to communicate with SAP R/3 in your application without recoding. All generated beans are based on this interface and provide you with the same flexibility. The Base Connector for SAP R/3 includes an actual layer via Java Native Interface (JNI). This communication layer is best suited for server-side Java or Java applications that are installed on fat clients.
You can also use the SAP Java Class Library (JRFC) as another communication layer. JRFC offers a pure Java library that accesses the SAP R/3 system through the JRFC server using CORBA/IIOP. This communication layer may be used for a quick start for intranet applications with client-side Java. JRFC is packaged in the SAP Automation Toolkit from SAP.

The picture below shows a simple web application scenario: A web application consisting of Java servlets and Java Server Pages (JSPs) running in WebSphere Application Server uses the SAP Java Proxies generated by Access Builder for SAP R/3 to connect to SAP R/3 via the Base Connector for SAP R/3. Supported client types for this scenario are HTTP clients like Web Browsers, PDAs etc. Other clients are not supported.



This scenario is a lightweight web application scenario without any further exploitation of the web application server's infrastructure. Therefore it is not web application server specific. For a better integration with the services of the web application server please have a look at the CCF Connector for SAP R/3.

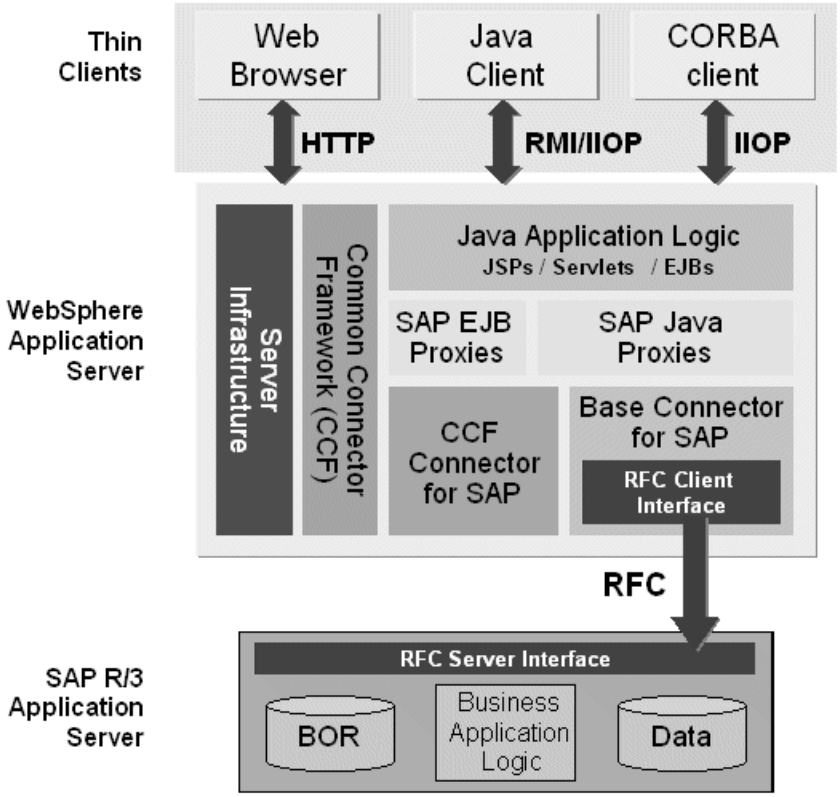
In addition to the features above the Base Connector for SAP R/3 provides a set of utility classes to ease the development of SAP Java applications like e.g. a fully customizable SAP Logon dialog or classes to display the BOR hierarchy.

CCF Connector for SAP R/3
The CCF Connector for SAP R/3 is built on top of the Base Connector for SAP R/3 explained above. This connector is fully implementing the Connector Interfaces of the Common Connector Framework (CCF).

This has the advantage that applications built on top of the Connector for SAP R/3 can be used on any server infrastructure which provides an implementation of the CCF infrastructure. To learn more about the Common Connector Framework (CCF) and its advantages please refer to the topic Overview in the CCF documentation.

Implementing the CCF connector interfaces makes the Connector for SAP R/3 consistent with other e-business connectors. Therefore writing applications using the Connector for SAP R/3 follows the same programming paradigm as programming the other connectors. The Access Builder for SAP R/3 is the tool you use at development time to generate the SAP specific classes. You can then use the Enterprise Access Builder (EAB) tool together with these generated classes to build EAB specific Commands and Navigators. This gives you the full flexibility of combining functions from different SAP objects and Remote Function Calls (RFCs) in one EAB Command or Navigator.

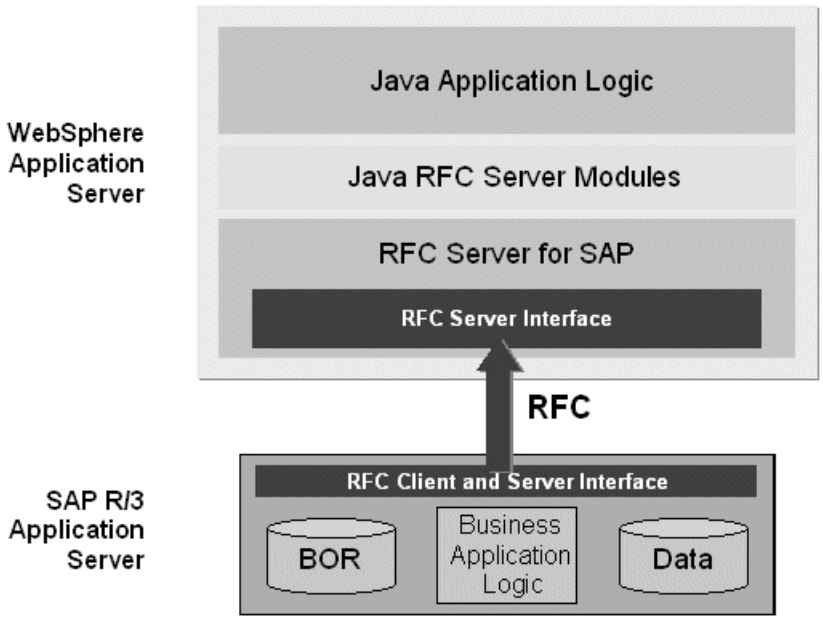
It is also possible to use the Access Builder for SAP R/3 to directly generate enterprise bean proxies. These EJB beans are also based on the Common Connector Framework. You can deploy them on any enterprise beans Server (EJS) providing support for stateful Session EJB beans and an implementation of the CCF. Additionally these generated EJB beans provide the client with a smart table cache to avoid long download times for large tables. If it is sufficient to do a one to one mapping of SAP business objects to EJB beans then this is the easiest approach. In all other cases you should use the EAB tool.



Scenarios using the CCF Connector for SAP R/3 add support for Sun's Enterprise JavaBeans and through this the support for a broader range of clients. The picture above shows the extended web application scenario: Using the CCF Connector for SAP R/3 allows to also support Java and CORBA client applications. In addition the web application now can benefit from the web application server's infrastructure services like security, naming and transactions through the use of the Common Connector Framework (CCF) and the according CCF Connector for SAP R/3.

RFC Server for SAP R/3

The RFC Server for SAP R/3 allows arbitrary RFC clients to call Java RFC Server Modules running in a Java RFC Server. Currently no specific WebSphere services are used. Therefore the RFC Server for SAP R/3 can run in all kinds of Java Servers, even in a simple Java Runtime Environment (JRE). This allows support for multipel different scenarios. The scenario outlined in the picture below is especially interesting: Calling Java RFC Server Modules from an ABAP/4 program running on the SAP R/3 application server.



The RFC Server for SAP R/3 provides the runtime and framework for building RFC Server applications in Java. The RFC Server for SAP R/3 enables you to implement function modules in Java that can be called from any RFC client. These function modules can either do simple calculations or use other connectors to retrieve and updata data in other Enterprise Information Systems (EIS).
Using the RFC Server for SAP R/3 together with the Base Connector for SAP R/3 or the CCF Connector for SAP R/3 it is possible to implement a bi-directional bridge between SAP and another EIS like CICS, PeopleSoft etc.

RELATED CONCEPTS

Access Builder for SAP R/3
Base Connector for SAP R/3
CCF Connector for SAP R/3
RFC Server for SAP R/3
Connectors: Overview

Chapter 2. Base knowledge

SAP interface technologies

SAP provides several interface technologies to integrate third-party applications with the SAP R/3 system.

- Remote Function Call (RFC)
- Business Objects and Business Application Programming Interfaces (BAPI)
- Business Object Repository

Remote Function Call (RFC)
The RFC technology is a platform-independent solution for remote communication such as

- Communication between two independent SAP R/3 systems
- Client/server communication between an external client and an SAP R/3 system acting as the server
- Client/server communication between an SAP R/3 system acting as the client and an external server

The RFC communication protocol used between the external client and the SAP R/3 system is not public. The base for all communication with external clients is the RFC Library, a platform-independent library of C routines with a published API. All other SAP interface technologies, including BAPIs and Application Link Enabling (ALE), are based on RFC.

Business Objects and BAPIs
SAP introduced the object technology in SAP R/3 3.0 from a business point of view instead of from a purely technical point of view. The Business Framework provides Business Objects that encapsulate business logic and business workflow on both technical and business levels. Business Objects offer robust interfaces to several business processes and a range of business data. These interfaces are called the Business Application Programming Interface (BAPI). With BAPI methods which represent high-level object methods it is possible for SAP R/3 Business Objects to communicate with other components.

A Business Object refers to a specific object class, also named `object type`, in the SAP context. Often, the term business object is also used for the actual instances of a Business Object. For example, “Mrs. Kate Bush with EmployeeNo. 9876”, is an instance of the object type named “Employee” which “exists” in the Business Object Repository.

A Business Object instance has a persistent data representation in the SAP R/3 database. For identification, each Business Object instance has one or more object keys. BAPI methods are used to work with this persistent data. There are three types of BAPI methods:

- Class methods
  Class methods may be invoked without association to a Business Object instance, for example methods to retrieve lists of existing Business Objects.
- Instance methods
  Instance methods may only be invoked for Business Object instances, for example methods to retrieve or manipulate the persistent data of a Business Object instance.

- Factory methods
  Factory methods create the persistent data representation for a new Business Object instance.

For example, the Business Object CompanyCode has two BAPI methods to access internal company data:

- getlist - this class method retrieves a list of existing companies
- getdetail - this instance method retrieves additional information about a specific company

External applications can only invoke these two methods. In other words, the BAPI methods are the only way to manipulate the Business Object information. The method getlist of the Business Object CompanyCode retrieves a list showing all existing instances of the object type CompanyCode. The external application invokes the BAPI method with its specific import, export, and table parameters.

Instance methods use the Business Object keys to identify the persistent representation of the instance. For example, in the SAP IDES R/3 system there exists a company called IDES AG with the key 001000. That means that a Business Object instance of the Business Object class Company (Business Object type BUS0014) with the key 001000 exists. A BAPI method that requires an instance of Company could be called with the appropriate key.

In the existing SAP R/3 versions, BAPI methods are implemented through RFC modules. Invoking the implementing RFC module is a very efficient way to call the BAPI methods.

Business Object Repository (BOR)
The Business Object Repository represents the business data, transactions, and events within the SAP R/3 system as an integrated set of SAP Business Objects. The BOR manages these objects which are related to each other in object models.

You can use the SWO2 or BAPI transactions on an SAP R/3 system to request information about SAP Business Objects. Alternatively, you can use the Access Builder for SAP R/3 to view the information stored in the SAP R/3 system.

In addition to this meta information, the Business Object Repository may also contain run-time instances. It is possible to create transient Business Objects in the BOR. Transient Business Objects are not synchronized with their persistent data representations in the SAP R/3 database. BAPI methods always deal with the persistent data, which is guaranteed to be consistent.

It is possible to invoke BAPI methods through the Business Object Repository. However, this is very inefficient compared to a direct invocation of the implementing RFC module.

RELATED CONCEPTS

Supported SAP interface technologies
BAPI transactions
Open BAPI Network - SAP web site for BAPI programming
SAPnet - SAP web site for extensive information to customers

Supported SAP interface technologies

The Access Builder and Connector for SAP R/3 fully supports both SAP interface technologies, the new object-oriented Business Application Programming Interfaces (BAPI) within the Business Framework as well as the traditional Remote Function Call (RFC). It also provides access to the meta information stored in the Business Object Repository, and the Access Builder for SAP R/3 utilizes this meta information for the generation of wrapper beans and EJB beans.

The Access Builder and Connector for SAP R/3 leverages the Common RFC Interface for Java, a middleware-independant layer for RFC access. Thus, applications built with the Access Builder and Connector for SAP R/3 can leverage different SAP R/3 access methods at run time without recoding.

The Access Builder and Connector for SAP R/3 support for the RFC interface offers

- Mapping of all ABAP data types to native Java data types and classes
- Caching of tables
- Invocation of RFC modules on the SAP R/3 system from the Java client
- Support of optional parameters
- Invocation of Java methods from the SAP R/3 system (JNI middleware only)
- Transactional RFCs (JNI middleware only)
- Thread-safe interface on all supported platforms (JNI middleware only)
- Access to meta information for RFC modules, structures, and tables
- Generation of RFC command beans

The support for the Business Objects and BAPIs covers

- Access to meta information in the Business Object Repository
- Generation of beans and EJB beans for Business Objects
- Intuitive mapping of class, instance, and factory BAPIs to Java static methods, instance methods, and constructors.
- Additional support for using BAPI Helpvalues in the user interface

The Access Builder and Connector for SAP R/3 does not support transient Business Objects in the BOR. For performance reasons, BAPIs are directly invoked through the RFC modules which implement them.

RELATED CONCEPTS

EJB Architecture
Access Builder for SAP R/3
SAP interface technologies
Common RFC Interface for Java
Run-time classes for SAP R/3
BOR Access Classes
Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules

RELATED TASKS

Starting the Access Builder for SAP R/3
Connecting to the SAP R/3 system
Calling RFC modules

RELATED REFERENCES

ABAP/4 and Java data types
API Reference

BAPI transactions

Beginning with SAP R/3 4.0 a new concept was introduced to BAPIs. The BAPIs which are introduced with SAP R/3 4.0 and later releases do not commit their changes automatically as in previous releases. The program executing a BAPI now has the choice to commit or rollback changes applied to one or more Business Objects using BAPIs. For this purpose there are two new BAPIs in SAP R/3 4.0:

BAPI_TRANSACTION_COMMIT

- ends current transaction
- commits changes (changes become persistent)
- starts a new transaction

BAPI_TRANSACTION_ROLLBACK

- ends current transaction
- discards changes
- starts a new transaction

Please note that older BAPIs automatically call BAPI_TRANSACTION_COMMIT after their execution.

The scope of a transaction is the connection to the SAP R/3 system. When the connection is opened then the transaction starts. To keep the interface straight the JNI middleware which IBM offers implements the following two methods extending the Connection interface of the Common RFC Interface for Java:

```
/**
  * Commits the transaction on the SAP R/3 System.
  *
  * @exception JRfcMiddlewareException
  *  a problem occurred in the specific middleware implementation
  *  consult the documentation of this middleware implementation for details
  * @exception JRfcRemoteException
  *  a problem occurred while committing the current transaction
  */
 public void bapiCommit() throws JRfcMiddlewareException, JRfcRemoteException;
 /**
  * Performs a Rollback of the current transaction.
  *
  * @exception JRfcMiddlewareException
  *  a problem occurred in the specific middleware implementation
  *  consult the documentation of this middleware implementation for details
  * @exception JRfcRemoteException
  *  a problem occurred while committing the current transaction
  */
 public void bapiRollback() throws JRfcMiddlewareException, JRfcRemoteException;
```

Please note that these methods are not yet part of the Common Interface for Java. Therefore other middleware providers currently will not implement them. To avoid incompatibilities the default behaviour of the JNI middleware is that the user does not have to use the new APIs - even when working with the new BAPIs. To explicitly enable one-phase-commit behaviour for new BAPIs you have to set the bapiTransactionEnabled property in your connection object. If you don't do that then the default behaviour is to do an implicit BAPI_TRANSACTION_COMMIT

after each BAPI call. This makes your application backwards compatible. The following sample illustrates how to use 4.0 BAPIs together with these methods:

```
...
// Create connection
IRfcConnectionFactory aConnectionFactory =
FactoryManager.getSingleInstance().getRfcConnectionFactory() ;
IRfcConnection connection =
aConnectionFactory.createRfcConnection(fieldConnectInfo, fieldUserInfo) ;
// Check if we have a JNI connection - otherwise
// we do not know how to commit or rollback so far...
if (connection instanceof com.ibm.sap.bapi.jni.Connection)
{
  ( (com.ibm.sap.bapi.jni.Connection) connection).setBapiTransactionEnabled(true) ;
}
// Create 4.0 Business Objects:_
BO1 bo1 = new BO1("KeyStringBO1") ;
BO2 bo2 = new BO2("firstKeyStringBO2", "secondKeyStringBO2", thirdKeyBigNumberBO2) ;
// Open connection - transaction begins implicitely
connection.open() ;
// Invoke (multiple) BAPI methods
bo1.method1(connection, bo1Method1Params) ;
bo1.method2(connection, bo1Method2Params) ;
...
bo2.method1(connection, bo2Method1Params) ;
bo2.method3(connection, bo2Method3Params) ;
// Check if we have a JNI connection -
//otherwise we do not know how to commit or rollback so far...
if (connection instanceof com.ibm.sap.bapi.jni.Connection)
{
  com.ibm.sap.bapi.jni.Connection jniConnection = (com.ibm.sap.bapi.jni.Connection) connection ;
  // Check if commit or rollback
  if(condition)
  {
    // Commit changes
    jniConnection.bapiCommit() ;
  }
  else
  {
     // Rollback changes
     jniConnection.bapiRollback() ;
  }
}
...
```

There are some important considerations to make:

- Invoking a BAPI method which was defined before SAP R/3 4.0 will immediately commit the current transaction and start a new transaction. This is not desireable when using 4.0 BAPIs with explicit transactional behaviour. If you want to achieve a one-phase commit behaviour you cannot mix BAPIs with different transactional behaviour.
- Currently there is no way to retrieve meta information from the BOR which indicates if a BAPI method uses an implicit or explicit transaction mode. Therefore the classes which the Access Builder for SAP R/3 generates cannot provide the user with the respective information. The only way to figure out which transaction mode a BAPI method uses is to read the respective documentation.

RELATED CONCEPTS

SAP interface technologies
Supported SAP interface technologies
Base Connector for SAP R/3
CCF Connector for SAP R/3

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules
Connecting to the SAP R/3 system

RELATED REFERENCES

ABAP/4 and Java data types
Run-time classes for SAP R/3

Advantages of using proxy beans

There are many reasons why using the generated proxy classes is better then handcoding your SAP access based on the Common RFC Interface for Java. Here are some of them:

Abstraction
The generated proxy classes provide a level of abstraction on top of the Common RFC Interface for Java. They are designed to ease the creation of a Java based application accessing SAP R/3.

Meta information
The generated proxy classes contain all the meta information needed to invoke a call against an SAP R/3 system. This meta information is retrieved at development time. If not using proxies then the developer would have to handcode the meta information which is time consuming and error prone. Alternativley the meta information could be retrieved at runtime using the mechanism described below (Why should I use the generated proxy classes for RFCs instead of using the autocreate functionality in com.sap.rfc.IRfcModuleFactory?). The effect in this case is a dramatic degradation in runtime performance.

Data type handling / static typing
The generated proxy classes provide you with a Java accessor method for each SAP data type. That means you are dealing directly with Java data types like java.lang.String, int and char instead of manually converting between SAP and Java data types back and forth. Using the Java data types instead of converting to the correct data type at runtime gives you more type safeness. In case of a programming error you get an error message at compile time instead of a conversion error at runtime.

Object orientation
When using proxy classes for SAP Business Objects (BOs) it is possible to work in an object oriented way:

- Invoke static methods for instance independend BAPIs
- Create objects of a specific BO type referring to an instance with a specific key in SAP R/3
- Invoke instance methods on BO objects with implicit key handling
- Create new BO instances in SAP R/3 and Java using the standard Java constructor mechanism

Full exploitation of different versions of SAP R/3 at the same time
With Java proxy classes it is possible to connect to SAP R/3 systems of different versions in one application using different sets of Java proxies. This gives the application the possibility to be backward compatible and exploit all the new features of a new SAP R/3 release in a Java like, typesafe way.

Tooling
The Java proxy classes are the foundation blocks for the VisualAge for Java Enterprise Access Builder and the CCF Connector for SAP R/3.

RELATED CONCEPTS

Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules
Access Builder for SAP R/3
Base Connector for SAP R/3
Common RFC Interface for Java

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules
Generating proxies from command line
Invoking BAPIs and RFCs dynamically

RELATED REFERENCES

Naming conventions for generated classes
Naming conventions for proxy beans

Chapter 3. Access Builder for SAP R/3

The Access Builder for SAP R/3 lets you access meta information of SAP Business Objects (BOs), BAPIs, RFCs, and structures and tables within your SAP R/3 system. You can save this meta information locally. You can also store multiple sets of meta information of different SAP R/3 systems locally. Locally stored meta information represents a snapshot of the current state of your SAP R/3 system.

You can use this meta information to generate HTML documentation and Java proxy beans for all contained Business Objects and Remote Function Call (RFC) modules. Generating Java proxy beans from meta information does not require an active connection to an SAP R/3 system. However, testing your written Java code, which uses the generated Java proxy beans, does require a connection to an SAP R/3 system.

You can apply these proxy beans to develop Java applications, applets, and servlets, and, whether you use the Visual Composition Editor or do traditional handwritten coding, your development effort will be reduced.

Information displayed by the Access Builder
The Access Builder displays the following kinds of meta information:

Business Object Repository (BOR)
The Business Object Repository organizes Business Objects in business domains and sub-domains.

The Access Builder provides two views of a Business Object Repository:

- A BOR hierarchy displayed like an information tree
- An alphabetically sorted Business Object list

RFC module repository
The RFC module repository organizes all RFC modules in logical groups.

The Access Builder provides two views of an RFC module repository:

- An alphabetically sorted list of the RFC groups and their members
- An alphabetically sorted RFC module list

Navigating through the repositories
After retrieving or loading repositories into the Access Builder you can navigate through the different views provided for the available repositories.

You navigate by scrolling, expanding, and collapsing information in the appropriate view. A search facility supports quick locating of specific information.

Generating HTML documentation and Java proxy beans
For all Business Objects and RFC modules you can generate HTML documentation as well as Java proxy beans. For Business Objects, you can also generate EJB bean proxies.

Using proxy beans in your applications you can do the following:

- Query information from SAP R/3
- Manipulate data in SAP R/3

- Create new data records and Business Object instances in SAP R/3

RELATED CONCEPTS

SAP interface technologies
Supported SAP interface technologies
Local repositories
Reference documentation

RELATED TASKS

Starting the Access Builder for SAP R/3
Starting the Access Builder for SAP R/3 outside of VisualAge for Java
Navigating through a repository
Creating a local repository
Generating proxy beans for SAP Business Objects and RFC modules
Customizing the Access Builder for SAP R/3

RELATED REFERENCES

Naming conventions for generated classes
BO and RFC proxy beans

Local repositories

The meta information for Business Objects and BAPIs is stored in the Business Object Repository (BOR) within the SAP R/3 system. In addition, the SAP R/3 system provides functions to also retrieve the meta information about RFC modules and their parameters.

The Access Builder for SAP R/3 stores meta information from SAP R/3 systems locally. This allows you to do most of your work with the Access Builder without being connected to any SAP R/3 system. Not having to retrieve information from the SAP R/3 system reduces processing time.

The locally stored meta information is grouped into two local repositories:

- Business Object Repository (BOR)
- RFC module repository

Business Object Repository (BOR)
A locally stored Business Object Repository consists of the following elements:

BOR information file
A BOR information file contains the hierarchy of all business domains and Business Objects. This is a binary file.

BO information files
For each Business Object, a BO information file contains syntactical information about such things as keys, attributes, methods, and events. This is a binary file. The default name is `business_object_type.ser`

HTML hierarchy and HTML index files
An HTML hierarchy file contains the hierarchy structure of the BOR. An additional HTML index file contains an alphabetically sorted list of all Business Objects inside the BOR with links to the corresponding documentation of each Business Object.
These HTML files are named `name_of_BOR_information_file.html` and `index.html`.

BO documentation files
The BO documentation file contains the documentation for each Business Object. The documentation can consist of syntactical information retrieved from the BOR and semantic information retrieved from the online reference of the SAP R/3 system.

RFC module repository
A locally stored RFC module repository consists of the following elements:

RFC repository information file
An RFC repository information file contains the logical grouping of all existing RFC modules. This is a binary file.

RFC information files
For each RFC module, an RFC information file contains information about parameters and documentation. This is a binary file. The default name is `rfc_name.ser`

RFC documentation files
The RFC documentation file presents the documentation contained in the corresponding RFC information file in HTML format.

RELATED CONCEPTS

Access Builder for SAP R/3
SAP interface technologies
Reference documentation

RELATED TASKS

Creating a local repository

Reference documentation

The Access Builder for SAP R/3 retrieves information about the SAP Business Objects and RFC modules and generates online documentation locally. This reference documentation gives you an overview of all features that Business Objects or RFC modules offer to the outside world.

Reference documentation for Business Objects
The following properties of a Business Object are documented:

Object type
The object type describes the basic object data with its unique name, classification, and the underlying object model.

Key fields
The key fields of an object describe a unique index to access a specific instance of the SAP Business Object. Consider as an example the SAP Business Object `Employee` and its key field `Employee Number`.

Attributes
An attribute stores object data for a specific object instance, in our example the name of an employee or the date of birth.

Methods
A method performs an operation which can manipulate object attributes. A method is defined by a method name and a couple of import/export

parameters and exceptions. BAPIs are special methods which can be invoked by external applications and provide a stable interface for these external applications.

Events

An event indicates that one or more attributes have been changed or something else happened to a specific instance of a Business Object.

You find the BAPIs in the methods section of the documentation. They are listed with the required parameters:

- Import parameters
- Export parameters
- Table parameters which you can use for import and export purposes

Reference documentation for RFC modules

The following properties of an RFC module are documented:

Method

A method performs the specific operation of the RFC module in the SAP R/3 system. It is defined by a method name and a couple of import and export parameters. These parameters depend on the RFC module and are listed as follows:

- Import parameters
- Export parameters
- Table parameters which you can use for import and export purposes

RELATED CONCEPTS

Access Builder for SAP R/3
SAP interface technologies
Local repositories
Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules

RELATED TASKS

Creating a local repository
Navigating through a repository

RELATED REFERENCES

BO and RFC proxy beans

Proxy beans for Business Objects

A Business Object proxy bean represents a Business Object. It contains constructors to instantiate the bean and methods that allow the invocation of BAPI methods.

The proxy beans for Business Objects are intended to be used in Java applets, applications, and simple servlets. They do not require any special infrastructure except a Java virtual machine and the Base Connector for SAP R/3 runtime library, plus a web application server like WebSphere Application Server if used by servlets.
The proxy beans contain constructors for instantiation and initialization, a set of methods to invoke the BAPI methods, and methods for the key handling. Input and output parameters of these methods are contained in special parameter

container classes. These parameter classes as well as the required structure and table helper classes are generated automatically when generating a BO proxy.

Parameter Container Classes
A parameter container class is generated for each BAPI method of the Business Object. This class is used as a container for the method parameters. The use of a special parameter container class might seem cumbersome for methods with few parameters. However, many BAPI methods provide a lot of parameters, and it is convenient to keep all of these parameters together in a container instance. The parameter class makes instantiating simple parameters easy, as it knows the necessary meta information. In addition using a parameter class makes client programs robust against changes in the parameter order of a method in an SAP Business Object.

The parameter container class has access methods for setting and getting the parameters. You use the set methods to provide the required and optional input parameters before invoking a BAPI method. Afterwards you retrieve information from the output parameters using the get methods.

RELATED CONCEPTS

Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules
Access Builder for SAP R/3
Base Connector for SAP R/3
Common RFC Interface for Java
Run-time classes for SAP R/3

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules
Generating proxies from command line
Using the proxy beans to access SAP Business Objects
Deploying your application
Running your application

RELATED REFERENCES

Naming conventions for generated classes
Naming conventions for proxy beans
BO and RFC proxy beans
ABAP/4 and Java data types

Enterprise bean proxies for Business Objects

Access Builder for SAP R/3 is able to generate stateful EJB session beans for Business Objects including the respective remote and home interfaces. These EJB beans provide a one-to-one mapping between a SAP Business Object on the one side and a stateful Session EJB on the other side. If another mapping than this is desired then the use of IBM's Enterprise Access Builder together with Access Builder for SAP R/3 generated RFC proxies is the way to go. To find more information about this please refer to chapter 'Building EAB Commands using SAP proxy beans'. To get there you may want to use the respective link under the category *Related Tasks* at the bottom of this document.

The EJB beans generated by Access Builder for SAP R/3 contain constructors for instantiation and initialization, a set of methods to invoke the BAPI methods, methods to set and get the object's keys, and access methods for the parameters of each method. The EJB bean can be considered as a container for all the BAPI methods and the respective parameters of an SAP Business Object.

Keys
The keys of a Business Object are also represented as EJB bean properties. The property name is the name of the key prepended by the string ″key_″. For example, the CompanyCode Business Object has the property key_Companycodeid (resulting in the generated access methods getKey_Companycodeid() and setKey_Companycodeid(value)).

BAPI methods
A BAPI method is represented by a remote method of the enterprise bean. The remote method has the same name as stored in the BOR (for example getdetail()).

Before invoking a BAPI method on an EJB bean the input parameters of this method have to be set. For instance methods, the Business Object's key(s) must also be set. When invoking a factory method the Business Object's key(s) is (are) set to the key values returned by the factory method.

BAPI parameters
The remote method does not have any parameters. The importing, exporting and table parameters of the BAPI are stored as properties of the EJB bean instead. Before the invocation of a given BAPI, the corresponding bean attributes for the import parameters must be set. The resulting export parameters are accessible through the corresponding bean attributes after the call.

In order to distinguish between the different BAPI method parameters, the generated EJB bean property name consists of two parts: The left part of the name is the name of the method to which the property belongs, the right part is the parameter name within the method. Both parts are separated by an underscore '_'. For example, the CompanyCode object has the property getdetail_CompanycodeAddress representing the parameter CompanycodeAddress in the Business Object method getdetail.

There are two access methods generated for this property:

- getGetdetail_CompanycodeAddress()
- setGetdetail_CompanycodeAddress(value)

BAPI parameters that are part of the Business Object's key are not represented explicitly as EJB bean properties, as their values are already accessible through the key properties.

Tables
Tables are also instantiated as EJB beans. This assures that a client process does not need to transfer a whole table from the server in order to start processing its entries. In contrast, the client can work remotely with the table, transferring only the data that it is interested in. This saves memory at the client side and does not stress the network connection too much (imagine how long it would take to transfer a complete serialized table of 10 MB size to the client before the first row could be accessed...).

On the other hand, accessing each row of table data remotely would produce much more network traffic because of every single row being requested and transported

separately. In order to minimize the network traffic and the accompanying communication overhead, every client side table object works with a default caching mechanism that stores a certain number of rows and requests a block of rows from the server as they are needed. It is implemented as a read ahead cache that stores 100 rows and requests blocks of 50 rows by default. The caching feature provides the EJB bean client with acceptable response times.

The table EJB beans do not have to be created manually or by the application. They are created automatically by the Business Object EJB beans, as they are needed. However, as nearly every EJB container uses different naming services and mechanisms to narrow the retrieved objects, an EJB container specific helper class is used to create at least an EjbTableHome instance.

The helper class implements the interface com.ibm.sap.bapi.ejb.EjbTableHomeFactory which contains the method createEjbTableHome() returning a com.ibm.sap.bapi.ejb.EjbTableHome interface instance. An implementation of this helper class for WebSphere Application Server is included.

RELATED CONCEPTS

Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules
Connectors: Overview
EJB Architecture
CCF Connector for SAP R/3
Access Builder for SAP R/3
Base Connector for SAP R/3
Common RFC Interface for Java
Run-time classes for SAP R/3

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules
Generating proxies from command line
Using the proxy beans to access SAP Business Objects
Setting up the CCF infrastructure
Building EAB Commands using proxy beans for RFC modules
Running your application
Deploying your application

RELATED REFERENCES

Naming conventions for generated classes
Naming conventions for proxy beans
BO and RFC proxy beans
ABAP/4 and Java data types

Proxy beans for RFC modules

You can generate RFC proxies to access RFC modules. An RFC module is a remotely accessible function on an SAP R/3 system. Every RFC module has a specific task and is separated from any other RFC module. You have to provide all information required to execute an RFC module as parameters.

For all structure and table parameters, the required wrapper classes are generated automatically when generating the RFC proxy.

From a non-object-oriented view, all BAPIs (methods of SAP Business Objects) can be seen as RFC modules. Therefore BAPI methods can be accessed using the mechanism for RFC modules as well as by using BAPI methods. Because the mechanism for RFC modules does not know anything about object instances and keys this information must be manually handled and explicitly provided as parameters.

RELATED CONCEPTS

Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules
Access Builder for SAP R/3
Base Connector for SAP R/3
Common RFC Interface for Java
Run-time classes for SAP R/3

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules
Generating proxies from command line
Calling RFC modules
Running your application
Deploying your application

RELATED REFERENCES

Naming conventions for generated classes
Naming conventions for proxy beans
BO and RFC proxy beans
ABAP/4 and Java data types

Chapter 4. Base Connector for SAP R/3

The Base Connector for SAP R/3 is part of the Access Builder and Connector for SAP R/3 component. It contains the run-time classes needed to access SAP R/3 systems and leverages the Common RFC Interface for Java, a middleware-independant layer for RFC access. Thus, applications built with the Base Connector for SAP R/3 can use different SAP R/3 access methods at run time without recoding. A JNI implementation of the Common RFC Interface for Java is included in the toolkit. This implementation is best suited for Java Server Applications but may also be used to develop standalone Java applications or Java applets.

The Base Connector for SAP R/3 consists of the following parts:

- The Run-time classes for SAP R/3 implementing the Common RFC Interface for Java
- The BOR Access Classes
- The classes for connection pooling

The CCF Connector for SAP R/3 is built on top of the Base Connector for SAP R/3.

RELATED CONCEPTS

Access Builder and Connector for SAP R/3
CCF Connector for SAP R/3
Common RFC Interface for Java
Run-time classes for SAP R/3
BOR Access Classes
Connection pooling

RELATED REFERENCES

Package com.sap.rfc
Package com.sap.rfc.exception
Package com.ibm.sap.bapi
Package com.ibm.sap.bapi.bor
Package com.ibm.sap.bapi.connectionmanager
Package com.ibm.sap.bapi.exception
Package com.ibm.sap.bapi.rfcserver

Common RFC Interface for Java

The Common RFC Interface for Java consists of a set of Java interfaces and classes. These interfaces and classes define a middleware-independant layer for RFC access. The Base Connector for SAP is implementing the Common RFC Interface for Java by providing a middleware based on the Java Native Interface (JNI).

If applications are built on top of the Common RFC Interface for Java they can leverage different SAP R/3 access methods at run time without needing recoding. This approach allows users to leverage different middleware implementations for different deployment scenarios.

A JNI based runtime library is designed for developer clients and server-centric applications that implement the business logic on the web application server and

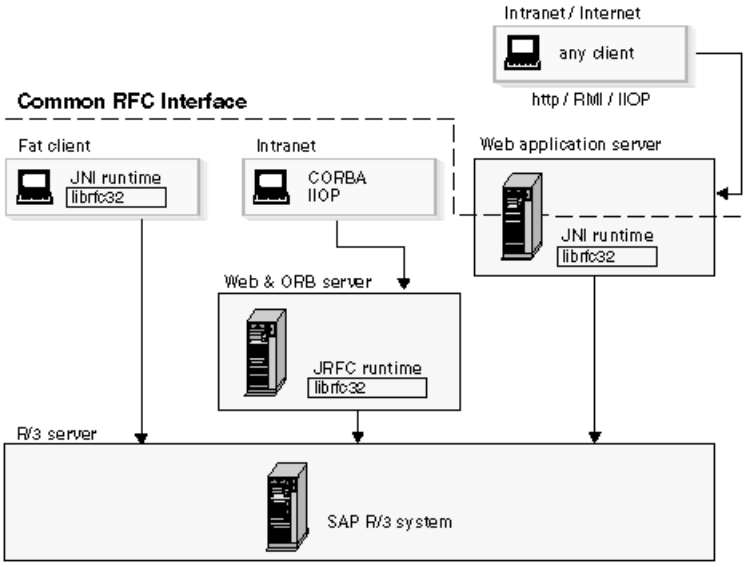
offer client communication using servlets, RMI, IIOP, or other protocols. This runtime library could also be used for client-centric applications on fat clients: either as stand-alone Java application or as installable applet within Netscape.

The SAP R/3 access is performed with a local RFC library (librfc32.dll under Windows).The Java classes access this platform-specific module through Java Native Interface (JNI). RFC library and JNI are available on major platforms.

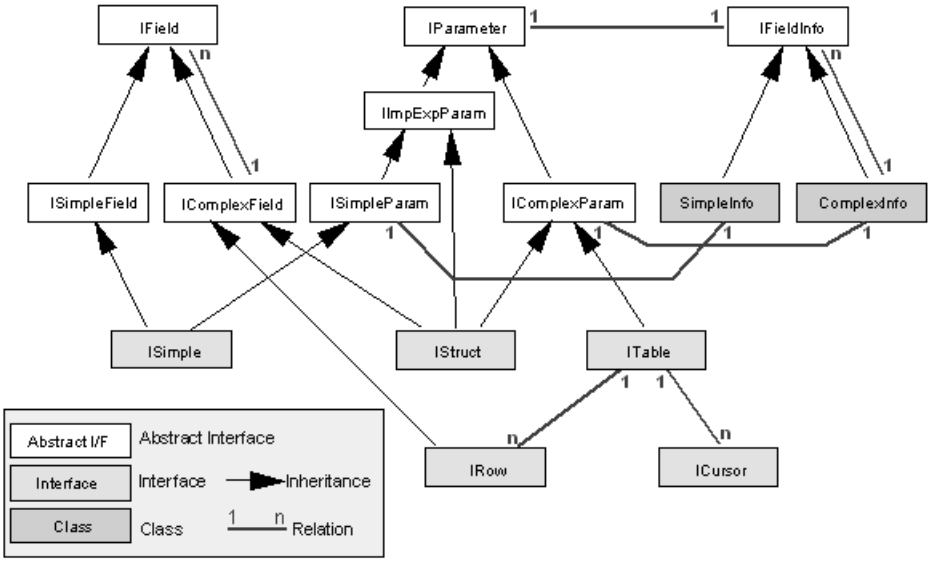
As another middleware implementation, the SAP Java Class library (JRFC) is available as part of the SAP Automation package. This run-time library consists of an ORB server that provides the access to the SAP R/3 system (through the RFC Library) and a pure Java client. This middleware allows a client-centric application purely implemented in Java - both user interface and business logic would reside on the client.

Further Run-time classes for SAP R/3 may be feasible. For example, in an intranet, pure Java clients could directly access the RFC sockets of the SAP R/3 systems.

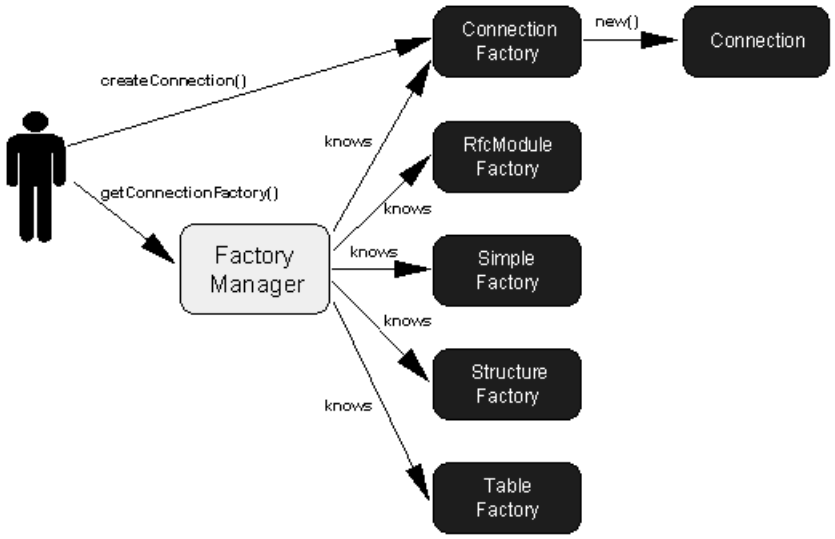
The following figure shows the concept of different access methods and the Common RFC Interface for Java.



The following figure shows the object model of the classes and interfaces of the Common RFC Interface for Java.

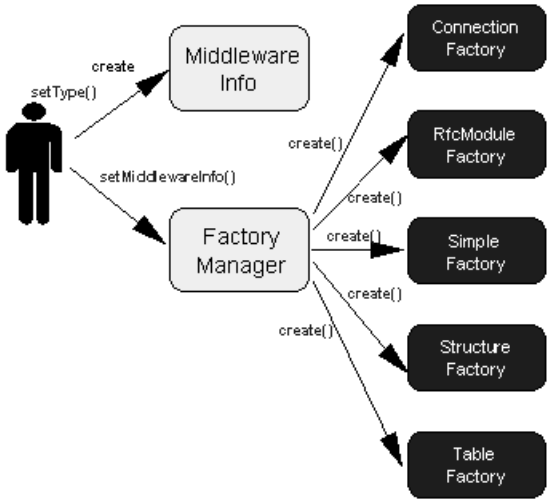


The following pictures show the benefits an application has when being built on top of the Common RFC Interface for Java.
During the start of the application it can choose which middlware to use. This scenario is outlined in the picture below:



First of all an instance of class MiddlewareInfo is created and initialized according to the requirements of the respective middleware. Then the FactoryManager singleton gets informed to use the middleware identified by the MiddleWareInfo object. The FactoryManager then instantiates factory objects for all relevant data types.

During the run of the application it can use the FactoryManager singleton to obtain references to the respective factories to create instances of all relevant types. An example for this is shown in the next picture.



RELATED CONCEPTS

Supported SAP interface technologies
SAP interface technologies
BAPI transactions
Base Connector for SAP R/3

RELATED REFERENCES

Common RFC Interface for Java
ABAP/4 and Java data types
Run-time classes for SAP R/3
Open BAPI Network, Java Section - SAP web site

Run-time classes for SAP R/3

The run-time classes for SAP R/3 are the basis of the run-time access to SAP Business Objects, BAPIs, and RFCs. They are used by all other components of the Access Builder and Connector for SAP R/3 and by the Java proxies generated by the Access Builder. You may also use them directly to develop dynamic access to SAP Business Objects, BAPIs, and RFCs.

Some run-time classes for SAP R/3 help to establish a connection to the R/3 system and some reflect SAP Business Objects and their specific features. The run-time classes for SAP R/3 implement the Common RFC Interface for Java and extend it for use with Business Objects with additional classes like SAPObject.

Classes for modeling Business Objects
A set of Java classes and interfaces represent SAP Business Objects and their features. The object model for these classes and interfaces is briefly described below.

Business Object

SAP Business Objects have a name and may have keys, attributes, methods, and events. Instances of SAP Business Objects are uniquely identified by their keys. Business Objects are represented by the Java class SAPObject.

Key Keys are used to uniquely identify an SAP Business Object. Keys are always simple fields. One Business Object may have several keys. All keys together form the unique identification of an SAPObject. Easy handling of SAP object keys is best supported with Access Builder for SAP R/3 generated BO proxy classes. There the keys are directly visible in the respective BO proxy interface and can be accessed by special key access methods.

Simple field
A simple field holds a single value of a specific ABAP/4 data type. Simple fields are represented by the interface ISimple. It automatically maps the ABAP/4 data types to Java data types. To use an ISimple as a parameter in a call to the SAP R/3 system the ISimple's meta information object (SimpleInfo) must be set correctly. You can figure out how this can be accomplished by having a look at the generated helper classes for structures, tables, or method parameters.

Attribute
Attributes may be simple fields, structures, or Business Objects. Attributes can have read-only or read-write access. A SAP Business Object may have multiple attributes. Currently there is no proper way of using the attributes of an SAP Business Object directly. Instead of this, attributes are accessed using the BAPI methods defined for a specific SAP object type.

Method
A Business Object method is identified by a name. It may have several parameters. Methods are not directly represented in this model. They are invoked by a generic method of SAPObject.

Parameter
A parameter has a name, a type, and a value. A parameter may be input, output, or both. Parameters may be simple fields, structures, tables, or Business Objects. Parameters are represented by the Java interfaces ISimple, IStructure, and ITable.

Structure
A structure has a name and represents a list of fields. Each field may be either a simple field or a structure. Structures are represented by the Java interface IStructure.

Table A table has a name and represents a list of rows. A table has a specific structure which is identical for all rows. Tables are represented by the Java interface ITable.

Row A row represents a list of fields. A field may be either a simple field or a structure. Rows are represented by the Java interface IRow.

Event Events are used in SAP Business Objects only for workflow applications and are not applicable for use together with BAPIs. The Access Builder and Connector for SAP R/3 does not support SAP Business Object events.

Meta Information
At run time, the exact meta information about simple fields and the layout of structures and table rows is needed. The object model of the run-time classes keeps this information in the classes SimpleInfo and ComplexInfo.

Simple field meta information
Simple field meta information includes a name, ABAP/4 data type, length, number of decimals, and an offset if it is part of a structure. Simple field meta information is represented by the Java interface SimpleInfo.

Structure meta information
Structure meta information includes a structure name, length, and an ordered list of field meta information - either for simple fields or for nested structures. Structure meta information is represented by the Java interface ComplexInfo.

Structure's field meta information
The meta information of a structure's field may be either meta information of a nested structure or a simple field. A structure's field meta information is represented by the Java interfaces SimpleInfo or ComplexInfo.

RELATED CONCEPTS

Common RFC Interface for Java
BOR Access Classes
Base Connector for SAP R/3

RELATED REFERENCES

ABAP/4 and Java data types
Common RFC Interface for Java
Run-time classes for SAP R/3

BOR Access Classes

The BOR Access Classes are used to retrieve and hold information from an SAP R/3 system. This information describes the repositories and their contained information about SAP Business Objects, such as attributes, methods, and events, and RFC modules.

The Access Builder uses these classes to give an overview of the SAP R/3 system and to retrieve and generate proxy beans and corresponding documentation. The generated proxy beans do no depend on the BOR Access Classes; they only depend on the run-time classes for SAP R/3. You can also use the BOR Access Classes to retrieve information you are interested in at run time.

The BOR Access Classes consist of the BorInfoMgr, SapObjectInfo, RfcFunctionInfo, ComplexInfo, RfcFunctionDescription, several descriptor classes, and some repository container classes.
The SapObjectInfo and descriptor classes have a similiar interface as the Introspection classes in the Java Development Kit (JDK).

BorInfoMgr
The BorInfoMgr class works as a central instance to retrieve information from the SAP R/3 system. Information needed from the BOR is retrieved by using BorInfoMgr's methods:

getTypeInfo
retrieves all information about an SAP Business Object in an instance of SapObjectInfo

getRfcFunctionInfo
returns all information about a single RFC module or BAPI method

getComplexInfo
returns information about a structure or table in an instance of ComplexInfo

getBorTree
retrieves the full tree of Business Objects from the BOR

getRfcTree
retrieves the full tree of all RFC groups from SAP R/3

getRfcFunctionDescription
retrieves all available documentation of an RFC module from SAP R/3 and returns it in an instance of RfcFunctionDescription.

Descriptor Classes
A specific descriptor class exists for every feature of an SAP business object. An instance of the appropriate class is created for each single feature. It contains the feature's name and advanced information about the internal structure in the SAP R/3 system.

BorDescriptor
The interface implemented by all feature descriptor classes holding BOR meta information.

FeatureDescriptor
The base class for all descriptors used to describe Business Object meta information. It contains methods common for all descriptor classes. It implements the BorDescriptor interface.

AttributeDescriptor
Contains information about an attribute of a Business Object. Subclass of FeatureDescriptor

EventDescriptor
EventDescriptor is currently not used. It is reserved for later use and will contain information about an event of a Business Object. Subclass of FeatureDescriptor

KeyDescriptor
Contains information about a single key of a Business Object. Subclass of FeatureDescriptor

MethodDescriptor
Describes a method of the Business Object which the user can call. Subclass of FeatureDescriptor

SapObjectInfo
An instance of the SapObjectInfo class keeps all information about a specific Business Object in contained instances of the respective descriptor classes. Subclass of FeatureDescriptor

FieldDescriptor
Contains the description of a simple or a structured field. Implements the BorDescriptor interface.

SimpleDescriptor
Contains the description of a simple field. Contains similar data as the class SimpleInfo of the SAP R/3 run-time Classes. Subclass of FieldDescriptor.

ComplexDescriptor
Contains the description of a complex field (structure or table). Contains similar data as the class ComplexInfo of the SAP R/3 run-time Classes. Subclass of FieldDescriptor.

ParameterDescriptor
Describes a parameter used in the parameter list of a Business Object's method or of an RFC function module.

RfcFunctionInfo
An instance of the RfcFunctionInfo class keeps all information about a specific RFC function module.

RfcFunctionDescription
An instance of the RfcFunctionDescription class keeps all documentation about a specific RFC function module contained in SAP R/3.

Container classes of the Business Object Repository

BorTree
An instance of the BorTree class contains the complete hierarchy of business objects stored in the Business Object Repository (BOR). The Business Object hierarchy contains a large number of elements where each element represents any kind of object. These objects may be business domains and sub-domains, informational objects, and Business Objects, too.

The number of elements contained in the Business Object Repository depends on the SAP R/3 system.

BorTreeObject
An instance of the BorTreeObject class contains the raw SAP R/3 data of a single element in the Business Object Repository.

ArrayElement
An instance of the ArrayElement class contains one BorTreeObject instance and some additional information about the organizational context in which this BorTreeObject instance resides inside the Business Object Repository.

The organizational context contains the locations of the parent, next, and previous siblings and the first child as well as the total number of children.

Container classes of the RFC module Repository

RfcTree
An instance of the RfcTree class contains the complete organization of an RFC module repository.

The RFC module repository consists of exactly two levels of information: The logical groups of RFC modules and the corresponding RFC modules themselves.

All RFC modules are alphabetically sorted for quick access and all group items contain only references to those RFC modules belonging to that group.

The number of RFC modules and their logical groups depend on the SAP R/3 system.

RfcElement
An instance of the RfcElement class contains information describing a single RFC module.

RfcGroup
An instance of the RfcGroup class contains information describing a single logical RFC group and a list of references to the RFC modules belonging to that group.

RELATED CONCEPTS

Run-time classes for SAP R/3
Common RFC Interface for Java
Base Connector for SAP R/3

RELATED TASKS

Invoking BAPIs and RFCs dynamically

RELATED REFERENCES

ABAP/4 and Java data types
BOR Access Classes
Run-time classes for SAP R/3

Connection pooling

Please note:
When using the CCF Connector for SAP R/3 no measures are required in regards of obtaining connections and connection pooling. This is done automatically by the CCF Connector for SAP R/3 implementation which itself takes advantage of the CCF Connection Manager either provided by the CCF Infrastructure implementation of a CCF compliant application server (e.g. WebSphere Application Server Enterprise Edition) or part of a manually setup CCF RuntimeContext. Please have a look at the respective chapters of the Common Connector Framework documentation for further details.

When looking at a typical application which accesses an SAP R/3 system, the following pattern can be noticed:

1. The application opens a connection to an SAP R/3 system
2. The application manipulates/retrieves data in/from the SAP R/3 system. (This step can be iterated.)
3. The application closes the SAP R/3 connection (usually before the application terminates).

As step two is typically triggered by user interaction, e.g. by clicking a push button, the period of time between different iterations of step two is significant. So in this scenario SAP R/3 connections lie fallow most of the time. In addition opening and closing connections to SAP R/3 systems are resource and especially time consuming operations. The Base Connector for SAP R/3 therefore provides a connection manager which is capable of pooling connections to address these issues and deal with SAP R/3 connections in an efficient way. The idea behind connection pooling is to share connections between applications. A typical application taking advantage of the provided connection manager would match the following pattern (please compare with the first pattern above):

1. (This step is left out)
2. (This step can be iterated)
   a. Reserve a suitable connection (by specifying ConnectInfo and UserInfo) from the connection manager
   b. Manipulate/retrieve data in/from the specified SAP R/3 system.
   c. Release the connection for further reuse by other/the same applications.
3. (This step is left out)

Serving the reserve request the connection manager first looks if a suitable unused connection exists in the pool. If the search is successful the found connection will

be returned. Otherwise a new connection to the specified SAP R/3 system will be established and passed back to the requestor. The application can use the returned connection to access and manipulate data in the SAP R/3 system, e.g. call some RFC modules. As soon as the interaction with the SAP R/3 system is done the connection should be released for further reuse. The release method of the connection manager resets the connection before it is put back into the pool. BAPI transactions therefore have to be commited before releasing a connection. Otherwise they will be rolled back.

The Connection Manager API
The connection pooling facility can be accessed through the ConnectionManager class in the com.ibm.sap.bapi.connectionmanager package. The following methods are all part of this class.

There are two alternative methods to obtain a connection from the pool: reserve(ConnectInfo, UserInfo) and reserveWait(ConnectInfo, UserInfo). Both methods take the usual SAP logon information as arguments and return an appropriate IRfcConnection. The returned connection can be used to access the SAP R/3 system and invoke the desired BAPIs and RFCs. As soon as the connection is no more needed it should be put back into the pool for reusage by simply invoking the release(IRfcConnection) method of the connection manager.

There are also some properties with appropriate getter/setter methods to configure the connection manager:

- maxConnections
- maxUnusedTime
- waitTimeWhenNoConnectionAvailable

The maxConnections property limits the pool size. If the current number of connections equals maxConnections, an invokation of reserve() would raise a NoConnectionAvailableException. This is different for reserveWait(...) which will wait until there is a connection available. The method checks periodically (every waitTimeWhenNoConnectionAvailable milliseconds) if there's an unused connection in the pool. Please note that this approach does not guarantee that connections are returned in a first-in-first-out way.

Before reserve(...) and reserveWait(...) return a connection they check the period of time since the connection was last used. If this value exceeds maxUnusedTime in ms, the validity of the connection is checked by issuing an RFC_PING call. Otherwise the connection is supposed to be valid.

The following code snippet illustrates the use of the connection manager:

```
import com.sap.rfc.*;
import com.sap.rfc.exception.*;
import com.ibm.sap.bapi.connectionmanager.*;
...
ConnectInfo ci=new ConnectInfo();
ci.setHostName("yourR3system");
ci.setSystemNo(0);
UserInfo ui=new UserInfo("user", "password", "800", "e");
IRfcConnection connection = null;
ConnectionManager connectionManager =
          ConnectionManager.getSingleInstance();
try
{
   connection = connectionManager.reserveWait(ci,ui);
   ...
   CompanyCode.getlist(connection, ...);
```

```
   ...
   connectionManager.release(connection);
}
catch (JRfcBaseException exc)
{
   // exception handling
}
```

RELATED CONCEPTS

Base Connector for SAP R/3
CCF Connector for SAP R/3
Common Connector Framework

RELATED TASKS

Connecting to the SAP R/3 system

RELATED REFERENCES

Package com.ibm.sap.bapi.connectionmanager

Chapter 5. CCF Connector for SAP R/3

By introducing the Common Connector Framework (CCF), IBM addressed the issues raised by the integration of existing Enterprise Information Systems (EIS). In the past, every EIS provider came up with its own proprietary solution, a so-called connector, to ensure local and/or remote access to the EIS in order to serve the markets demand for openness. With the today's need for system integration in mind, this approach has several drawbacks, e.g. although the connectors for different EIS's are semantically very similar, they all look different. As a result, the lerning curve of a developer working with a new connector is quite high, there are different high level tools for different connectors, etc.
As a solution for these drawbacks, the Common Connector Framework provides a common client programming model and a common infrastructure programming model for connectors. The common client programming model reduces the lerning curve for switching from one connector to another drastically. It also has the function as the plug-in interface for higher level tools, making those independent of a particular connector. The common infrastructure programming model defines the contract between a CCF compliant component server and a CCF connector. It enables the component server to control the state of the connector, whereas the connector can leverage services provided by the component server (such as security, transactions, connection pooling) in an easy way.

In order to fit into the CCF, Connector for SAP R/3 contains a CCF compliant connector implementation which is built upon the Base Connector for SAP R/3. Currently there are three ways to involve the CCF Connector for SAP R/3:

- Using Enterprise Access Builder (EAB) Commands and Navigators
- Using enterprise bean proxies for SAP business objects generated by Access Builder for SAP R/3
- Programming directly against the CCF client interfaces

Using Enterprise Access Builder (EAB) Commands and Navigators
The Enterprise Access Builder for Transactions provides tools to model and implement interactions with Enterprise Information Systems. Since accessing these systems is done via the CCF client interfaces of the respective CCF connector, the tools are independent of the specific EIS. As a result a developer can use the same tools to model interactions with e.g. CICS and SAP systems. While EAB Commands encapsulate single interactions with an EIS, Navigators are used to model complex interaction flows. Please see the EAB documentation for details. Modelling interactions with SAP R/3 systems typically involves the following steps:

1. Identify the required SAP R/3 data and/or business logic.
2. Identify the appropriate RFC modules to access the data/business logic.
3. Generate proxy beans for the required RFC modules with Access Builder for SAP R/3.
4. Create and edit EAB Commands using the generated RFC proxies as input/output beans.
5. Optionally compose Navigators using Commands and/or Navigators created in steps 4 and 5.
6. Use the Commands/Navigators in servlets, enterprise beans, Procedural Adapter Objects and applications. Please see the EAB documentation for a detailed description of the different scenarios.

The following matters have to be considered when using the EAB tools in connection with the CCF Connector for SAP R/3:

- The CCF Connector for SAP R/3 specific ConnectionSpec and InteractionSpec classes, i.e. com.ibm.connector.sap.SAPConnectionSpec and com.ibm.connector.sap.SAPInteractionSpec, must be used.
- Input and output bean must be proxy beans for RFC modules generated by Access Builder for SAP R/3
- Input and output bean must be of the same type.
- As the generated RFC proxies contain all required information, no properties have to be set in the SAPInteractionSpec.

Using enterprise bean proxies for SAP business objects generated by Access Builder for SAP R/3

Access Builder for SAP R/3 allows you to directly generate enterprise bean proxies for SAP Business Objects. The generated stateful Session EJB beans can be deployed on any enterprise beans server (EJS) providing support for stateful Session EJB beans and an implementation of the CCF infrastructure interfaces. Additionally these generated enterprise beans provide the client with a smart table cache to avoid long download times for large tables. If it is sufficient to do a one to one mapping of SAP Business Objects to EJB beans then this is the easiest approach. In all other cases you should use the EAB tools.

Each Business Object EJB bean must be bound to a specific SAP R/3 system at deployment time. The deployer has to set the necessary connection properties through the EJB bean's deployment descriptor. The deployer is not required to set all properties, e.g. gatewayHost and gatewayService don't have to be set if no gateway is used, or hostName and systemNo is not needed if loadBalancing is activated). If one EJB bean will be used with different SAP R/3 systems, it must be deployed once for every system using the appropriate connection properties. The different EJB beans must be addressed with their individual names.

When accessing SAP R/3 systems an authorization must be performed. Running on a full CCF compliant component server, the CCF Connector for SAP R/3 can retrieve the logon information from the security service of the server. But CCF Connector for SAP R/3 also provides two alternative circumventions for environments which are missing this support.

- A special EJB bean (com.ibm.sap.bapi.ejb.EjbUserManager) is provided to manage the associations between the client instances and the SAP R/3 logon parameters. The EjbUserManager class contains methods for setting the SAP R/3 logon information. Client applications have to instantiate the EjbUserManager on every EJB server. The methods setUserInfo(userID, password, client, language) or setUserInfo(userID, password, client, language, codepage) associate the client instance with the SAP R/3 user information for all EJB beans that are instantiated afterwards on the EJB server. Therefore, one of these methods must be called before any other SAP Business Object EJB bean is instantiated. The SAP R/3 user information may be changed later during runtime. This change does not affect the previously instantiated EJB beans, it only applies to all subsequent EJB bean instantiations of the client.
- An alternative approach is to specify the SAP R/3 user information during instantiation of the enterprise bean proxy. This can be done by using the create(com.sap.rfc.UserInfo) instead of the create() method in the home interface of the Business Object enterprise bean proxy. In this case the EjbUserManager doesn't have to be deployed.

Programming directly against the CCF client interfaces
Please see the Common Connector Framework documentation for a detailed description of this approach. Even though it is possible to script directly against the CCF Connector for SAP R/3, this is not recommended. Using EAB Commands and Navigators will relieve a developer from the burden of scripting the same sequences over and over again.

CCF infrastructure interfaces
The CCF Connector for SAP R/3 leverages services (such as security, transactions, connection pooling) through the CCF infrastructure interfaces. CCF compliant component servers have implemented these interfaces and adapt the CCF Connectors view to their actual service implementation. When running on a component server providing none or no full support of the CCF infrastrucure services, the required services have to be adapted by setting up the CCF infrastructure manually. The same applies in case of developing a fat (standalone) Java application. Please see the Access Builder and Connector for SAP R/3 Release Notes for the current status of available CCF infrastructure implementations (e.g. WebSphere Application Server, VisualAge for Java Enterprise Edition).

RELATED CONCEPTS

Common Connector Framework
Enterprise Access Builder for Transactions
Base Connector for SAP R/3
Proxy beans for RFC modules
Enterprise bean proxies for Business Objects

RELATED TASKS

Setting up the CCF infrastructure
Building EAB Commands using proxy beans for RFC modules
Using enterprise bean proxies for SAP Business Objecs
Generating proxy beans for SAP Business Objects and RFC modules

RELATED REFERENCES

Package com.ibm.connector.sap
BO and RFC proxy beans

Chapter 6. RFC Server for SAP R/3

The Access Builder and Connector for SAP R/3 provides a framework for building RFC Server applications in Java. The RFC Server for SAP R/3 allows to implement function modules in Java that can be called by ABAP/4 programs (or any other RFC Client). The RFC Server for SAP R/3 supports standard calls as well as transactional remote function calls. Furthermore, it offers interfaces to do security checking and to customize the process of instantiating the function modules. The RFC Server for SAP R/3 is designed for a multithreaded usage and therefore can process multiple remote function calls simultaneously. In addition, it is possible to register IdleEventListeners at the RFC Server, so that the running application can be notified with IdleEvents. Therefore the application is able to do other tasks in case of the RFC Server being idle.

Note: Only one RFC Server can be started per application process.

Architecture
Basically, the RFC Server for SAP R/3 is built upon the two interfaces IRfcServer and IRfcServerThread. They are implemented in the classes RfcServer and RfcServerThread.

The IRfcServer instance acts as the central parent object and provides methods for managing the offered function modules and for configuring, starting, and stopping the RFC Server. When it is started the IRfcServer registers itself at the configured SAP gateway and starts the desired number of threads in form of IRfcServerThread instances. Each of them is capable of processing a separate RFC request.

Configuring and controlling the RFC-Server for SAP R/3
Before you can start an RFC Server you have to configure it by carrying out several tasks: First of all, you have to set an RfcServerInfo instance. It defines the gateway at which the server should register and the program ID with which it should register. Then you can set an optional transaction manager or security manager or both. Finally, you have to add all the server function modules that the RFC Server should offer to the RFC Clients to the IRfcServer instance.

If you have completed all of these tasks you can start the IRfcServer using the start method specifying the number of threads that should be able to process incoming requests. The IRfcServer will return from the start method before any of the IRfcServerThreads is started. A special starter thread will perform this task. The IRfcServer will indicate its state by firing PropertyChangeEvents on its state property. When it is started the state property will change from RfcServerConstants.STATE_STOPPED to RfcServerConstants.STATE_STARTING.

If all threads have been started the state property will change to RfcServerConstants.STATE_RUNNING. Listeners can be added and removed all the time, no matter whether the server is running or not.

You can stop the RFC Server using the stop method which takes the maximum shutdown time in milliseconds as an argument. The IRfcServer will immediately return from the stop method and change its state to RfcServerConstants.STATE_STOPPING. A special stopper thread will stop each of the related IRfcServerThreads. If it is within the given maximum shutdown time, each thread will try to end its current processing of an incoming remote function

call before it shuts down. Otherwise the thread is interrupted and stopped when the maximum shutdown time has been reached - regardless of what it is currently doing.

If you definitely would like to stop all of the threads properly, then you have to specify a maximum shutdown time that is larger than 5000 milliseconds plus the largest processing time of all server function modules. You can also specify a maximum shutdown time of RfcServerConstants.TIME_INFINITE to make sure that no thread will be interrupted. After all the threads have shutdown, the IRfcServer's state will change to RfcServerConstants.STATE_STOPPED.

Creating, managing, and executing function modules
The function modules that the RFC Server should offer have to implement the IRfcServerModule interface and must provide a meta information object as an instance of IRfcServerModuleInfo. The IRfcServerModuleInfo instance must contain the name and a textual description of the server function module and meta information about all of its import, export, and table parameters.

The IRfcServerModule contains an execute method that takes all import and table parameters and in return exports the export and table parameters. You have to implement your business logic of the function module in this method. It is essential that the name of the server function module is unique. The RFC Server will not accept different server function modules with the same name, since it would not be clear which of them to use in case of being called. In addition the name must not be longer than 31 characters.

In order to offer your remote function modules through your RFC Server you have to add them to the IRfcServer instance. You can do this by calling one of the four addRfcServerModule methods. If there should be only one instance of your IRfcServerModule object you may use the add methods that take an instance of your module. In this case it is important to know that every call to this function module leads to the same IRfcServerModule instance. If your RFC Server has multiple threads listening for RFC requests it is possible that the respective execute method is called simultaneously within different threads. Therefore, you have to take care for thread synchronization within your execute method if necessary. If you want to avoid that or if you want to instantiate your remote function modules only when they are needed you can also provide an IRfcServerModuleFactory object that instantiates the IRfcServerModules. Its method createRfcServerModule does that and will be called when the corresponding function module is requested. Then the execute method of the returned IRfcServerModule will be processed. The object will be released after the call.

This behaviour assures that it is always a different instance of the IRfcServerModule that is called. The advantages of the factory concept are that you do not have to take care of thread synchronization problems within your execute methods and that you can save memory in large RFC Server environments where you do not want to instantiate all available function modules in advance. To use this technique you have to register the function modules at the IRfcServer instance with the addRfcServerModule methods that take an IRfcServerModuleInfo and an IRfcServerModuleFactory object as arguments. The additional argument IRfcServerModuleFactoryParam provides the flexibility to freely customize the instantiation process itself. The IRfcServerModuleFactoryParam instance will be passed to the createRfcServerModule method in the factory object. You are free to use one factory instance for all IRfcServerModules or an own factory instance for each of them.

You can mix both techniques of adding IRfcServerModule objects to the RFC Server within one server. But you can only add new function modules if the server is not running.

Transaction management
The RFC Server for SAP R/3 can optionally accept and process transactional remote function calls. A transaction invoked by an RFC Client (ABAP/4 program) usually consists of several remote function calls that are all running with the same transaction ID. You have to make sure that every call within a transaction is executed only once and can be rolled back if requested. Since such a management depends on your business logic provided by your IRfcServerModules you have to implement the transaction manager by yourself. For this you have to implement the interface IRfcServerTransactionManager. It defines the following methods supporting a one-phase-commit transaction management:

- checkTransactionId(String transactionId)
- beforeExecution(String transactionId, int transactionSequenceNo, IRfcServerModule rfcServerModule)
- commit(String transactionId)
- rollback(String transactionId)
- confirmTransactionId(String transactionId)

Because there is only one central instance of the transaction manager for every IRfcServer object you have to consider that its methods can be called by multiple threads simultaneously and must be implemented thread safe. The IRfcServerTransactionManager instance can only be set or exchanged, if the server is not running.

Security management
The RFC Server for SAP R/3 is capable of restricting access to the offered function modules in two ways. The first method is to check if the calling user or the calling system has the authorization to execute the specified function module before the appropriate execute method is entered. Therefore you have to set an IRfcServerSecurityManager instance for the IRfcServer object. The IRfcServerSecurityManager interface only contains the method checkAuthorization which you have to implement.

The method's arguments are an RfcAttributeInfo object and an IRfcServerModuleInfo instance specifying the remote function module that is requested for execution. It should return a boolean value indicating the permission or prohibition for calling the desired function module. The RfcAttributeInfo object carries all information about the current (thread related) RFC connection including the name of the calling user and the calling system. With this information it should be possible to do the authorization check. If its result is positive, the next action that will take place is the creation of the corresponding IRfcServerModule instance if necessary and then the call to its execute method. Otherwise the exception 'ACCESS_DENIED' will be thrown to the calling RFC Client (ABAP/4 program).

If you wish to exclude some of the function modules from the authorization checks (because they are for public availabilty for example), you can use the addRfcServerModule methods with the checkAuthorization flag set to false, when adding the function modules to the IRfcServer object. This flag has no effect, if you do not set the IRfcServerSecurityManager instance for the IRfcServer object. The IRfcServerSecurityManager instance can only be set or exchanged if the server is not running.

Alternatively or additionally to the above security checks can be performed within the execute method of the function modules themselves. There you have the possibility to use the given parameter values for the authorization check, too. You can get the corresponding RfcAttributeInfo object by performing a cast of the current thread to an IRfcServerThread instance. Then you can use the getRfcAttributeInfo method of this instance to receive the RfcAttributeInfo object. If the following authorization check would then be negative you can throw a JRfcRfcAbapException or you can selectively fill in or omit some sensitive return values.

Idle events
Idle events are fired by the IRfcServer instance when all its related IRfcServerThread instances have been waiting for a remote function call request but none of them received one for a specified period of time, the so called IdleEventTime. The IRfcServer instance keeps firing idle events every IdleEventTime period until the next remote function call request is received by an IRfcServerThread instance. All idle events are fired within a separate idle event processor thread so that none of the IRfcServerThread instances is blocked. They are all still able to process incoming RFC requests all the time.

The adjustable IdleEventTime property enables you to find the right balance between the processing of idle events and using up system performance. If there were too many idle events fired, it is possible that so much system performance is used up by this task that other threads or processes running simultaneously could be impeded. The IdleEventTime is given in milliseconds and can be adjusted using the IRfcServer's setIdleEventTime method. If you specify a value of RfcServerConstants.TIME_INFINITE there will never be an IdleEvent fired. This is also the default behaviour.

To receive IdleEvents you furthermore have to implement the IdleEventListener interface in one of your objects and register it at the IRfcServer instance using the addIdleEventListener method. When an idle event occurs, the method onIdle(IdleEvent event) of all registered IdleEventListener objects will be called. The given IdleEvent object contains the IRfcServer instance that fired it as the source of the event.

RELATED CONCEPTS

Supported SAP interface technologies

RELATED TASKS

Developing an RFC Server application
Configuring SAP R/3 to call an RFC Server application from ABAP/4

Chapter 7. Utilities

Logon bean

A Java application using SAP R/3 functionality must connect to an SAP R/3 system before certain actions can be processed. For that reason you often want the user to log on to the SAP R/3 system.

The Logon bean provides an SAP logon panel where the user can provide logon information. You can use it in any visual development environment that supports Java beans. In VisualAge for Java, you can simply drop it onto the surface of the Visual Composition Editor, connect it to a Logon model, and integrate it into your application.

The new Logon bean provides extended capabilities. The user interface is completely based on the JFC library. The distinction between model and view of a Logon bean allows you to create your own implementation of logon classes using a standard view. A default implementation is provided which adapts the functionality of the former Logon view, which is now deprecated.

The Logon bean uses a design pattern called Model-View-Controller (MVC) often used in building user interfaces. In an MVC UI there are three communicating objects: model, view, and controller. The model is the underlying logical representation, the view is the visual representation, and the controller specifies how to handle user input. When a model changes it notifies all views that depend on it. The controller determines what action to take when receiving keyboard input.

The Logon bean uses a common variation of MVC in which view and controller are combined in one object. This class provides the following functionality:

- The **controller** controls user input (for example disables/enables entry fields during logon) and provides the model with data entered by the user. It also maps UI controls (for example buttons) to specific methods (for example logon()) provided by the model.
- The **view** displays the data the model provides. Every time logon data changes the model notfies the view. After the notification the view obtains the new data and updates the UI in order to display the new values.
- The Logon bean communicates with the model through the `LogonModel` interface. From there it can be used with different logon model implementations.

The `LogonModel` interface defines the communication between the model and the view/controller. It provides methods to get and set the logon data which is necessary to connect to an SAP R/3 system. Every time logon data changes the logon model notifies all registered listeners. It also defines the logon() and logoff() methods and the corresponding events.

The abstract class `AbstractLogonModel` provides a base implementation for the `LogonModel` interface. Standard behaviors like get and set methods for `UserInfo`, `ConnectInfo`, and their properties are defined here. It also provides default implementations of events defined in `LogonModel`. You can extend the `AbstractLogonModel` base class by your own implementation.

Providing the user with a list of SAP R/3 systems
Usually, a standard SAP Logon panel provides a list of SAP R/3 systems to which users can log on. Instead of manually typing information like host name, gateway service, system number etc., users select an entry in the list and double click it. After that the SAP GUI is shown where users enter their personal data (username, password etc.) and log on to the SAP R/3 system. To provide the list of available SAP R/3 systems the logon panel uses data from a file called **saplogon.ini**.

The **Logon bean** shipped with Access Builder for SAP R/3 provides functionality similar to the standard SAP logon panel. The bean reads a logon ini file and displays its contents as a list. In contrast to the standard SAP Logon panel, the Logon bean cannot add or modify entries in the ini file.

Users select one entry in the list to specify the information which the `ConnectInfo` object needs. They have to enter the information stored in the `UserInfo` object (except the `Codepage` property) manually. After that they can establish an RFC connection to the SAP R/3 system.

Structure of the logon ini file
The file saplogon.ini has the following structure:

```
[SectionName1]
item1=value1
item2=value2
itemn=valuen
[SectionName2]
item1=value1
item2=value2
itemn=valuen
...
```

The following table shows all sections which the Logon bean recognizes (all other sections are ignored).

Section	Content	Usage
[Router]	Router string	R/3 system: • sever selection: router string and host name of SAP R/3 application server are concatenated • load balancing: router string and and message server host name are concatenated R/2 system: • router string and gateway host name are concatenated
[Server]	R/3 logon • server selection: host name of SAP R/3 application server (server selection, for example pswdf028) • group selection: name of group (for example PUBLIC) R/2 logon • gateway host	R/3 system: • `ConnectInfo.setHostName()` • `ConnectInfo.setGroupName()` R/2 system: • `ConnectInfo.setGatewayHost()`

[Database]	System number	`ConnectInfo.setSystemNo()`
[Description]	Description of SAP R/3 System. for example DIP (pswdf028)	description of the system
[MSSysName]	System Name, for example DIP	`ConnectInfo.setSystemName()`
[MSSrvName]	Message Server Host name, for example pswdf028	`ConnectInfo.setMsgServer()`
[Codepage]	Codepage, for example 1100	`see [CodepageIndex]`
[CodepageIndex]	**-1** use default codepage **!= -1** use value specified in section [Codepage]	`UserInfo.setCodepage()`
[Origin]	**MS_SEL_GROUPS** group selection, load balancing on **MS_SEL_SERVER** server selection, load balancing off **USEREDIT** server selection, load balancing off	`ConnectInfo.setLoadBalancing()`
[System]	**2** logon to SAP R/2 **3** logon to SAP R/3	`ConnectInfo.setRfcMode()`

Setting the ini file for the Logon bean

There are several ways to provide the Logon bean with information about the file name and location of the logon ini file:

Who	When	How to
application developer	at design time	by setting the property `defaultFileNameSapLogonIni`
application developer	at run time	by using the method Logon.`setFileNameSapLogonIni()`
user	at run time	• by providing the logon ini file in the current directory. The file name should be **saplogon.ini** • by providing a file called **saplg.ini** in the current directory. This file should contain a single line of text specifying the path and file name of the ini file. For example http://p28095/saplogon1.ini

To access the ini file you have to specify a valid file name (path + file name, for example c:\\winnt\\saplogon.ini) or a valid URL (for example http://myhost//saplogon.ini)

Reading the ini file during startup of the Logon bean
There is a defined order in which the Logon bean reads the various logon ini files during startup. If one step is successful then the following steps are ignored:

- Read **saplogon.ini** in the current directory
- Read **saplg.ini** in the current directory. Read logon ini file according to settings within saplg.ini.
- Read logon ini file according to settings during design time (`defaultFileNameSapLogonIni` property)

After the initial startup sequence you can specify the logon ini file by using the `FileNameSapLogonIni` property.

Change password Feature
The logon bean provides functionality to change the user s password on the SAP R/3 system. You can only use this functionality if you use the new extended logon-model **`ExtendedLogonModel`**.

There are three properties which control the change-password feature:

Name of property provided by the logon bean	Function
`EnablePasswordChange`	• `true`(default value): the button on the logon bean which allows the user to change the password is visible • `false`: the button on the logon bean which allows the user to change the password is not visible
`CheckPasswordExpiration`	• `true`: While the user is logging on (after pressing the logon button) the logon bean checks if the password has expired. If this is the case, then the change-password panel pops up. This panel gives the user the possibility to change the password • `false`(default value): The logon bean does not check if the password has expired **Note:** this property works even if the `EnablePasswordChange` property is set to `false`

Name of property provided by the logon bean	Function
`EnforcePasswordChange`	Has an effect when the `CheckPasswordExpiration` is set to `true` and the user is logging on after pressing the logon button • `true`: In case that the password has expired, the user is forced to change the password before he/she can continue. If the user cancels changing the password, or something goes wrong while changing the password, (e.g. using the old password as the new password) then the attempt to log on is cancelled and then the user has to try it again. • `false`(default value): The user can cancel changing the password **Note:** this property works even if the `EnablePasswordChange` property is set to `false`

To provide the change-password functionality the current version of the ExtendedLogonModel uses the following function modules in SAP R/3:

- `SUSR_LOGIN_CHECK_RFC`: to check if the password has expired
- `SUSR_USER_CHANGE_PASSWORD_RFC`: to change the password

This function modules are usually available on SAP R/3 releases 4.5b or higher. Before the bean makes an attempt to change the password, it check if these function modules are available on the SAP R/3 system. If this is not the case, then changing the password is not possible.

Additional Features:
Logon bean:

- `isServerListEnabled` returns `true`, if the server list panel is enabled, otherwise the function returns `false`
- `setServerListEnabled` enables/disables the server list panel

ExtendedLogonModel:

- the `LogonLanguage` property returns the string value of the language associated with the current connection

RELATED CONCEPTS

saplogon.ini

RELATED TASKS

Integrating the Logon bean into your application

RELATED REFERENCES

Logon bean

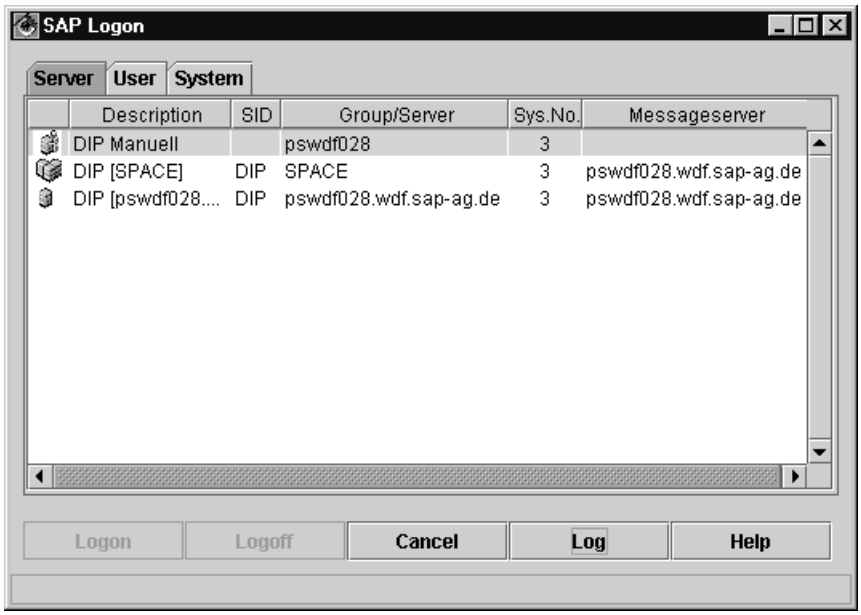
saplogon.ini

The following sample saplogon.ini file shows how the Logon bean interprets the different sections. The Logon bean only reads the highlighted sections.

```
[Configuration]
SessManNewKey=5
[MSLast]
MSLast=MLP
[MSWinPos]
NormX=683
NormY=450
[Router]
Item1=
Item2=/H/router/H/
Item3=
[Router2]
Item1=
Item2=
Item3=
[RouterChoice]
Item1=0
Item2=0
Item3=0
[Server]
Item1=pswdf028.wdf.sap-ag.de
Item2=SPACE
Item3=pswdf028
[Database]
Item1=28
Item2=28
Item3=28
[System]
Item1=3
Item2=3
Item3=3
[Description]
Item1=DIP [pswdf028.wdf.sap-ag.de]
Item2=DIP [SPACE]
Item3=DIP Manuell
[Address]
Item1=155.56.158.26
Item2=155.56.158.26
Item3=
[MSSysName]
Item1=DIP
Item2=DIP
Item3=
[MSSrvName]
Item1=pswdf028.wdf.sap-ag.de
Item2=pswdf028.wdf.sap-ag.de
Item3=
[MSSrvPort]
Item1=sapmsDIP
Item2=sapmsDIP
Item3=
[SessManKey]
Item1=2
Item2=1
Item3=3
[SncName]
Item1=
Item2=
Item3=
[SncChoice]
Item1=0
Item2=-1
```

```
Item3=-1
[Codepage]
Item1=1614
Item2=1100
Item3=1100
[CodepageIndex]
Item1=26
Item2=-1
Item3=-1
[Origin]
Item1=MS_SEL_SERVER
Item2=MS_SEL_GROUPS
Item3=USEREDIT
```

The following figure shows how the Logon bean displays this sample saplogon.ini information:



RELATED CONCEPTS

Logon bean

RELATED TASKS

Integrating the Logon bean into your application

RELATED REFERENCES

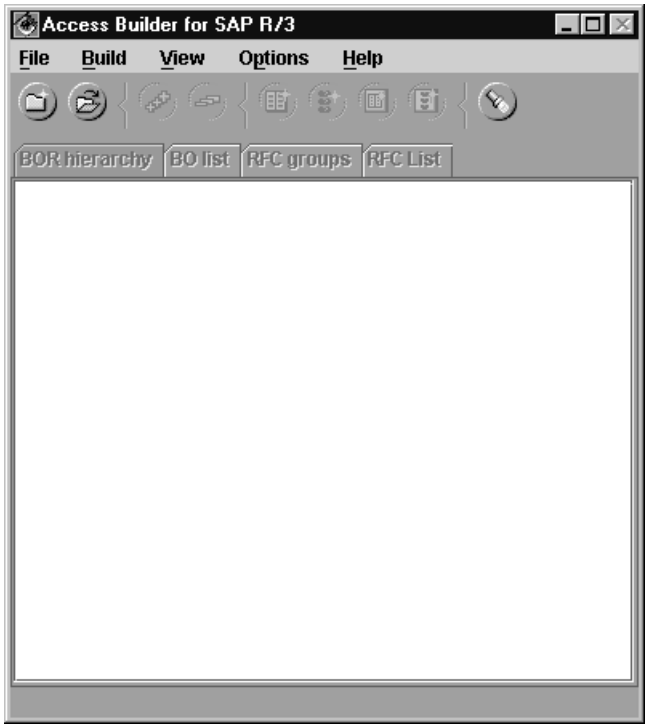
Logon bean

Chapter 8. Using the Access Builder

Starting the Access Builder for SAP R/3

You can start the Access Builder for SAP R/3 window with the following steps:

1. Select the package that will contain the generated proxy beans
2. Select the **Selected** menu
3. Select the **Tools** submenu
4. Select **Access Builder for SAP R/3**



The Access Builder main window appears. You can select the **File > Open** menu item to open a locally stored repository or the **File > New** menu item to create a new local copy of a repository in your SAP R/3 system.

RELATED CONCEPTS

Access Builder for SAP R/3
Local repositories

RELATED TASKS

Starting the Access Builder for SAP R/3 outside of VisualAge for Java
Customizing the Access Builder for SAP R/3
Navigating through a repository
Creating a local repository
Generating proxy beans for SAP Business Objects and RFC modules

Starting the Access Builder for SAP R/3 outside of VisualAge for Java

If you prefer, you can also invoke the Access Builder for SAP R/3 outside of VisualAge for Java. While you may gain some performance increase, you loose the tight integration into VisualAge for Java.
To do this, you need to install JDK 1.2 and the environment variable JDKHOME must point to the base directory of the JDK installation.

Perform the following steps:

1. Open a command line window
2. Change to the directory where VisualAge for Java is installed
3. Enter the subdirectory ide\tools\com-ibm-sap-bapi
4. Invoke the command file AccessBuilder.cmd

Now the Access Builder for SAP R/3 main window appears in the same way as if you started the Access Builder from within VisualAge for Java.

The following is different between external and internal invocation of the Access Builder:

- Some performance increases
- Generated proxy beans are not imported into VisualAge for Java automatically. You have to import the beans manually.
- Generated EJB beans are not deployed in the EJB development environment of VisualAge for Java
- Online help documentation is available in English only

RELATED CONCEPTS

Access Builder for SAP R/3

RELATED TASKS

Starting the Access Builder for SAP R/3
Customizing the Access Builder for SAP R/3
Navigating through a repository
Creating a local repository

Navigating through a repository

Access Builder for SAP R/3 provides two views for each of the two local repositories.

1. A tree-like view showing the organization of the repository
2. A sorted list showing only relevant object information
   For a Business Object Repository these are the Business Objects and for an RFC module repository these are the RFC modules.

Navigating through a tree-like view
By default, a repository is shown only with its top entries. For a Business Object Repository, the displayed items are the repository identifier and the top business domains. For an RFC module repository, the displayed items are the repository identifier and all logical RFC groups.

The visible area can be scrolled using the scrollbar on the right side of the view.

You can open or close domains by double-clicking them, by selecting **View > Expand** or **View > Collapse** from the menu bar, or by clicking the **Expand**

or **Collapse**

button in the toolbar.

You can also open or close all domains by selecting **View > Expand all** or **View > Collapse all** from the menu bar.

Navigating through a list view
You can scroll the view by using the scrollbar on its right side.

Navigating by searching
If the number of items in the current view is rather large and you know the item you are looking for (more or less) exactly you can search for the item.

To invoke the search window select **View > Search** in the menu bar, click the **Search**

button in the toolbar or click the right mouse button to invoke the pop-up menu and select **Search**.

RELATED CONCEPTS

Access Builder for SAP R/3
Local repositories
Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules
Creating a local repository
Searching for information
Customizing the Access Builder for SAP R/3

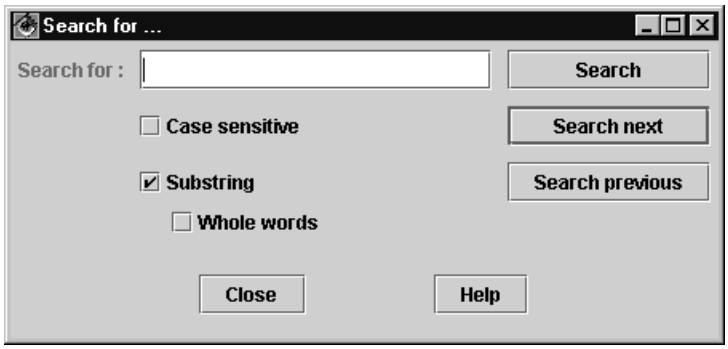
RELATED REFERENCES

BO and RFC proxy beans

Searching for information

You can search for information in any view currently displayed in the Access Builder's main window. You invoke the search window by selecting **View > Search** from the menu bar or by clicking the Search

button.



Perform the following steps to search for specific information:

- Enter the expression to be searched in the text field.
  Use wildcards (see definition below) to extend the range of possible search results.
- Select **Case sensitive** if the case should match.
  By default searching is not case-sensitive, so this check box is not checked.
- Select **Substring** if the expression should be treated as part of a single entry (see definition below). Otherwise the expression is treated to be the whole entry.
  The search is successful if the expression can be found anywhere in any entry.
  By default this check box is checked if the current view displays a tree.
  Otherwise it is not checked.
- Select **Whole words** if the expression should be treated as a word. This means that the expression must be separated from the rest of the entry by white spaces or common punctuation characters.
  This check box is only accessible if **Substring** is checked.
  By default this check box is not checked.
- Click **Search** to start a (new) search.
  Searching starts at the beginning of the data set (see definition below).
- Click **Search next** to continue searching.
  Searching starts at the current location in the data set and is performed to the end of the data set.
  If this button starts a new search the beginning of the data set is assumed to be the current location.
- Click **Search previous** to continue reverse searching.
  Searching starts at the current location in the data set and is performed to the beginning of the data set.
  If this button starts a new search the end of the data set is assumed to be the current location.

If the search is successful the Access Builder's main window will automatically show the entry containing the found expression.

Definitions

Data set
: The data set being used for searching depends on the current view displayed in the Access Builder's main window.
  If the current view in the Access Builder's main window is changed while the search facility is active the search facility is notified and changes its data set too.

Entry
: An entry is a single line of text in a data set.

Wildcards

Wildcards are placeholders for zero, one or more characters. The following wildcards are supported:

- '*': The asterisk is a placeholder for any number of characters.
- '?': The question mark is a placeholder for exactly one character.

Example

Assuming that the BOR hierarchy of an English IDES 4.5A system is currently displayed in the Access Builder's main window, the following search conditions

- Search expression: "*Code*"
- Case sensitive checked
- Substring not checked

will lead to the following results:

- CompanyCode
  This is found in the path Business Application Components / Financial Accounting / General Ledger Accounting / Basic Functions / CompanyCode
- Code group
  This is found in the path Business Application Components / Quality Management / Quality Planning / Basic Data / Code group

RELATED CONCEPTS

Access Builder for SAP R/3
Local repositories

RELATED TASKS

Navigating through a repository
Creating a local repository

Creating a local repository

The Access Builder can use locally stored Business Object repositories and RFC module repositories. This capability enables you to do most of your work with the Access Builder without being connected to a SAP R/3 system.

You can create a local Business Object Repository or an RFC module repository or both at any time by connecting to a SAP R/3 system and retrieving the information you are interested in. Typically, you perform this task if you access another SAP R/3 system. If you are aware of changes in your SAP R/3 system, you might want to update a locally stored Business Object or RFC module repository.

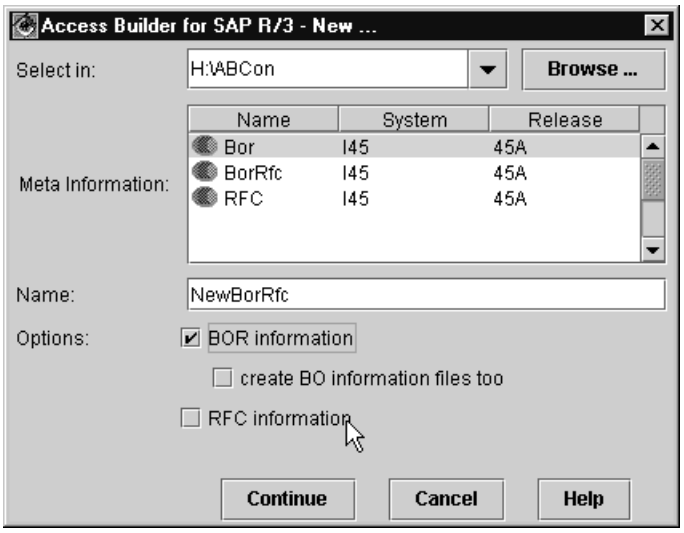
To retrieve the information:

1. Select **File > New** from the menu bar, click the **New**

   button in the toolbar or click the right mouse button to invoke the pop-up menu and select **New**.
   A dialog box opens which enables you to select the information you are

interested in and the files where you want it to be stored.



2. To log on to a SAP R/3 system provide all user and connection information in the logon dialog box.
3. After logging on to SAP R/3 a progress bar indicates the progress of the current operation.
   Important: The whole process may take from several minutes to about a day depending on the requested information and your system and network capabilities. For performance reasons you might want to start the Access Builder from outside of VisualAge for Java for this specific task.

The information is retrieved in the following order:

1. BOR information file (if selected)
   Important: Retrieving from an SAP R/3 Release 4.x may take only some minutes while retrieving from an SAP R/3 Release 3.1x may take up to some hours because much more communication traffic is required.
2. RFC module repository information file (if selected)
   Important: Retrieving from a typical SAP IDES R/3 system with its approximately 6000 RFC modules takes about 15 minutes.
3. BO information files (if selected)
   This is only possible if you also retrieve the BOR information file.
   Important: This step may take about a day to perform. So use it with caution!

After retrieving all information the set of possible operations depends on the repositories currently available.

Further operations on a local Business Object Repository
To generate the documentation for the local BOR select **Build > All documentation** from the menu bar. This means that HTML documentation for all business objects in the BOR is generated and additionally an index of the BOR itself is created.

To also generate Java proxy beans for all Business Objects in the local BOR, select **Build > All Java Proxy beans** in addition, or select **Build > Generate all** instead of **Build > All documentation**. The kind of generated Java proxy beans depends on the current options which you can select in the **Options window**.

The files are retrieved in the following order:

- BO information files if required
  If no connection is currently established to your SAP R/3 system you are prompted to provide your logon information because missing information must be retrieved from SAP R/3.
- BO documentation files
- BO hierarchy and index files
- Java source files of the BO proxies if selected

Further operations on a local RFC module repository
Because of the huge number of available RFC modules in an RFC module repository it is neither necessary nor useful to generate the documentation or Java Proxy beans or both of all RFC modules. However it is possible to generate documentation or Java Proxy beans or both for some selected RFC modules.

To generate this information perform the following steps:

1. Select all the RFC modules you are interested in
2. Select **Build > Selected HTML documentation** from the menu bar to generate documentation for the selected RFC modules
3. Select **Build > Selected Java Proxy bean(s)** from the menu bar to generate Java Proxy beans for the selected RFC modules

If no connection is currently established to your SAP R/3 system you are prompted to provide your logon information because missing information must be retrieved from SAP R/3.

RELATED CONCEPTS

Access Builder for SAP R/3
Local repositories

RELATED TASKS

Starting the Access Builder for SAP R/3 outside of VisualAge for Java
Loading an local repository
Updating a local repository
Navigating through a repository
Customizing the Access Builder for SAP R/3

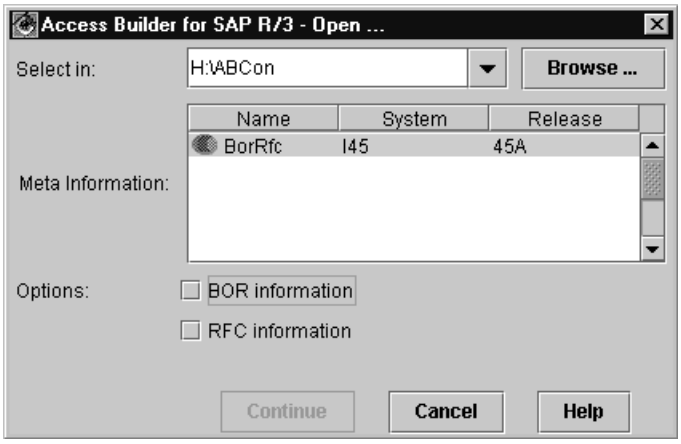
Loading a local repository

The Access Builder can use locally stored Business Object Repositories and RFC module repositories. This capability enables you to do most of your work with the Access Builder without being connected to any SAP R/3 system.

You can load an existing local Business Object Repository or an RFC module repository or both at any time.

To load a repository:
Select **File > Open** from the menu bar, click the Open

button in the toolbar or click the right mouse button to invoke the pop-up menu and select **Open**.
A window is shown which enables you to select the repository files containing the information you are interested in.



RELATED CONCEPTS

Access Builder for SAP R/3
Local repositories

RELATED TASKS

Starting the Access Builder for SAP R/3 outside of VisualAge for Java
Creating a local repository
Updating a local repository
Navigating through a repository
Customizing the Access Builder for SAP R/3

Updating a local repository

A locally stored repository represents a snapshot of the current state of an SAP R/3 system at the creation time of the repository. Any changes in the SAP R/3 system after creating the local repository are not reflected there until a new snapshot is created.

The following changes may occur in a SAP R/3 system:

- The Business Object Repository has been changed
  Business domains may have been changed, added or removed, or any kind of objects may have been added or removed.
- Existing Business Objects have been changed
  Meta information of any Business Object may have been changed, for example keys, attributes, methods and related parameters, events or documentation may have been added, changed, or removed.
- The RFC module repository has been changed
  Single RFC modules or logical groups may have been added, changed, or removed.

Depending on the changes mentioned above different actions may be necessary to restore matching between the local repositories and the changed SAP R/3 system. The following sections describe what to do in either case.

The Business Object Repository has been changed
If you are aware of changes in the Business Object Repository and you want to update your locally existing BOR, you have to perform the following steps:

1. Open the local Business Object Repository

2. Select **File > Update** from the menu bar
   The system tries to connect to the SAP R/3 system using the same connection information as used to create the repository and retrieves all BOR information again.

Existing Business Objects have been changed
If you are aware of changes of Business Objects and you want to update all meta information of Business Objects, you have to perform the following steps:

1. Open the local Business Object Repository
2. Select **Build > (Re-)Generate BO information files** from the menu bar
   The system tries to connect to the SAP R/3 system using the same connection information as used to create the repository and retrieves all BO meta information.

   Important:
   This may take a very long time (up to a day depending on your system, SAP R/3, and network capabilities).
   Hint: Perform this at weekend.

The RFC module repository has been changed
If you are aware of any changes in the RFC module repository and you want to update your locally existing RFC module repository, you have to perform the following steps:

1. Open the local RFC repository
2. Select **File > Update** from the menu bar
   The system tries to connect to the SAP R/3 system using the same connection information as used to create the repository and retrieves all RFC module repository information again.

If you are aware of any changes in one or more specific RFC modules and you want to update their meta information you are required to do some work manually:

- Open the affected RFC module repository in Access Builder and remember the path in the file-system where the repository is located
- Open a command line window of your system and switch to the path remembered above
- Enter the subdirectory **BORname.META**, where **BORname** is the name of the BOR to be opened, and enter the subdirectory **ser**.
  In this directory there may exist a very large number of files.
- Delete all files corresponding to those RFC modules you want to get updated
  These files have the following name signature: RFC_Name.ser
- Back in Access Builder select all RFC modules you want to get updated
- Generate either Java proxies or documentation for these RFC modules

RELATED CONCEPTS

Access Builder for SAP R/3
Local repositories

RELATED TASKS

Creating a local repository
Loading an local repository
Navigating through a repository
Customizing the Access Builder for SAP R/3

Generating proxy beans for SAP Business Objects and RFC modules

The Access Builder for SAP R/3 provides several ways to generate proxy beans for Business Objects and RFC modules.

Before you can start generating proxy beans you first need to know the following:

- Do I need proxies for Business Objects or RFC modules?
- Which proxies do I need? How many proxies do I need?
- How good do I know my SAP R/3 system?
- Which kind of proxies do I need? Which infrastructure should the proxies support?

Depending on the answers to the questions above you may have to retrieve the repositories from SAP R/3 and store them locally before starting to generate your proxies.
After loading or retrieving the corresponding repositories into the Access Builder switch to a view in which the objects you are interested in are located. Select the objects and set up the appropriate generation options to your needs as described in the following.

You set up the generation options in the **Generation** page of the **Options** window as follows:

- Click on the radio button beside the preferred proxy bean
- Change the default path for these beans to your needs if required (on the right of the radio button)
- Change the default package name for these beans to your needs if required (below the path entry field)
- Select the checkboxes which will satisfy your preferred behaviour in handling documentation

After returning from the Options window select **Build > Selected Java proxy bean(s)** from the menu bar or click the **Generate**

button in the toolbar.

RELATED CONCEPTS

Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules

RELATED TASKS

Customizing the Access Builder for SAP R/3
Creating a local repository
Navigating through a repository
Running your application
Using the proxy beans to access SAP Business Objects
Calling RFC modules

RELATED REFERENCES

Naming conventions for generated classes
BO and RFC proxy beans
Options window

Generating documentation for SAP Business Objects and RFC modules

The Access Builder for SAP R/3 provides several ways to generate documentation for Business Objects and RFC modules.

Typical generation
After loading or retrieving the corresponding repositories into the Access Builder switch to a view in which the objects you are interested in are located. Select the objects. Select **Build > Selected HTML documentation** from the menu bar, click the Generate documentation

button in the toolbar or click the right mouse button and select the **Selected HTML documentation**.

Another way of generating HTML documentation for the selected view is to generate all at once. To do this, select **Build > All HTML documentation** from the menu bar.

RELATED CONCEPTS

Reference documentation
Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules

RELATED TASKS

Customizing the Access Builder for SAP R/3
Creating a local repository
Navigating through a repository

RELATED REFERENCES

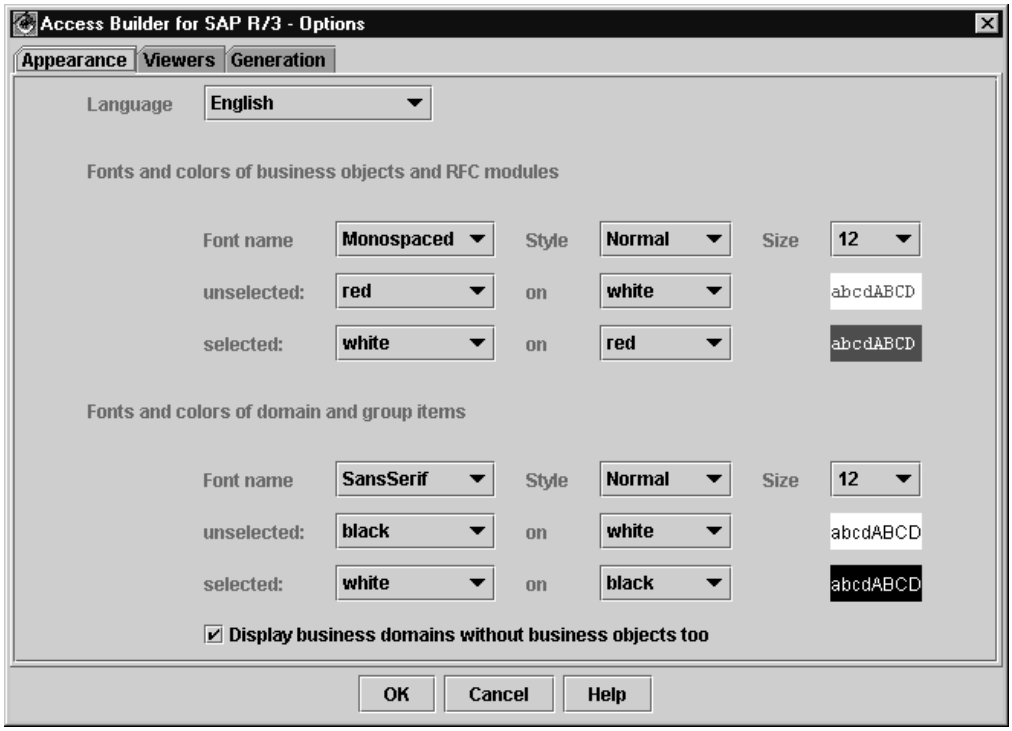
Advanced Generation window

Customizing the Access Builder for SAP R/3

You can customize several aspects of the Access Builder for SAP R/3:

- Appearance, such as language and colors
- Viewers for displaying different categories of information
- Generation options for creating Java proxy beans.

Customizing appearance



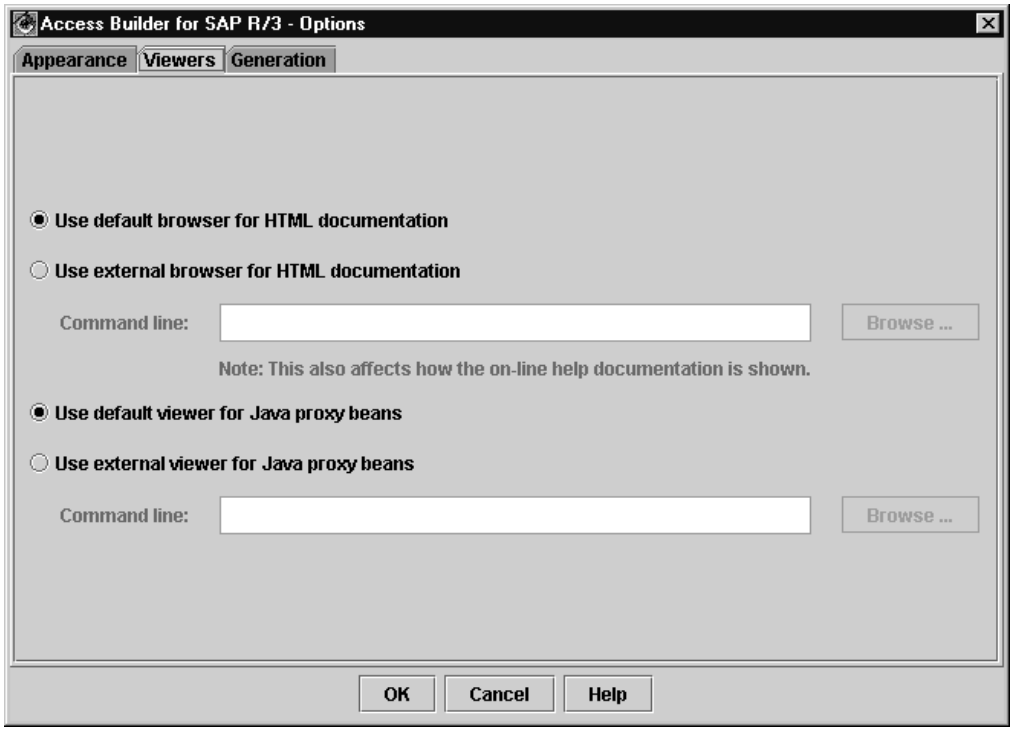
To customize the appearance:

1. Select **Options > Change options** from the menu bar. The Access Builder for SAP R/3 - Options window appears.
2. In the Appearance page, change the settings you want to alter.
3. Click **Ok** to apply the changes and close the window, or **Cancel** to close the window without applying the changes.

To save the changes permanently, select **Options > Save options** from the menu bar.

Customizing viewers

You can customize the Access Builder to use your preferred external viewers for displaying information instead of the default viewers.

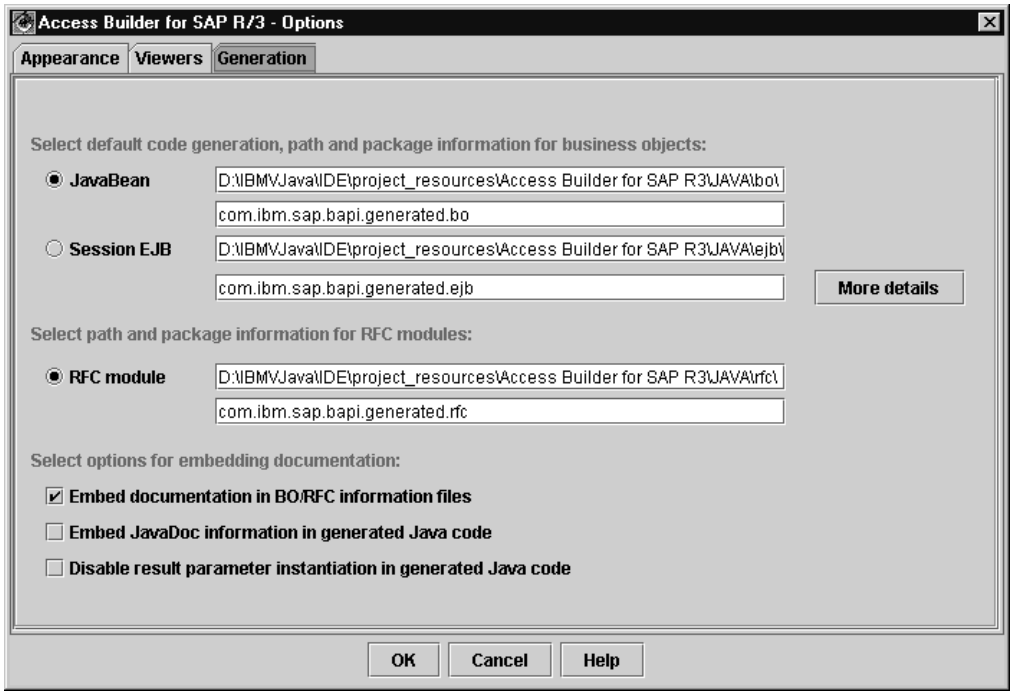


To specify an external viewer:

1. Select **Options > Change options** from the menu bar. The Access Builder for SAP R/3 - Options window appears.
2. Click the **Viewers** tab.
3. Select the appropriate **Use external viewer** option
4. Type the external viewer in the **Command line** field. Provide the full path and all required parameters.
   The fields are empty when the default viewers are selected.
5. Click **Ok** to apply the changs and close the window, or **Cancel** to close the window without applying the changes.

Customizing the generation of Java proxy beans

The Access Builder allows to generate different kinds of proxy beans for Business Objects and RFC modules. For each type of generated proxy beans, you can customize generation options like the path where generated files are to be stored and the name of the package to which the generated files belong.



To customize the generation options:

1. Select **Options > Change options** from the menu bar. The Access Builder for SAP R/3 - Options window appears.
2. Click the **Generators** tab.
3. Type path and package information for the different generation types
4. Select the type of proxy beans to be generated for Business Objects
5. If applicable, click the **Details** button to change additional generation options
6. Select the options for embedding documentation
7. Click **Ok** to apply the changes and close the window, or **Cancel** to close the window without applying the changes.

RELATED CONCEPTS

Access Builder for SAP R/3
Local repositories

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC Modules
Creating a local repository

Chapter 9. Developing an application

Connecting to the SAP R/3 system

The classes which the Access Builder for SAP R/3 generates help you access and manipulate the information in an SAP R/3 system through Business Objects, BAPIs, and RFCs. Before you can work with this information, you must connect to the SAP R/3 system. To do this, you must prepare the middleware you want to use to access the SAP R/3 system.

Selecting and preparing the middleware access method
The Common RFC Interface for Java enables you to choose from a set of different middleware types at run time. The Common RFC Interface for Java classes and interfaces are packaged in com.sap.rfc. This common interface supplies a set of base classes for various types of middleware. In addition, every middleware implementation comes with its own set of factory classes that are used to create instances of the base classes.

Select a set of factory classes through the factory manager. There is one global FactoryManager instance, and you cannot create further instances. Access this FactoryManager instance through a static method:

```
FactoryManager aFactoryManager = FactoryManager.getSingleInstance();
```

Define the middleware type with a MiddlewareInfo instance. For example, to select the JNI middleware type your code might look like this:

```
MiddlewareInfo aMiddlewareInfo = new MiddlewareInfo();
aMiddlewareInfo.setMiddlewareType( MiddlewareInfo.middlewareTypeJNI);
```

You can use the MiddleWareInfo(String[] args) constructor to simply pass in command line arguments. This constructor allows you to select the middleware type through command line parameters:

```
MiddlewareInfo aMiddlewareInfo = new MiddlewareInfo(args);
```

This constructor is typically invoked in the main() method of a program to pass in the command line parameters. The middleware type parameter passed into the constructor can be

-conn "JNI"
to select the JNI middleware shipped with Access Builder and Connector for SAP R/3. This middleware is used by default.
The quotes may be omitted.

-conn "Orbix orb-server"
to select the JRFC middleware available from SAP. The orb-server defines the host name of the JRFC server.

The constructor ignores any other data on the command line.

Finally, you globally set the middleware type in the factory manager. This ensures that all subsequent requests get the factory classes of the selected middleware type.

```
aFactoryManager.setMiddlewareInfo( aMiddlewareInfo);
```

Important note: The call to setMiddlewareInfo() creates the factory instances for the IRfcConnectionFactory, IRfcModuleFactory, ISimpleFactory, IStructureFactory,

and ITableFactory under the cover. Therefore, setMiddlewareInfo() **must** be called before attempting to retrieve a factory instance from the factory manager.

Providing user and connection information
SAP R/3 provides different ways to connect to a specific SAP R/3 system. You can specifiy the system's host name and system number directly, or you can use load-balancing by providing the name of a message server. If the connection is routed through a gateway server you also have to specify the host name and service number of the gateway.

Define the connection and user information in ConnectInfo and UserInfo instances. The code below gives you an example of what this might look like:

```
ConnectInfo aConnectInfo = new ConnectInfo(
              3,                    // Rfc mode 3, requires SAP R/3 system
              null,                 // aDestination
              "your.r3.host",       // TCP/IP name of SAP R/3 host
              21,                   // number of SAP R/3 system
              null,                 // TCP/IP name of gateway host
              null,                 // number of gateway host service
              null,                 // system name
              null,                 // name of SAP R/3 system group
              null,                 // name of SAP R/3 message server
              false,                // true: use load balancing
              true ) ;              // true: check authorization immediately
UserInfo aUserInfo = new UserInfo (
              "userid",             // SAP R/3 user name
              "password",           // SAP R/3 password
              "800",                // SAP R/3 client number
              "e" ) ;               // national language used by SAP R/3
```

Establishing a connection to the SAP R/3 system
To establish a connection to the SAP R/3 system you can either use the connection manager of Access Builder and Connector for SAP R/3 which supports connection pooling or you can create a connection with the connection factory from the factory manager. With connection pooling you can reuse existing connections to the SAP R/3 system.

Using the connection manager
The connection manager manages a pool of connections. You can obtain a connection from the pool by calling either the `reserve`- or the `reserveWait`-method of the ConnectionManager. Your code might look like this:

```
IRfcConnection aConnection =
  ConnectionManager.getSingleInstance().reserve(aConnectInfo, aUserInfo);
```

If you use `reserveWait` and no connection can be obtained from the pool at the moment, this method retries to obtain a connection after waiting the amount of time specified in the property waitTimeWhenNoConnectionAvailable of the connection manager. This is repeated until successful.

If you don't need the connection any longer you can return it to the pool with the release-method:

```
ConnectionManager.getSingleInstance().release(aConnection);
```

Note that the connection shouldn't be closed neither before nor after releasing it.

You can specify the maximum number of connections available in the pool by setting the maxConnections property of the connection manager.

Creating a connection with the connection factory
To establish the SAP R/3 connection using the selected middleware retrieve the connection factory from the factory manager and create a connection instance from this connection factory.

```
// retrieve factory for connections (using current middleware type)
IRfcConnectionFactory aConnectionFactory =
   FactoryManager.getSingleInstance().getRfcConnectionFactory();
// establish a new SAP R/3 connection with the given user and system information
IRfcConnection aConnection =
   aConnectionFactory.createRfcConnection( aConnectInfo, aUserInfo);
aConnection.open() ;
```

If your program or applet uses multiple threads make sure that each thread that calls a Business Objects method or an RFC method uses its own connection. You are now ready to retrieve and manipulate information in the SAP R/3 system.

When you have finished working with the SAP R/3 system, you can close the connection at any time using the following method:

```
aConnection.close();
```

To re-open the connection with the same user and connection information, type:

```
aConnection.open();
```

RELATED CONCEPTS

Connection pooling
Run-time classes for SAP R/3
BAPI Transactions

RELATED TASKS

Integrating the Logon bean into your application
Using the proxy beans to access SAP Business Objects
Calling RFC modules
Tracing the SAP R/3 connection of your application
Running applets with JNI code in AppletViewer
Running applets with JNI code in Netscape Communicator
Running applets with JNI code in Java Plug-In

RELATED REFERENCES

Common RFC Interface for Java
Run-time classes for SAP R/3

Integrating the Logon bean into your application

The Logon bean and all related classes are located in the com.ibm.sap.bapi.util.logon package of the IBM Access Builder for SAP R3 Libraries project.

Connecting the LogonModel to the Logon bean
Generally speaking, you always connect the `Logon` bean with an object implementing the LogonModel interface. The DefaultLogonModel provides a default implementation of this interface.

Version 3.5 comes with the new `ExtendedLogonModel` which provides additional functionality used to change password on the SAP R/3 system. If you want the logon bean to provide this functionality you have to use this model.

There are two ways to connect the Logon bean to the Logon model (select the appropriate one):

- First you instantiate a DefaultLogonModel or ExtendedLogonModel object and a Logon object. Then you call the Logon.setLogonModel() method to provide the Logon bean with the model. Since this method also registers all necessary listeners nothing else needs to be done. If you want set the default property values for the ConnectInfo or the UserInfo object or both within your development environment, you must use the custom property editors of the Logon model.
- You call the Logon.createDefaultLogonModel() or then Logon.createExtendedLogonModel() method to create an instance of DefaultLogonModel or ExtendedLogonModel. This method also registers all necessary listeners. In this case you have to set the default property values for the ConnectInfo or the UserInfo object or both within your development environment using the custom property editors of the Logon bean.

Specifying location and name of the logon ini file
During development time you specify the name and the location of the logon ini file by using the `defaultFileNameSapLogonIni` property. During run time you set the name of the ini file by using the `Logon.setFileNameSapLogonIni()` method. The name and location of the logon ini file can be either a valid file name (path + file name, for example c:\\winnt\\saplogon.ini) or a valid URL (for example http://myhost/saplogon.ini).

Using the Logon bean
After initializing the Logon bean and the LogonModel you usually call the bean's `show()` method. After that the user enters the logon information and logs on to the SAP R/3 system. Depending on whether the logon succeeds or fails the model sends event notifications to registered listeners using the Logon event. You use the LogonSucceeded notification to obtain the RfcConnection object reference from the model by calling the getConnection() method. Because the Logon bean does not disappear after a successful logon you should listen to the logonSucceeded event and call the Logon.dispose() method.

Working with the Logon bean visually
In VisualAge for Java, you can visually develop your application in the Visual Composition Editor. To use the Logon bean in the Visual Composition Editor you perform the following steps:

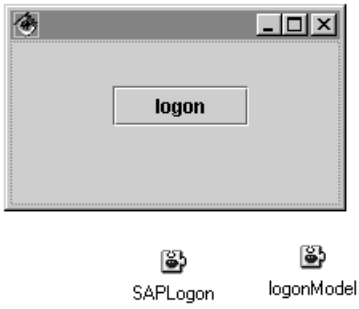
1. Add the logon bean to the composition editor
2. Add the DefaultLogonModel or ExtendedLogonModel
3. Set default properties
4. Connect the Logon bean to the LogonModel
5. Work with the Logon bean

Adding the Logon bean in the composition editor

1. Click **Choose Bean** in your Composition Editor.
2. Click the **Browse** button to select a valid class.
3. Type `Logon` until the correct class and package name are selected and click **OK**.
4. Specify **Class** as bean type.
5. Provide an appropriate name like SAPLogon and click **OK**.

Adding the DefaultLogonModel or ExtendedLogonModel to the composition editor

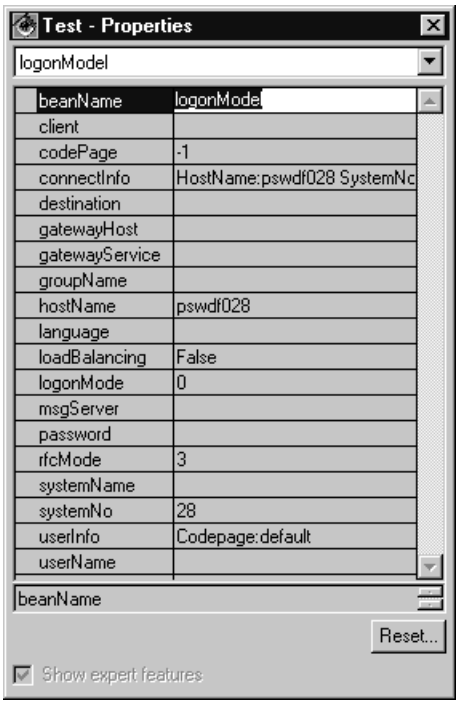
1. Click **Choose Bean** in your Composition Editor.
2. Click the **Browse** button to select a valid class.
3. Type `DefaultLogonModel` or `ExtendedLogonModel` until the correct class and package name are selected and click **OK**.
4. Specify **Class** as bean type if you want to create a new instance of the Logon model or **Variable** if you want to use the Logon.createDefaultLogonModel() or Logon.createExtendedLogonModel() to preserve your custom property settings
5. Provide an appropriate name like logonModel and click **OK**.



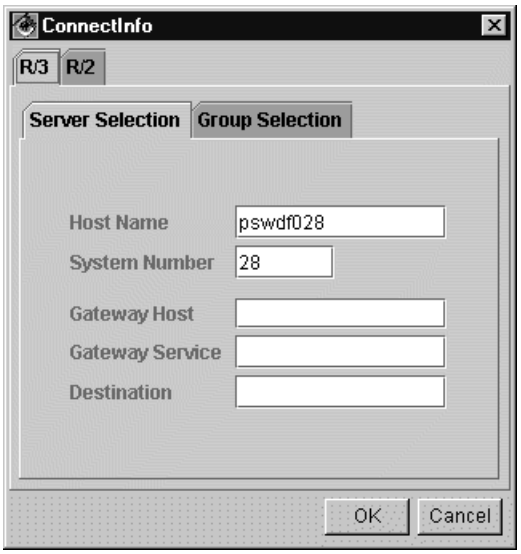
Setting default properties

To preset connection information that will never change

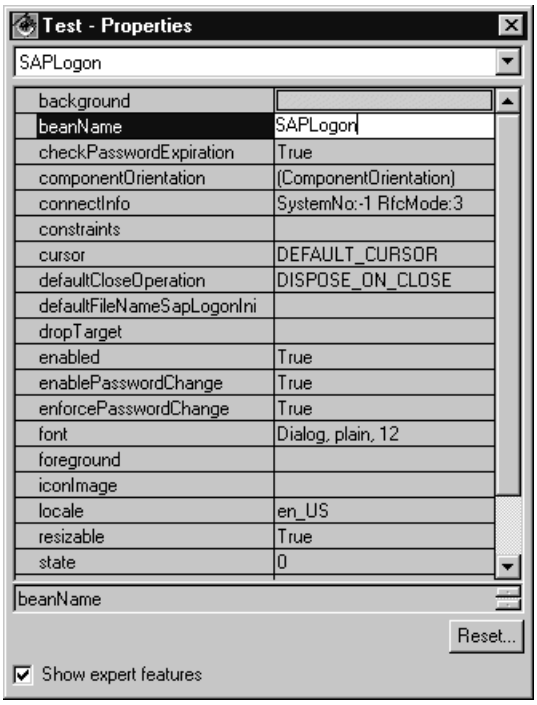
1. Double-click the Logon model. The Properties dialog box opens.



2. Select the logonConnectInfo property and click the **...** button. The ConnectInfo prompter opens.



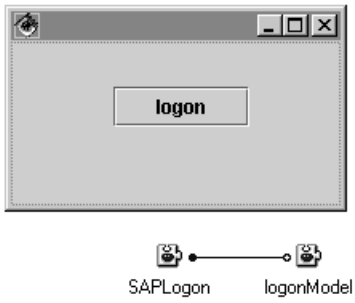
3. In the ConnectInfo prompter you can preset all connection information. For UserInfo a corresponding panel is available to change these settings in the logon dialog.
4. Double-click the Logon bean. Set the location of the logon ini file in the defaultFileNameSapLogonIni property. You can use either a file name or an URL.



Connecting the Logon bean to the LogonModel
There are two ways to connect the two beans: with the Logon.setLogonModel() or with the Logon.createDefaultLogonModel() / Logon.createExtendedLogonModel() method.

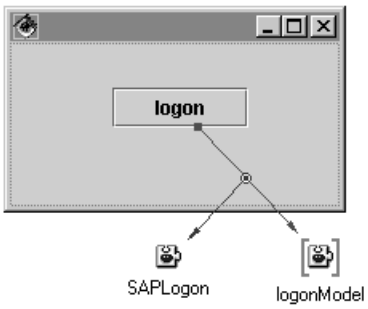
If you use Logon.setLogonModel() the type of the Logon model should be **Class**.

1. Create a connection from the `this` property of the logon model to the logonModel property of the Logon bean.



If you use the Logon.createDefaultLogonModel() or Logon.createExtendedLogonModel() method on a specific event

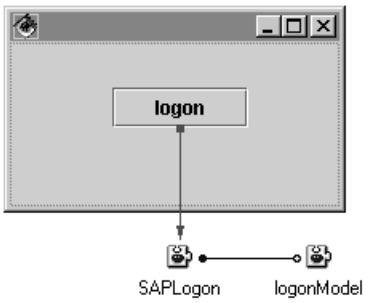
1. Connect the actionPerformed event from the button to the `this` property of the logon Model.
2. Connect the `value` property of the previously created connection to the createDefaultLogonModel or createExtendedLogonModel method of the Logon bean.



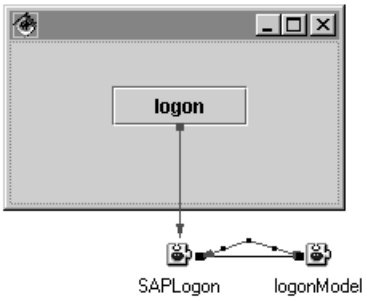
Working with the Logon bean

The Logon bean needs five gif files to display the first row of the SAP servers list correctly (as an example see ″saplogon.ini″ topic below). These files are SingleServerSel.gif, ServerGroupSel.gif, SingleServerSel.gif, ServerGroupSelSNC.gif, SingleServerSelSNC.gif and are located in the `directory_of_VAJ\ide\project_resources\IBM Access Builder for SAP R3 Libraries` directory. After intergrating the logon bean in your application you have to copy these gif files in the current directory of your application.

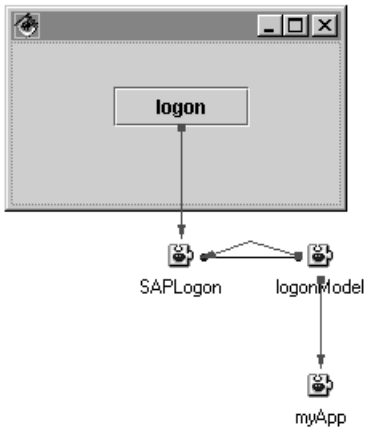
1. To make the logon window visible after the logon button has been clicked create a new connection from the actionPerformed() event of the logon button to the show() method of the Logon bean.



2. To automatically dispose of the logon window, when the logon is successful create a new connection from the logon model logonSucceeded() event to the dispose() method of the logon window.



3. After the logon...
4. Listen to the logonSucceeded event and dispatch to your application.



Enabling the change password feature
The change password feature of the Logon bean is manipulated using the three properties enablePasswordChange, checkPasswordExpiration, and enforcePasswordChange. For a detailed description of these properties see ″Logon bean″ topic below.

RELATED CONCEPTS

Logon bean
saplogon.ini

RELATED REFERENCES

Logon bean

Using the proxy beans to access SAP Business Objects

Before you can use the proxy beans you must know which Business Object you want to use in your Java program and you must have created the proxy beans for that Business Object.

The BO proxy bean represents the Business Object. It contains constructors to instantiate the bean, methods that allow the invocation of BAPI methods, and methods for handling the object identifier.

The Business Framework defines three categories of BAPI methods:

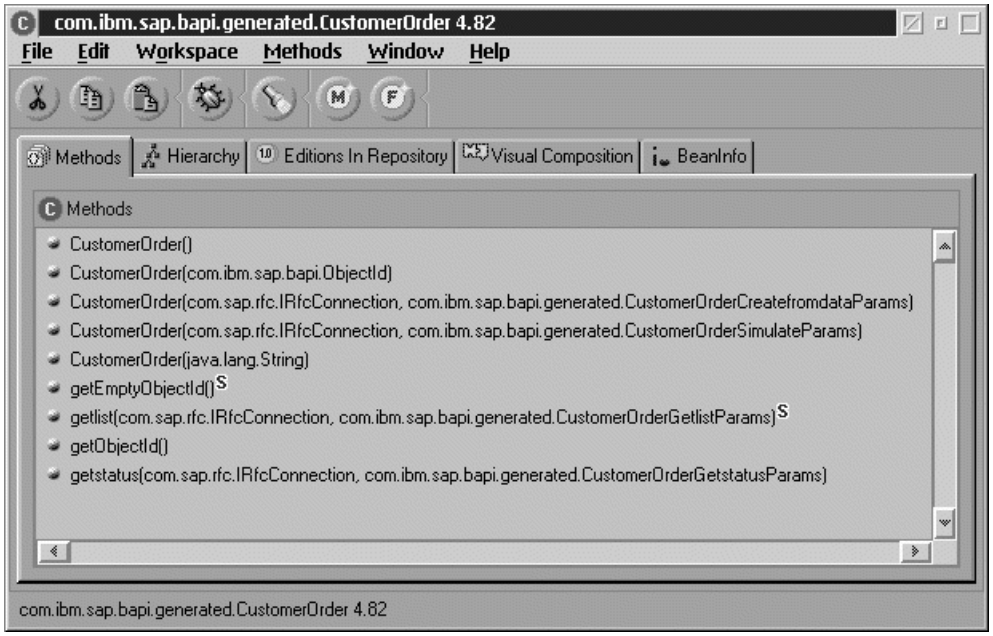
Instance methods
Instance methods are invoked on a concrete Business Object instance.

Class methods
Class methods can be invoked without relationship to any concrete Business Object instance.

Factory methods
Factory methods are used to create new Business Object instances.

The generated proxy beans use the respective Java concepts of instance methods, static methods, and constructors for these categories. Note that you have to pass a connection to an SAP R/3 system when calling any BAPI method of the Business Object. This connection must be valid in the context of the thread calling the method.

In the following instructions the BO proxy bean for the business object CustomerOrder (object type BUS2032) is used as an example.



Invoking a class method
Class methods are mapped to Java static methods. Therefore you do not need to create an instance of a BO proxy bean to invoke a class method.

A parameter container class is generated for each BAPI method of a business object. This class is used as a container for the method parameters. Before you invoke the BAPI method you create an instance of the corresponding parameter container class. The following code shows how to invoke the getlist BAPI method of the CustomerOrder Business Object:

```
CustomerOrderGetlistParams getlistParams = new CustomerOrderGetlistParams();
CustomerOrder.getlist(connection, getlistParams);
```

The results of the method call are passed back in the parameter container.

Setting the import parameters
To set any required or optional import parameters before invoking the method getlist, provide the following code:

```
// Create parameter container
CustomerOrderGetlistParams getlistParams = new CustomerOrderGetlistParams() ;
// Fill import parameters
getlistParams.setCustomerNumber("1234567890");
getlistParams.setSalesOrganization("0001");
...
```

The getlist method does not require any tables or structures as import parameters.

Getting information from a structure parameter
If a parameter container class contains a structure parameter an instance of the corresponding structure class is created with an instance of the parameter container class. Get and set access methods are provided to retrieve and set the data in the structure.

The getlist method returns information about the successful completion in the BapireturnStructure parameter. You access the structure parameter with the following statement:

```
BapireturnStructure returnStructure = getlistParams.getReturn();
```

To retrieve the message from the Bapireturn structure use the following code:

```
String message = returnStructure.getMessage();
```

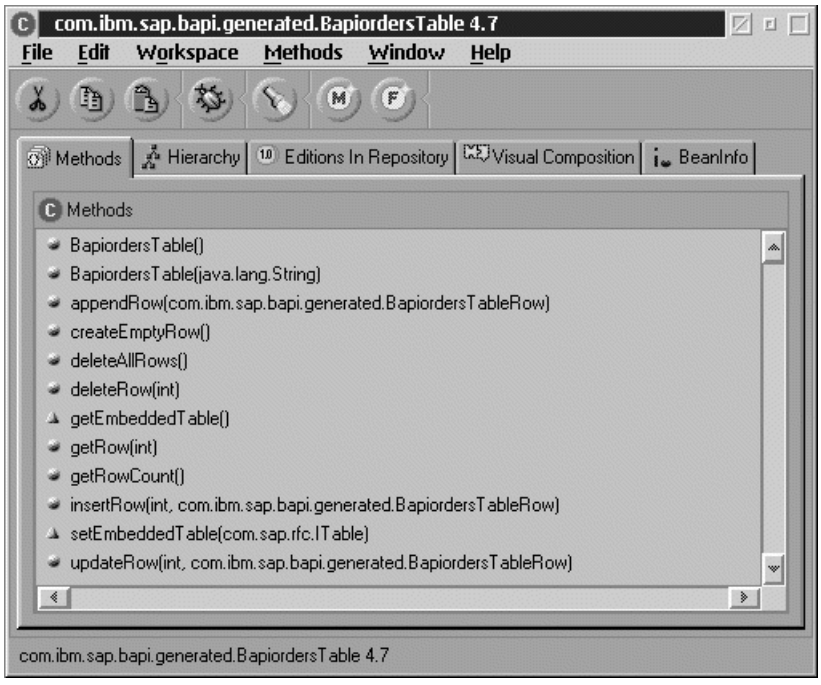
Getting information from a table parameter
The table class represents an SAP R/3 table that is used as parameter in a BAPI method. Access methods are provided to access rows in the table. The table row class represents one row in the table. Table row classes are handled exactly the same way as structure classes: get and set access methods are provided to retrieve and set the columns in the row.

If a parameter container class contains a table parameter an instance of the corresponding table class is created with an instance of the parameter container class.

The getlist method returns information about the list of sales orders in the BapiordersTable parameter. You access the table parameter with access methods:

```
BapiordersTable ordersTable = getlistParams.getSalesOrders();
```

The BapiordersTable is shown in the picture below.



The following code outlines how to iterate over the whole table and retrieve the rows stored in the table:

```
int rowCount = ordersTable.getRowCount();
BapiordersTableRow row = null;
for (int i=0; i<rowCount; i++)
```

```
{
    row = ordersTable.getRow(i);
    // retrieve data elements from row
}
```

Invoking an instance method
BAPI instance methods are mapped to instance methods of a BO proxy bean. Therefore you first have to create an instance of a BO proxy bean. As instances of SAP Business Objects are identified by one or more object keys, you must set the key(s) of the Business Object instance also.

Creating an instance of a BO proxy bean.
Each proxy bean has a constructor with a parameter list that contains a single key of the corresponding Business Object. For example, the business object CustomerOrder has only one key field, the field SALESDOCUMENT. So the easiest way to create a new instance of class SalesOrder with a well-known key is to use the following syntax:

```
CustomerOrder aCustomerOrder = new CustomerOrder("0005000001");
```

When this lone of code has been executed the variable aCustomerOrder contains a reference to the Business Object with the key 0005000001.

An instance method of the Business Object proxy bean can now be invoked.

Setting the key(s) after class creation
For each key of an object access methods with the following signatures exist:

- public <Key_type> getKey_<Key_name>() ;
  for example `public java.lang.String getKey_Salesdocument();`
- public void setKey_<Key_name>(<Key_type> <Key_name>) ;
  for example `public void setKey_Salesdocument(java.lang.String salesdocument);`

Note: Classes generated by previous releases used a separate class ObjectId for the key handling. This is no longer required.

The access methods are a much more convenient way of dealing with the keys of an object. You can use simple string parameters to set complex data types like date, time, and number objects.

Invoking the instance method
As the keys are retrieved from the previously defined object identifier, they are passed implicitly. Key(s) are not included in the parameter container and you do not need to pass them explicitly. You might set required or optional import parameters as you do with a class method.

The following code shows how to invoke the getstatus method of the CustomerOrder Business Object:

```
CustomerOrderGetstatusParams getstatusParams = new CustomerOrderGetstatusParams();
aCustomerOrder.getstatus(connection, getstatusParams);
```

As with class methods, the results of the above method call are passed back into the parameter container.

Invoking a factory method
In the SAP R/3 system, a factory method takes as input all the data necessary for creating the persistent data representation of a Business Object in the SAP R/3

database. Factory methods are mapped to constructors of BO proxy beans. The constructor invokes the corresponding BAPI factory method under the covers and constructs an instance of the specified BO proxy bean. If an error occurs an exception describing the error condition is passed back to the caller. If no error occurs the object's keys are filled according to the return values of the BAPI method and a completely initialized instance of the BO proxy bean is returned to the caller.

The following code example shows you how to create an instance of a BO proxy bean. You have to provide a valid connection to an SAP R/3 system for calling this constructor, as you did when calling an instance method.

```
CustomerOrderCreatefromdataParams createfromdataParams =
        new CustomerOrderCreatefromdataParams();
// Fill the data for your new CustomerOrder
// into createfromdataParams
CustomerOrder aSalesOrder = new CustomerOrder(connection, createfromdataParams);
```

Providing the data in the parameter container class can be a somewhat lengthy undertaking because the data structures of Business Objects are relatively complex.

RELATED CONCEPTS

SAP interface technologies
Proxy beans for Business Objects
Enterprise bean proxies for Business Objects

RELATED TASKS

Navigating through a repository
Generating proxy beans for SAP Business Objects and RFC modules
Running your application
Connecting to the SAP R/3 System
Calling RFC modules
Developing SAP R/3 applications using the InfoBus technology
Tracing the SAP R/3 connection of your application

RELATED REFERENCES

Naming conventions for generated classes
BO and RFC proxy beans

Calling RFC modules

Access Builder for SAP R/3 enables you to call common RFC modules in any R/3 system. The RFC module is called from Java with the help of an RFC proxy bean.

You generate the RFC proxy bean with the Access Builder. The name of the generated RFC proxy bean class is the same as the name of the encapsulated RFC module. All parameters which the RFC module uses are accessed as attributes of the RFC proxy bean.

Additional classes are generated for table or structure parameters. For example, the RFC_SYSTEM_INFO RFC module uses one structure, so two classes are generated:

- RFC_SYSTEM_INFO
- RfcsiStructure

Using an RFC proxy bean in your code

Typically, you perform the following steps:

1. You might want to include an import statement for the package of generated classes at the beginning of your code. The following line refers to the default package name for generated classes.
   ```
   import com.ibm.sap.bapi.generated.*;
   ```
2. Create an instance of the RFC proxy bean for further use:
   ```
   RFC_SYSTEM_INFO rfcSystemInfo = new RFC_SYSTEM_INFO();
   ```
3. Instantiate the import parameters and pass them to the RFC proxy with the set methods. With RFC_SYSTEM_INFO, there is no import parameter.
4. Execute the RFC module with the execute() method:
   ```
   rfcSystemInfo.execute();
   ```
5. Use the get methods to retrieve the export parameters. In this case you are probably interested in the system information structure RfcsiStructure:
   ```
   RfcsiStructure si = rfcSystemInfo.getRfcsiStructure();
   ```
6. Retrieve the data in this structure with the appropriate get methods and write the SAP R/3 system information to the screen:
   ```
   System.out.println( "SAP Information");
   System.out.println( "————————-");
   RfcsiStructure si = rfcSystemInfo.getRfcsi_export();
   System.out.println( "Destination : " + si.getRfcdest() + "\n");
   System.out.println( "Host        : " + si.getRfchost());
   System.out.println( "System ID   : " + si.getRfcsysid());
   System.out.println( "Database    : " + si.getRfcdatabs());
   System.out.println( "DB host     : " + si.getRfcdatabs());
   System.out.println( "DB system   : " + si.getRfcdbsys() + "\n");
   System.out.println( "SAP release : " + si.getRfcsaprl() + "\n");
   System.out.println( "RFC Protocol: " + si.getRfcproto());
   System.out.println( "Characters  : " + si.getRfcchartyp());
   System.out.println( "Integers    : " + si.getRfcinttyp());
   System.out.println( "Floating P. : " + si.getRfcflotyp());
   System.out.println( "SAP mach id : " + si.getRfcmach());
   System.out.println( "Timezone    : " + si.getRfctzone() + si.getRfcdayst());
   ```

RELATED CONCEPTS

SAP interface technologies
Proxy beans for RFC modules

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules
Running your application
Connecting to the SAP R/3 system
Using the proxy beans to access SAP Business Objects

RELATED REFERENCES

BO and RFC proxy beans

Chapter 10. Using the CCF Connector for SAP R/3

Setting up the CCF infrastructure

Since the CCF Connector for SAP R/3 relies on the CCF infrastructure implementation in regards of support for security, transactions, connection pooling, etc., it is essential that the CCF infrastructure is set up correctly. If this isn't done by your component server or in case you are developing a fat Java application you have to initialize the respective objects of the Common Connector Framework manually. Please have a look at Setting Up the Common Connector Framework RuntimeContext chapter of the CCF documentation for details.

RELATED CONCEPTS

CCF Connector for SAP R/3
Common Connector Framework

Building EAB Commands using proxy beans for RFC modules

Since the general process for creating and editing Enterprise Access Builder (EAB) Commands and Navigators is the same for all CCF connectors, this chapter only describes the CCF Connector for SAP R/3 specific aspects of building EAB Commands. Please see the Enterprise Access Builder for Transactions documentation if you are not familiar with the EAB concepts and usage of the EAB tools.

As described in the CCF Connector for SAP R/3 chapter the process of modelling interactions with SAP R/3 systems typically involves the following steps:

1. Identify the required SAP R/3 data and/or business logic.
2. Identify the appropriate RFC modules to access the data/business logic.
3. Generate proxy beans for the required RFC modules with Access Builder for SAP R/3.
4. Create and edit EAB Commands using the generated RFC proxies as input/output beans.
5. Optionally compose Navigators using Commands and/or Navigators created in steps 4 and 5.
6. Use the Commands/Navigators in servlets, enterprise beans, Procedural Adapter Objects and applications. Please see the EAB documentation for a detailed description of the different scenarios.

Steps one to three can be accomplished with the Access Builder for SAP R/3, steps four and five with the EAB tools. When creating EAB Commands you have to specify the following objects:

- ConnectionSpec (optional)
- InteractionSpec
- Input and Output beans

ConnectionSpec
The class com.ibm.connector.sap.SAPConnectionSpec implements the CCF Connector for SAP R/3 specific connection spec and specifies the target SAP R/3 system.
Please note that in general you should not have to provide this information at

design time of the EAB Command. Instead the deployer of the EAB Command or the object in which the EAB Command is contained should do this job. The component server (e.g. WebSphere Application Server Enterprise Edition) provides tools which can be used to define the settings for a respective object running in the server.
When providing a SAPConnectionSpec you don't have to initialize all properties. Depending on the target SAP R/3 system only the required properties must be set. The simplest way to connect to an SAP R/3 system is by just specifying host name and system number:

Variable	Value (example)	Description
hostName	9.7.12.7	Hostname of the SAP R/3 system to which the EAB Commands should connect.
systemNo	0	System number of the SAP R/3 system to which the EAB Commands should connect.
rfcMode	3	The EAB Commands should connect to an SAP R/3 system.

An alternative to the settings above is to prepare a load-balanced SAP R/3 connection. This requires some more information about the SAP R/3 system. Please note that the load balancing happens on the SAP R/3 server side and has nothing to do with the load balancing feature of the application server (if available) where the EAB Command is deployed.

Variable	Value (example)	Description
loadBalancing	true	The EAB Commands should use load balancing.
systemName	DIP	System name (system ID) of the SAP R/3 system to which the EAB Commands should connect.
msgServer	pswdf028.wdf.sap-ag.de	Message server of the SAP R/3 system to which the EAB Commands should connect.
groupName	PUBLIC	Group name of the SAP R/3 system to which the EAB Commands should connect.
rfcMode	3	The EAB Commands should connect to an SAP R/3 system.

InteractionSpec
The class com.ibm.connector.sap.SAPInteractionSpec implements the CCF Connector for SAP R/3 specific interaction spec. In contrast to other connectors, you don't have to specify any properties in the interaction spec, since the input beans contain all necessary information for the interaction with the SAP R/3 system.

Input and Output beans
When creating an EAB Command you have to provide input and output beans. In the CCF Connector for SAP R/3 case these beans have to be of the same type. The

beans must implement the com.ibm.sap.bapi.GeneratedRfcCommand interface. Although it is possible to code these beans manually, the recommended way is to generate proxy beans for RFC modules with the Access Builder for SAP R/3. Those proxy beans are wrapper classes for one single remote procedure call (either RFC or BAPI) to the SAP R/3 system. They contain all the meta information required to execute a call on the SAP R/3 system. Additionally, they act as a container for all input and output parameters of the call.
Please note: Since the input/output beans don't implement the IBuffer interface please uncheck the respective box in the Command Create Guide.

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise Access Builder for Transactions
Proxy beans for RFC modules

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules
Generating documentation for SAP Business Objects and RFC modules
Setting up the CCF infrastructure

RELATED REFERENCES

Package com.ibm.connector.sap
BO and RFC proxy beans

Using enterprise bean proxies for SAP Business Objects

As described in the CCF Connector for SAP R/3 chapter Access Builder for SAP R/3 allows you to directly generate enterprise bean proxies for SAP Business Objects. These proxies are stateful Session EJB beans and they can be deployed on any enterprise beans server (EJS) providing support for stateful Session EJB beans and an implementation of the CCF infrastructure interfaces.

When deploying these generated EJB beans you have to ensure that the state management attribute in the EJB beans deployment descriptor is set to stateful. In addition you have to specify several environment variables in the deployment descriptor:

- The fully qualified classname of a factory for EjbTables (see the paragraph below). For WebSphere Application Server Advanced Edition you must set the variable ejbTableHomeFactoryClassName to com.ibm.sap.bapi.ejb.factory.EjbTableHomeFactoryTransarc, for WebSphere Application Server Enterprise Edition this variable must be set to com.ibm.sap.bapi.ejb.factory.EjbTableHomeFactoryCBroker.
- The classname for the class implementing the CCF ConnectionSpec interface. In most cases you will use the default implementation by setting the variable connectionSpecClassName to com.ibm.connector.sap.SAPConnectionSpec
- The required properties of the connection spec class to specify the target SAP R/3 system (see the paragraph below). The property name always starts with the string "connectionSpec" followed by a dot and then the class' real property name. All primitive datatypes are supported and can be set as normal strings, boolean values are assigned through the string representations "true" or "false".

EjbTables
Since most of the SAP Business Objects are dealing with tables you have to deploy the com.ibm.sap.bapi.ejb.EjbTable EJB bean in addition to the desired Business

Object EJB beans in most cases. Please ensure that the state management attribute in the deployment descriptor of the EjbTable EJB bean is set to stateful.
In the deployment descriptor of every business object EJB bean you have to specify the class name for a factory for EjbTables which must implement the com.ibm.sap.bapi.ejb.EjbTableHomeFactory interface. The Business Object EJB beans use the specified class to instantiate any EjbTables that are needed.

EjbUserManager
In the case your client programs are using the EjbUserManager the com.ibm.sap.bapi.ejb.EjbUserManager EJB bean has to be deployed on the EJB server. Please ensure that the state management attribute in the deployment descriptor of the EjbUserManager EJB bean is set to stateful.

Connection spec properties
The properties of the connection spec specify the target SAP R/3 system. Please note that in general you should not have to provide this information at development time. Instead the person deploying the EJB beans should specify the target SAP R/3 system.

Depending on the SAP R/3 system only the required properties have to be set. The simplest way to connect to an SAP R/3 system is by just specifying host name and system number:

Variable	Value (example)	Description
connectionSpec.hostName	9.7.12.7	Hostname of the SAP R/3 system to which the EJB bean should connect.
connectionSpec.systemNo	0	System number of the SAP R/3 system to which the Ejb bean should connect.

An alternative to the settings above is to prepare a load-balanced SAP R/3 connection. This requires some more information about the SAP R/3 system. Please note that the load balancing happens on the SAP R/3 server side.

Variable	Value (example)	Description
connectionSpec.loadBalancing	true	Hostname of the SAP R/3 system to which the EJB bean should connect.
connectionSpec.systemName	DIP	System name (system ID) of the SAP R/3 system to which the EJB bean should connect.
connectionSpec.msgServer	pswdf028.wdf.sap-ag.de	Message server of the SAP R/3 system to which the EJB bean should connect.
connectionSpec.groupName	PUBLIC	Group name of the SAP R/3 system to which the EJB bean should connect.

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules
Generating documentation for SAP Business Objects and RFC modules
Setting up the CCF infrastructure

RELATED REFERENCES

Package com.ibm.connector.sap
Package com.ibm.sap.bapi.ejb
Package com.ibm.sap.bapi.ejb.factory
BO and RFC Proxy Beans

Deploying EJB beans into WepSphere Application Server Enterprise Edition 3.02

This section describes the steps to deploy enterprise beans into Websphere Application Server Enterprise Edition (WAS EE). You should be familiar with the basic concepts of WAS EE. Please see the respective parts of the WebSphere Application Server Enterprise Edition documentation.

The following steps have to be performed to deploy and install SAP Business Objects enterprise bean proxies in WAS EE:

1. Ensure that the following WebSphere Application Server Enterprise Edition components are installed and configured properly:
   - Component Broker Server
   - Java Client
   - Java Client SDK
   - Server SDK
   - CICS/IMS Application Adaptor
   - CICS/IMS Application Adaptor Client SDK
2. Generate the enterprise bean proxies (Session EJB beans) for the desired SAP R/3 Business Objects with Access Builder for SAP R/3.
3. Create the EJB-JAR containing the desired SAP Business Objects enterprise bean proxies in VisualAge for Java.
   - Package the generated enterprise beans in an EJB group.
   - The com.ibm.sap.bapi.ejb.EjbTable enterprise bean (Project `Connector for SAP R/3`) has to be added to the EJB group if the proxy beans are using SAP tables (most of them do...)
   - The following properties have to be set in the deployment descriptor of each enterprise bean proxy and the EjbTable bean:
     – The Transaction Attribute has to be TX_NOT_SUPPORTED
     – The State Management Attribute has to be #STATEFUL
   - Several variables have to be set in the deployment descriptor environment of each enterprise bean proxy. The ejbTableHomeFactoryClassName, connectionSpecClassName and the necessary connectionSpec properties are required.

Variable	Value
ejbTableHomeFactoryClassName	com.ibm.sap.bapi.ejb.factory. EjbTableHomeFactoryCBroker

connectionSpecClassName	The classname of the ConnectionSpec implementation for the CCF (normally com.ibm.connector.sap.SAPConnectionSpec).
connectionSpec.hostName	Hostname of the SAP R/3 system to which the EJB bean should connect
connectionSpec.systemNo	System number of the SAP R/3 system to which the EJB bean should connect

or alternatively with load balancing functionality:

Variable	Value
ejbTableHomeFactoryClassName	com.ibm.sap.bapi.ejb. factory.EjbTableHomeFactoryCBroker
connectionSpecClassName	The classname of the ConnectionSpec implementation for the CCF (normally com.ibm.connector.sap.SAPConnectionSpec).
connectionSpec.loadBalancing	true
connectionSpec.systemName	System name (system ID) of the SAP R/3 system to which the EJB bean should connect
connectionSpec.msgServer	Message server of the SAP R/3 system to which the EJB bean should connect
connectionSpec.groupName	Group name of the SAP R/3 system to which the EJB bean should connect

4. Export the EJB group as EJB-JAR file into an empty directory. In the remainder of this chapter the EJB-JAR file name will be referenced as EJB-JAR.
5. Ensure that the following classes are in the CLASSPATH system environment variable. They are needed at deployment **and** run time.
   - Access Builder for SAP R/3 Libraries (`Drive:\IBMVJava\eab\runtime30\ivjsap20.jar` where `Drive:\IBMVJava` is the location of your VisualAge for Java installation)
   - Infobus (`Drive:\IBMVJava\eab\runtime30\infobus.jar`)
   - The enterprise bean proxies and helper classes generated with the Access Builder for SAP R/3 (in step 2).
   - The Common Connector Framework (CCF)
6. Generate and build the deployment code using the **cbejb** tool:
   - Open a comand prompt
   - Change to the directory that contains the EJB-JAR file.
   - Run the **cbejb** tool with the commandline option -ccf on the EJB JAR file (this may take a while...):

     ```
     cbejb -ccf EJB-JAR
     ```
7. Load the application into the System Manager. For a detailed description see the WebSphere Application Server Enterprise Edition: System Administration Guide.
   - Start the System Manager User Interface.
   - Change the View Level to Control.
   - Select **Load Application** from the pop-up menu of the desired host in the Host Images folder.
   - Select the generated EJB-JARFamily.ddl file, for example Working\Nt\PRODUCTION\EJB-JARFamily.ddl

- Click OK to load the application family.

8. **Optional:** Create an application server / a server group in the System Manager. For a detailed description see the WebSphere Application Server Enterprise Edition: System Administration Guide.
   - Select **Tasks -> Create Server** in the System Manager User Interface.
   - Select the desired Zone, Configuration and Server / Server group
   - Type in the name of the server to create, click **Add**
   - Click **Finish**
9. Configure the application on a server / server group in the System Manager. For a detailed description see the WebSphere Application Server Enterprise Edition: System Administration Guide.
   - Select **Tasks -> Configure Server** in the System Manager User Interface.
   - Add the enterprise bean proxies to the Applications to Configure window.
   - Add the EjbTableApp enterprise bean to the Applications to Configure window (this is only required if the enterprise bean proxies are using SAP tables).
   - Select the Zone, the Configuration and the Server / Server Group on which the application should be configured
   - Click **Finish**
10. Activate the configuration selected in the last step in the System Manager. For a detailed description see the WebSphere Application Server Enterprise Edition: System Administration Guide.
    - In the System Manager User Interface expand the **Management Zones > zone > Configurations** where zone is the Zone selected in the last step.
    - Select **Activate** from the pop-up menu of the configuration on which you configured the enterprise bean proxies.
    - Wait until activation has completed.
11. Bind the JNDI names of the enterprise bean proxies into the Component Broker namespace.
    - Open a command prompt and change to the directory containing the EJB-JAR file.
    - Bind the JNDI name of each enterprise bean proxy in the Component Broker namespace:

      ```
      ejbbind EJB-JAR package.boEjb
      ```

      where package.boEjb is the fully qualified name of the enterprise bean proxy
    - Bind the JNDI name of the EjbTable enterprise bean in the Component Broker namespace (this is only required if the enterprise bean proxies are using SAP tables):

      ```
      ejbbind EJB-JAR com.ibm.sap.bapi.ejb.EjbTable
      ```
12. Ensure that the Connector for SAP R/3 dynamic link libraries are in the library path. You find the libraries for the respective platforms in the following directories:
    - AIX
      - Drive:\IBMVJava\eab\lib\libivjsid20.so
      - Drive:\IBMVJava\eab\lib\libivjsij20.so
    - Solaris
      - Drive:\IBMVJava\eab\solaris\libivjsid20.so

- Drive:\IBMVJava\eab\solaris\libivjsij20.so
- Windows NT
  - Drive:\IBMVJava\eab\bin\ivjsij20.dll
  - Drive:\IBMVJava\eab\bin\ivjsid20.dll

Please also add the respective SAP RFC library to the library path. For Windows NT this DLL is named librfc32.dll and can be found in `Drive:\IBMVJava\eab\bin\`. For the other platforms please refer to the documentation of the SAP client.

13. The application is now ready to serve client requests.

Use the Component Broker Session Service in your client application to group several BAPI calls into one transaction. For a detailed description of the Session Service see the WebSphere Application Server Enterprise Edition: Advanced Programming Guide.

- Use the com.ibm.CBCUtil.CBSeriesGlobal.Initialize(args) method to initialize the Session Service.
- Use the following sequence to get the current session object:

```
org.omg.CORBA.Object orbCurrent =
com.ibm.CBCUtil.CBSeriesGlobal.orb().resolve_initial_references("ISessions::Current");


com.ibm.ISessions.Current sessionCurrent = com.ibm.ISessions.CurrentHelper.narrow(orbCurrent);
```

- Use the following statement to begin a session:

```
sessionCurrent.beginSession("application name");
```

- Use the following statement to end a session:

```
sessionCurrent.endSession(Mode ,true);
```

where Mode is com.ibm.ISessions.EndMode.EndModeCheckPoint to commit the changes made in this session, or com.ibm.ISessions.EndMode.EndModeReset to rollback the changes made in this session.

Before running your client application:

- Ensure that the following classes are in the classpath.
  - Access Builder for SAP R/3 Libraries (Drive:\IBMVJava\eab\runtime30\ivjsap20.jar where Drive:\IBMVJava is the location of your VisualAge for Java installation)
  - Infobus (Drive:\IBMVJava\eab\runtime30\infobus.jar)
  - The enterprise bean proxies and helper classes generated with the Access Builder for SAP R/3 (in step 2 above).
  - The jar files containing the stubs for the generated enterprise bean proxies. (They are located in the Production subdirectory of the directory you specified in step 6 above, e.g. Working\Nt\PRODUCTION*Client.jar)
  - The somojor.zip file provided by ComponentBroker (Drive:\CBroker\lib\somojor.zip, where Drive:\CBroker is the location of your Component Broker installation)
- Ensure that ioser.dll from Drive:\CBroker\bin is in the library path.
- Ensure that the Connector for SAP R/3 dynamic link libraries are in the library path. You find the libraries for the respective platforms in the following directories:
  - AIX
    - Drive:\IBMVJava\eab\lib\libivjsid20.so
    - Drive:\IBMVJava\eab\lib\libivjsij20.so

– Solaris
  - Drive:\IBMVJava\eab\solaris\libivjsid20.so
  - Drive:\IBMVJava\eab\solaris\libivjsij20.so
– Windows NT
  - Drive:\IBMVJava\eab\bin\ivjsij20.dll
  - Drive:\IBMVJava\eab\bin\ivjsid20.dll

Please also add the respective SAP RFC library to the library path. For Windows NT this library is named librfc32.dll and can be found in `Drive:\IBMVJava\eab\bin\`. For the other platforms please refer to the documentation of the SAP client.

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

RELATED TASKS

Setting up the CCF infrastructure
Generating proxy beans for SAP Business Objects and RFC modules
Deploying your application

Chapter 11. Running and deploying your application

Running your application

Before you can run your application within VisualAge for Java you must add the middleware library to the class path of your application class.

Setup for JNI middleware
VisualAge for Java already computed the class path for you. But it missed the required projects for the middleware library because these classes are **dynamically** loaded when you set the middleware type in the factory manager.

To add the required projects for the JNI middleware type you perform the following steps

1. From the Workbench, select your application or applet class.
2. Select the **Properties** item from the pop-up menu.
3. In the Properties window, select the `Class Path` pane.
4. Click the **Edit** button beside the project path entry field.
5. In the `Class Path` window, add the following projects and click **OK**.
   - IBM Access Builder for SAP R3 Libraries
   - Infobus
   - Netscape Security
6. Select **Save in repository** to save these settings also in the repository.
7. Click **OK** to save these properties.

Setup for JRFC middleware
If you have installed the client part of JRFC middleware from SAP you can easily use it from within VisualAge for Java. Instead of importing the JRFC client classes, you explicitly add the external path to the classpath settings of your application.

1. From the Workbench, select your application or applet class.
2. Select the **Properties** item from the pop-up menu.
3. In the Properties window, select the `Class Path` pane.
4. Click the **Edit** button beside the project path entry field.
5. In the `Class Path` window, add the following projects and click **OK**.
   - IBM Access Builder for SAP R3 Libraries
   - Infobus
   - Netscape Security
6. Select **Extra directories path**
7. Click the **Edit** button beside the extra directory path
8. In the extra class path window, click **Add Directory**
9. In the Select a directory window, locate the classes directory of the JRFC Client installation directory and click **OK** twice to close both windows.
10. Select **Save in repository** to save these settings also in the repository.
11. Click **OK** to save these properties.

Adding middleware information to the command line
If you use the MiddleWareInfo(String[] args) constructor and simply pass the middleware information from the command line you update the command line with the following steps:

1. From the Workbench, select your application or applet class.
2. Select **Properties** from the pop-up menu.
3. In the Properties window, select the `Program` pane.
4. In the **Command line arguments** entry field add the middleware type information
   - for JNI middleware
     `-conn JNI`
   - for JRFC middleware
     `-conn "Orbix jrfc-server"`
5. Select **Save in repository** to save these settings also in the repository.
6. Click **OK** to save these properties.

RELATED CONCEPTS

Common RFC Interface for Java

RELATED TASKS

Connecting to the SAP R/3 system
Integrating the Logon bean into your application
Deploying your application
Tracing the SAP R/3 connection of your application

Deploying your application

VisualAge for Java is a development environment without source and binary files. Before you can run your application stand-alone you have to export your code and the generated proxy beans to the file system.

The run-time libraries are available as JAR files and are also located in the eab\runtime30 subdirectory of VisualAge for Java.

ivjsap35.jar
Connector for SAP R/3 run-time libraries including
- Base Connector for SAP R/3
- CCF Connector for SAP R/3
- RFC Server for SAP R/3
- Utility Classes (Logon, Helpvalues, ...)

infobus.jar
Infobus (ivjsap35.jar requires this archive).

In addition, the Logon bean (com.ibm.sap.bapi.util.logon) and the other utility classes (com.ibm.sap.bapi.util) require JFC 1.1 or higher (belonging to the Java 2 platform).

To run your application outside of VisualAge for Java, it is sufficient to specify the VisualAge for Java eab\runtime30 subdirectories in your class path. Alternatively you can specify each single compressed file in your class path.

To deploy the minimal set of required classes for a given application or applet, you may use the **Select referenced types and resources** button in the Export SmartGuide to let VisualAge for Java find all required classes. As it cannot find dynamically loaded classes, you have to specify the following factory classes to include the JNI run time:

```
com.ibm.sap.bapi.SimpleFactory
com.ibm.sap.bapi.StructureFactory
com.ibm.sap.bapi.jni.RfcConnectionFactory
com.ibm.sap.bapi.jni.RfcModuleFactory
com.ibm.sap.bapi.jni.TableFactory
```

The JNI middleware implementation also requires native run-time libraries. The native run-time library is shipped in two versions

- ivjsij35
  optimized version for deployment
- ivjsid35
  debug and trace enabled version for analyzing problems

AIX

The following run-time libraries are located in the `eab/lib` subdirectory of VisualAge for Java. Verify that this directory is included in your LIBPATH environment variable. See the file `sap.txt` in the `eab/lib` subdirectory for special notes on the AIX deployment.

- libivjsij35.so
- libivjsid35.so

LINUX

The following run-time libraries are located in the `eab/linux` subdirectory of VisualAge for Java. Verify that this directory is included in your LD_LIBRARY_PATH environment variable. See the file `sap.txt` in the `eab/linux` subdirectory for special notes on the Linux deployment.

- libivjsij35.so
- libivjsid35.so

OS/2

The following run-time libraries are located in the `eab/dll` subdirectory of VisualAge for Java. Verify that this directory is included in your LIBPATH environment variable in config.sys.

- ivjsij35.dll
- ivjsid35.dll
- librfc.dll

SOLARIS

The following run-time libraries are located in the `eab/solaris` subdirectory of VisualAge for Java. Verify that this directory is included in your LD_LIBRARY_PATH environment variable. See the file `sap.txt` in the `eab/solaris` subdirectory for special notes on the Solaris deployment.

- libivjsij35.so
- libivjsid35.so

WIN

The following run-time libraries are located in the `eab/bin` subdirectory of VisualAge for Java. Verify that this directory is included in your PATH environment variable.

- ivjsij35.dll
- ivjsid35.dll
- librfc32.dll

390

The following run-time libraries are located in the `eab/os390` subdirectory of VisualAge for Java. Verify that this directory is included in your LIBPATH environment variable.

- libivjsij35.so
- libivjsij35.x
- libivjsid35.so
- libivjsid35.x
- librfc
- librfc.x

400

The following .SAVF file containing the run-time libraries is located in the `eab/os400` subdirectory of VisualAge for Java. See the file `sap.txt` in the `eab/os400` subdirectory for special notes on the OS/400 deployment.

- IVJSIJ35.SAVF
  contains the following run-time libraries:
  - IVJSIJ35.SRVPGM
  - IVJSID35.SRVPGM
  - LIBRFC.SRVPGM

NOTE: The included RFC run-time library is property of SAP AG. You are granted usage of this library in your development environment. Deploying this required library with your application in a production environment is only valid with the sufficient amount of SAP R/3 licenses according to the terms and conditions of SAP R/3.

RELATED CONCEPTS

Common RFC Interface for Java

RELATED TASKS

Running your application
Tracing the SAP R/3 connection of your application

Running applets with JNI code in AppletViewer

The shipped middleware layer uses the Java Native Interface (JNI) to access a native run-time library on the same computer to connect to an SAP R/3 system. For security reasons, it is only possible to access native libraries through the Java Native Interface (JNI) from within Java applications. Java applets that are located in HTML pages are restricted to using local resources. However, it is possible to call a system library via JNI if a Certifying Authority (CA) signed the JAR file with all contained classes.

You have to set up the environment variables for the Java class path and the path for the native run-time library in the same way as for deploying an application.

Follow these steps to start a Java applet connecting to an SAP R/3 system with Appletviewer:

1. Create an identity <name> with a new keypair and a self-signed certificate in the default keystore

   ```
   keytool -genkey -alias <name>
           -dname "cn=<your_name>, ou=<your_org_unit>, o=<your_company>, c=<your_country>"
   ```

   You will be prompted for a password for <name> and a password for the default keystore. If the default keystore already exists enter the password you specified when creating it. Otherwise select and enter a new password for the default keystore. If <name> already exists in the default keystore enter the password you specified when creating it. Otherwise select and enter a new password.
2. Export the certificate created in the default keystore to the file <name>.x509

   ```
   keytool -export -alias <name> -rfc -file <name>.x509
   ```

   You will be prompted for the password of the default keystore.
3. Sign the archive

   ```
   jarsigner -verbose <jarname>.jar <name>
   ```

   You will be prompted for the password of the default keystore and the password for <name>.
4. Test the signed JARs using `OrderApplet.class` within an HTML page

   ```
   <APPLET CODE=com.ibm.sap.bapi.demo.salesorder.OrderApplet.class
     ARCHIVE="<path>/<jarname>.jar" WIDTH=250 HEIGHT=300></APPLET>
   ```
5. Show the HTML-page

   ```
   appletviewer mypage.html
   ```

More information about signed JARs, keytool and jarsigner can be found at the JavaSoft homepage (see link in the related concepts section below).

RELATED CONCEPTS

JDK 1.2 - Signed Applet Example (JavaSoft homepage, Internet)

RELATED TASKS

Running applets with JNI code in Netscape Communicator
Running applets with JNI code in Java Plug-In
Connecting to the SAP R/3 System
Deploying your application
Running your application
Tracing the SAP R/3 connection of your application

Running applets with JNI code in Netscape Communicator

The shipped middleware layer uses the Java Native Interface (JNI) to access a native run-time library on the same computer to connect to an SAP R/3 system. For security reasons, it is only possible to access native libraries through the Java Native Interface (JNI) from within Java applications. Java applets that are located in HTML pages are restricted to using local resources.

Follow the steps described below to run applets containing JNI code in a Netscape Communicator environment. To achieve this goal, you will need the following Netscape tools and capabilities:

- Netscape Communicator Version 4.07 or later
- Netscape Signing Tool Version 1.3
  This tool helps you to create test certificates and to sign JAR files.
- Netscape Capability API
  The program must explicitely request certain privileges (such as local access to files and dynamic link libraries) using the Netscape Capability API. This is accomplished with the run-time classes for SAP R/3.
- Netscape Object Signing
  Applets and classes requesting these privileges must be stored in signed JAR files. You can create and sign these JAR files with your own certificate or a test certificate created by the Netscape Signing Tool.

Creating a test certificate with the Netscape Signing Tool
The Netscape Signing Tool allows you to create a self-signing test certificate with a restricted period of validity. This certificate is stored inside Netscape Communicator's certificate database. As an additional feature, the Signing Tool provides a file containing X509 information about your test certificate. You can import this file into Netscape Communicator environments located on other machines.

It is advisable to use a separate set of certificate database files for your test certificates. Before copying these files the local password for Netscape communicator must be disabled. To disable the local password and create a separate set of files:

1. Start Netscape Navigator.
2. Click the **Security** toolbar button. The Security Info window is shown.
3. Select **Passwords** and click **Set Password**.
4. Remove the password.
5. Click **OK**, even if there was no password before.
6. Close the Security Info window.
7. Close Netscape Navigator.
8. Create a target directory for the certificate database files.
9. Copy the database files cert7.db and key3.db from the Netscape Communicator user directory to your target directory.
10. Enable the local Netscape Communicator password again if it was enabled before.

To create a test certificate:

1. Place the signtool.exe somewhere in your path.
2. Invoke the signtool from the target directory:
   `signtool -G"Test Certificate" -d.`
3. Pass a certificate name of your own choice with the -G option; -d. specifies the current directory as the location of the certificate database.
4. The signtool program requests you to enter some optional information about the certificate, such as the issuer of the certificate and the company. Complete that information.
5. After completion of the call, the current directory contains an import file named x509.cacert.

6. To get a list of all certificates inside the database call
   `signtool.exe -L`

Certificates that are capable of signing objects are marked with an asterisk (*).

Importing the test certificate into Netscape Communicator

1. Copy the export file x509.cacert and the following HTML file to the target machine.

   ```
   <HTML>
    <BODY>
     <a href="x509.cacert">Click Here to Import the Test Certificate</a>
    </BODY>
   </HTML>
   ```

2. Start Netscape Communicator.
3. Make sure that Netscape Communicator has a MIME type of application/x-x509-ca-cert by selecting **Edit > Preferences** from the menu and searching through the Navigator/Applications panel. If such a MIME type doesn't exist, create one with the following information:

   ```
   Description of type: X509 CA certificate
   File extension: cacert
   MIME type: application/x-x509-ca-cert
   Application to use: C:\WINNT\system32\rundll32.exe
    c:\winnt\system32\inetcpl.cpl,SiteCert_RunFromCmdLine %1
   ```

   For Windows 95 or Windows 98 replace C:\WINNT\system32\ with C:\WINDOWS\SYSTEM\.
4. Open the HTML file and activate the link.
5. You are requested to enter some information. Select **Accept this Certificate Authority for certifying software developers**.
6. Type a nickname for the test certificate.
7. After completion you can find an entry for the test certificate in the **Signers** list of the Security Info window.

Creating the Java Archive (JAR) Files

1. Create a subdirectory signdir in the target directory

   ```
   md signdir
   cd signdir
   ```

2. Export all required classes to the signdir directory
3. Sign these files and repackage them

   ```
   signtool -b"testcert" -k"Test Certificate" -Z myapplet.jar -d. signdir
   ```

   The name of the certificate (-k option) must be the same as the name in the -G option of the generation call. The -b option specifies the name of the signature files created in the META-INF directory.

Preparing the environment for Netscape Communicator

Typically, you would use a deployment mechanism like Netscape SmartUpdate to automatically distribute the applet and the binaries to the clients. For testing, you may manually prepare your Netscape browser with the following steps.

1. Copy the JAR file to the `....\Netscape\Communicator\Program\Java\classes` directory.
2. Copy the native run-time libraries to the `....\Netscape\Communicator\Program\Java\Bin` directory.

Running the applet
Now you are ready to run your applet. For example, to run the sales order sample applet:

```
<HTML>
 <BODY>
  <APPLET code=com.ibm.sap.bapi.demo.salesorder.OrderApplet.class
   archive=myapplet.jar width=750 height=550>
  </APPLET>
 </BODY>
</HTML>
```

RELATED CONCEPTS

Code Signing for Java Applets (Internet)
Netscape Signing Tool 1.3 (Internet)

RELATED TASKS

Running applets with JNI code in AppletViewer
Running applets with JNI code in Java Plug-In
Connecting to the SAP R/3 System
Deploying your application
Running your application
Tracing the SAP R/3 connection of your application

Running applets with JNI code in Java Plug-In

Java Plug-in 1.2 is a software product shipped by Sun Microsystems Inc. It enables Netscape Navigator 3.0 (or later) and Internet Explorer 3.02 (or later) to run Java 2 platform conforming applets. Java Plug-in bypasses the browser's default Java virtual machine by providing its own implementation of a Java VM which covers Java 2 platform capabilities like JNI, RMI, or the Swing classes. You must specify an applet inside the HTML file using a browser-specific notation for plug-ins. Since Java Plug-in is an OCX for Internet Explorer, use the <OBJECT ...>...</OBJECT> tag. For Netscape Navigator, use the <EMBED ...>....</EMBED> tag to activate Java-Plug-in.

The following example illustrates the use of these tags to make an applet run in both browsers with Java Plug-in 1.2. Note that the JAR file myapplet.jar must be signed with the Java 2 SDK jarsigner tool or the Netscape Signing Tool.

```
<HTML>
<BODY>
<OBJECT classid="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
 width="750" height="550">
 <PARAM NAME="code" VALUE="com.ibm.sap.bapi.demo.salesorder.OrderApplet.class">
 <PARAM NAME="archive" VALUE="myapplet.jar">
 <PARAM NAME="type" VALUE="application/x-java-applet;version=1.2.2">
 <COMMENT>
  <EMBED type="application/x-java-applet;version=1.2.2" width="750" height="550"
   code="com.ibm.sap.bapi.demo.salesorder.OrderApplet.class"
   archive="myapplet.jar">
  </EMBED>
 </COMMENT>
</OBJECT>
</BODY>
</HTML>
```

RELATED CONCEPTS

Java-Plug-in web site (Internet)

RELATED TASKS

Running applets with JNI code in AppletViewer
Running applets with JNI code in Netscape Communicator
Connecting to the SAP R/3 System
Deploying your application
Running your application
Tracing the SAP R/3 connection of your application

Chapter 12. Developing an RFC Server application - overview

There are nine steps involved in developing an RFC Server application. Some of these steps are optional.

1. **Create meta information for each RFC Server module**
2. **Add business logic for each RFC Server module**
3. **Create an RFC Server module factory**
   This is an optional step which allows the RFC Server environment to create an instance of a module only when it is needed.
4. **Add transaction management**
   This step is only required if your RFC Server should manage transactional RFC requests.
5. **Add security management**
   This step is only required if you want to restrict access to your RFC Server.
6. **Configure the RFC Server properly**
7. **Create an RFC Server object**
8. **Create an idle event listener**
   This step is optional.
9. **Complete the RFC Server application by starting the RFC Server**

RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

Configuring SAP R/3 to call an RFC Server application from ABAP/4
Creating an ABAP/4 Report

Creating meta information for an RFC Server module

The meta information contains the name of the RFC Server module as well as a textual description and additional information about import, export, and table parameters. To provide this meta information you have to implement the interface IRfcServerModuleInfo. It comprises the following methods:

`String getRfcModuleName()`
returns a unique name of the module

`String getRfcModuleDescription()`
returns a description of the module

`RfcModuleParameterInfo getRfcModuleParameterInfo()`
returns information about import, export, and table parameters

RfcModuleParameterInfo is a helper class whose objects hold arrays of

- IFieldParamInfo for import parameters
- IFieldParamInfo for export parameters
- ComplexParamInfo for table parameters

An object of type IFieldParamInfo can be a SimpleParamInfo or a ComplexParamInfo, since a parameter can be a simple or a structured field. The

RFC Server run time can query these arrays, and it is your task to assemble the parameter information as demonstrated in the following code snippet:

```
/**
 * Implementation of IRfcServerModuleInfo containing two simple import parameter, one
 * structured export parameter and one table parameter.
 */
import com.sap.rfc.*;
import com.ibm.sap.bapi.rfcserver.*;
public class MyRfcServerModuleInfo implements com.ibm.sap.bapi.rfcserver.IRfcServerModuleInfo
{
   // standard constructor
   public MyRfcServerModuleInfo() {
    super();
   }
   // returns module description
   public String getRfcModuleDescription() {
    return "A simple RFC Server module";
   }
   // returns module name
   public String getRfcModuleName() {
    return "MY_RFC_SERVER_MODULE";
   }
   // returns parameter information
   public com.ibm.sap.bapi.rfcserver.RfcModuleParameterInfo getRfcModuleParameterInfo()
   {
    /* parameter information for first import parameter NAME */
    SimpleParamInfo imp1 =
     new SimpleParamInfo("NAME", "F_NAME", IFieldInfo.RFCTYPE_CHAR, 32, 0);
    /* parameter information for second import parameter AGE */
    SimpleParamInfo imp2 =
     new SimpleParamInfo("AGE","F_AGE", IFieldInfo.RFCTYPE_INT, 0, 0);
    /* array of simple field infos used construct a complex info for
     an export parameter */
    IFieldInfo[] fieldInfosExp = {
     new SimpleInfo("F_ERRCODE", IFieldInfo.RFCTYPE_NUM, 6, 0),
     new SimpleInfo("F_ERRTEXT", IFieldInfo.RFCTYPE_CHAR, 128, 0)
    };
    /* array of simple field infos used construct a complex info for
     a table parameter */
    IFieldInfo[] fieldInfosTab = {
     new SimpleInfo("F_EMAIL", IFieldInfo.RFCTYPE_CHAR, 48, 0),
    };
    ComplexParamInfo exp1 =
     new ComplexParamInfo("RETURN", fieldInfosExp, "RETURN_FORMAT");
    ComplexParamInfo tab1 =
     new ComplexParamInfo("DATA", fieldInfosTab, "DATA_FORMAT");
    return new RfcModuleParameterInfo(
        new IFieldParamInfo[]{imp1, imp2},
        new IFieldParamInfo[]{exp1},
        new ComplexParamInfo[]{tab1}
    );
   }
}
```

RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

Developing an RFC Server application
Configuring SAP R/3 to call an RFC Server application from ABAP/4
Creating an ABAP/4 Report

Adding business logic

You can add business logic to an RFC Server module by implementing the interface IRfcServerModule which consists of the following two methods:

void execute(IImpExpParam[] importParams, IImpExpParam[] exportParams, ITable[] tableParams)
: Place your application logic here. You can modify export and table parameters or throw an exception of type JRfcRfcAbapException in case of an error.

IRfcServerModuleInfo getRfcModuleInfo()
: the RFC Server run time calls this method to retrieve information about your server module. Return a reference to a valid instance of IRfcServerModuleInfo.

Within the execute() method you can access the current server thread as an instance of IRfcServerThread by performing the cast (IRfcServerThread)Thread.currentThread(). This may be useful since IRfcServerThread offers some methods which allow access to the RFC Server environment.

RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

Developing an RFC Server application
Configuring SAP R/3 to call an RFC Server application from ABAP/4
Creating an ABAP/4 Report

Creating an RFC Server module factory

There are two ways to add modules to an RFC Server: adding an instance of your RFC Server module class or adding an object which implements the interface IRfcServerModuleFactory. The second way allows the RFC Server environment to create an instance of a module only when it is needed. This is an advantage if the server contains a large number of modules. There is only one method you have to implement:

IRfcServerModule createRfcServerModule(IRfcServerModuleFactoryCreateParam createParam)
: This method returns an RFC Server module which depends on a parameter of type IRfcServerModuleFactoryCreateParam. It's up to you to implement this parameter interface.

RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

Developing an RFC Server application
Configuring SAP R/3 to call an RFC Server application from ABAP/4
Creating an ABAP/4 Report

Adding transaction management

This task must be carried out if your RFC Server should manage transactional RFC requests. The RFC Server run-time environment delegates transactional calls to your implementation of IRfcServerTransactionManager. Remember: a transaction which an RFC client (ABAP/4 program) invokes usually consists of several calls. The call to your module is only one part of the transaction and should only be executed once within this transaction.

The following methods must be implemented:

int checkTransactionId(String transactionId):
This is the first call of a transactional call sequence. It enables you to check if your part of the current transaction is already done or not. Therefore you must store the transaction ID (for example in a file or database). When this method has been entered it is your first task to look for an entry in the transaction ID storage. If there is one available (which means that your module has already been called) leave the method returning the value 1. The RFC run time will then skip the request. If no entry was found (which means that this is the first call to your module in the current transaction) save the transaction ID and return 0. In case of an error return -1. Note: if the current transaction contains more than one call to your RFC Server this method will only be called once.

beforeExecution(String transactionId, int transactionSequenceNo, IRfcServerModule rfcServerModule)
This method is called just before your business logic is executed for each call to a module within your RFC Server. You can place code there to save information about the state of your application in order to process following rollback requests. The parameter transactionSequenceNo contains a unique identifier of the current call within the transaction.

int commit(String transactionId)
Enableless you to commit changes to resources associated with the current transaction. Return 0 if ok or -1 in case of an error.

int rollback(String transactionId)
Enables you to rollback changes to resources associated with the current transaction. Return 0 if ok or -1 in case of an error.

int confirmTransactionId(String transactionId)
Notifies you that the transaction is completed (successfully or not). This is the place where you can remove the transaction ID from storage.

RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

Developing an RFC Server application
Configuring SAP R/3 to call an RFC Server application from ABAP/4
Creating an ABAP/4 Report

Adding security management

This is an optional task and must be carried out if you want to restrict access to your RFC Server. The implementation of the interface IRfcServerSecurityManager contains only one method:

boolean checkAuthorization(RfcAttributeInfo rfcAttributeInfo, IRfcServerModuleInfo rfcServerModuleInfo)

You can retrieve information about the current RFC request (for example user name, client, system name,...) from the RfcAttributeInfo parameter. This information is the starting point of your decision whether the given RFC Server module can be executed or not. Returning the value `false` refuses execution and raises an exception which the client ca catch. The value `true` enables the execution of the module.

There is another way to check authorization even without adding a security manager. You can retrieve an RfcAttributeInfo object from the current IRfcServerThread which can be accessed as described in the topic ″Adding business bogic″. This allows you to include the parameter information passed to the module into your decision whether to pass or refuse the call.

RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

Developing an RFC Server application
Configuring SAP R/3 to call an RFC Server application from ABAP/4
Creating an ABAP/4 Report

Configuring an RFC Server

For an RFC Server to run properly it has to be configured. Configuration data is passed to the RFC Server as an RfcServerInfo object. There are two ways to configure an RFC Server:

Using the saprfc.ini file

To use this file you set the `destination` property in RfcServerInfo. The underlying RFC run time searches the saprfc.ini file for a corresponding entry and uses the specified configuration data to accept client connections.

Setting configuration properties in your program

Instead of using saprfc.ini you can set the following attributes in RfcServerInfo:

- programId, name under which a client can access your RFC Server
- gatewayHost, host name of the SAP gateway
- gatewayService, service of the SAP gateway (for example sapgw00)
- traceMode, switches the trace of the RFC run time on or off

RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

Developing an RFC Server application
Configuring SAP R/3 to call an RFC Server application from ABAP/4
Creating an ABAP/4 Report

Creating an RFC Server object

An RfcServer object can be constructed by passing an RfcServerInfo object as well as (if necessary) your implementations of IRfcServerTransactionManager or IRfcServerSecurityManager or both to the appropriate constructor. You can also call the standard constructor and set these three attributes afterwards.

The next step is to add your modules to the RfcServer object either as instances or factories. To achieve this you have to call one of the addRfcServerModule(...) methods:

- void addRfcServerModule(IRfcServerModule mod)
- void addRfcServerModule(IRfcServerModule mod, boolean checkAuth)
- void addRfcServerModule(IRfcServerModuleInfo info, IRfcServerModuleFactory factory, IRfcServerModuleFactoryCreateParam param)
- void addRfcServerModule(IRfcServerModuleInfo info, IRfcServerModuleFactory factory, IRfcServerModuleFactoryCreateParam param, boolean checkAuth)

The boolean parameter checkAuth = false allows you to skip authorization checks for the specified module if a security manager is installed. If not this parameter is ignored.

RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

Developing an RFC Server application
Configuring SAP R/3 to call an RFC Server application from ABAP/4
Creating an ABAP/4 Report

Creating an idle event listener

An idle event listener is an implementation of the interface IdleEventListener which can be registered with an RFC Server. It will be notified whenever a server thread returns from waiting for RFC requests without having received one. An idle event listener can be added either to your RfcServer or to an IRfcServerThread object which can be retrieved in the execute() method as described in ″Adding business logic″.

In the first case the listener receives idle time events from each IRfcServerThread, in the second case only events belonging to a specific thread are received. In either case use the addIdleEventListener() method to add the idle event listener. The RfcServer object inernally adds the listener to each thread.

You can specify a value for idle time as a property of both the RfcServer and the IRfcServerThread object. The default value is -1, which means an infinite idle time.

RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

Developing an RFC Server application
Configuring SAP R/3 to call an RFC Server application from ABAP/4
Creating an ABAP/4 Report

Completing the RFC Server application

Up to now you have a well configured RfcServer object which does nothing. In the last step you simply have to call the start(int numThreads) method of the RfcServer object. It needs one parameter: the number of concurrently running threads.

After the server has started the properties rfcServerInfo, transactionManager and securityManager can no longer be changed. It is not possible to add new modules to a running RFC server either.

RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

Developing an RFC Server application
Configuring SAP R/3 to call an RFC Server application from ABAP/4
Creating an ABAP/4 Report

Chapter 13. Performing specific tasks

Generating proxies from command line

To generate Java proxy beans the Access Builder for SAP R/3 uses the Generator Framework for SAP. This Generator Framework also has a command line interface which can be used to generate Java proxy beans without using the Access Builder for SAP R/3 GUI. This is especially interesting in the following cases:

- To generate proxies on a platform where the Access Builder for SAP R/3 GUI tool is not supported
- To integrate proxy generating into an automatic build process

Setup
To be able to use the Generator Framework for SAP you have to setup the following requirements:

- Install a Java 2 SDK or Java 2 Runtime Environment respectively
- Add the following JAR files to your Java classpath:
  - ivjsgn35.jar
  - ivjsap35.jar
  - infobus.jar
- When generating EJB beans proxies then additionally the classes of the packages
  - javax.ejb
  - javax.ejb.deployment

  have to be in the Java classpath

Choose the right generator
For each output type there is one particular generator. After you have decided which type of proxy you need you can choose the generator accordingly. The following table helps you finding out which generator to use for what:

Proxy beans for Business Objects	com.ibm.sap.bapi.generator.GeneratorBo
EJB beans proxies for business objects	com.ibm.sap.bapi.generator.GeneratorSessionEjb
Proxy beans for RFC modules	com.ibm.sap.bapi.generator.GeneratorRfcModule

Generating proxy beans for Business Objects
To generate proxy beans for Business Objects you have to specify some parameters. The parameters are the same for both proxy beans for Business Objects and EJB beans proxies for Business Objects. You can see the list of expected parameters as displayed below in the left column by invoking either com.ibm.sap.bapi.generator.GeneratorBo or com.ibm.sap.bapi.generator.GeneratorSessionEjb without parameters:

```
java com.ibm.sap.bapi.generator.GeneratorBo
```

or

```
java com.ibm.sap.bapi.generator.GeneratorSessionEjb
```

The following table explains the parameters and the expected values:

Parameter	Explanation
-sapobjecttype <sapobjecttype>	Replace <sapobjecttype> with the type name of the SAP Business Object you want to create a proxy for. **Example:** -sapobjecttype BUS0002
-sapobjectname <sapobjectname>	Replace <sapobjectname> with the name of the SAP Business Object you want to create a proxy for. **Example:** -sapobjecttype CompanyCode
-embedsapdoc <true/false>	Replace <true/false> with either *true* or *false*. Specifying *true* means that in addition to retrieving meta information about a specific type the corresponding documentation is retrieved from SAP R/3 and stored locally. This is usually not necessary when generating Java proxies. **Example:** -embedsapdoc false
-packagename <java.package.name>	Replace <java.package.name> with the name of the Java package which is going to contain the Java classes. **Example:** -packagename foo.sap.bapi.generated
-userinfo <client> <username> <password> <language>	Replace the variables <client>, <username>, <password>, and <language> with the user logon information of your SAP R/3 system: • Replace <client> with your SAP R/3 client number (e.g. *800*) • Replace <username> with your SAP R/3 user id • Replace <password> with the password of the user id • Replace <language> with a language string valid for your SAP R/3 system (e.g. *E*) **Example:** -userinfo 800 muster ides E
-connectinfo <hostname> <systemno>	Replace the variables <hostname> and <systemno> with the system logon information of your SAP R/3 system: • Replace <hostname> with the IP address or hostname of your SAP R/3 system • Replace <systemno> with the number of the SAP R/3 instance you want to connect to **Example:** -connectinfo sapr3.boeblingen.de.ibm.com 0

Parameter	Explanation
-useserialized <true/false>	Replace <true/false> with either *true* or *false*. Specifying *true* means that a connection to SAP R/3 is only established if the meta information for the item to generate cannot be found on the local harddisk. Usually this variable will be set to *true* to avoid unnecessary, time consuming calls to SAP R/3. If the meta information in SAP R/3 has been changed then this variable must be set to *false* to refresh the meta information stored locally. **Example:** -useserialized true
-templatepath <input_path_for_templates>	Replace <input_path_for_templates> with the name of the path where the template files used for Java proxy generating are stored. In VisualAge for Java the template files are located under the VisualAge for Java product directory in the path *\ide\tools\com-ibm-sap-bapi\sys* **Example:** -templatepath d:\ibmvjava\ide\tools\com-ibm-sap-bapi\sys
-serializedpath <input_path_for_serialized_beans>	Replace <input_path_for_serialized_beans> with the name of the path where you want to put the serialized meta information. **Example:** -serializedpath f:\sapgen\ser\
-generatepath <output_path_for_generated_stuff>	Replace <output_path_for_generated_stuff> with the name of the path where you want to put the generated Java proxy files. **Example:** -generatepath f:\sapgen\java\

An actual call to create a proxy bean could look like this:

```
java com.ibm.sap.bapi.generator.GeneratorBo
     -sapobjecttype BUS0002 -sapobjectname CompanyCode
     -embedsapdoc false -packagename foo.sap.bapi.generated
     -userinfo 800 muster ab30 en
     -connectinfo cbsapsrv.boeblingen.de.ibm.com 0
     -useserialized true -templatepath f:\sapgen\sys\
     -serializedpath f:\sapgen\ser\
     -generatepath f:\sapgen\java\bo\
```

An actual call to create an EJB bean proxy could look like this:

```
java com.ibm.sap.bapi.generator.GeneratorSessionEjb
     -sapobjecttype BUS0002 -sapobjectname CompanyCode
     -embedsapdoc false -packagename foo.sap.bapi.generated
     -userinfo 800 muster ab30 en
     -connectinfo cbsapsrv.boeblingen.de.ibm.com 0
     -useserialized true -templatepath f:\sapgen\sys\
     -serializedpath f:\sapgen\ser\
     -generatepath f:\sapgen\java\ejb\
```

Generating proxy beans for RFC modules

To generate proxy beans for RFC modules you have to specify some parameters.

You can see the list of expected parameters as displayed below in the left column by invoking com.ibm.sap.bapi.generator.GeneratorRfcModule without parameters:

```
java com.ibm.sap.bapi.generator.GeneratorRfcModule
```

The following table explains the parameters and the expected values:

Parameter	Explanation
-rfcmodulename <rfcmodulename>	Replace <rfcmodulename> with the name of the RFC you want to create a proxy for. **Example:** -sapobjecttype BUS0002
-embedsapdoc <true/false>	Replace <true/false> with either *true* or *false*. Specifying *true* means that in addition to retrieving meta information about a specific type the corresponding documentation is retrieved from SAP R/3 and stored locally. This is usually not necessary when generating Java proxies. **Example:** -embedsapdoc false
-packagename <java.package.name>	Replace <java.package.name> with the name of the Java package which is going to contain the Java classes. **Example:** -packagename foo.sap.bapi.generated
-userinfo <client> <username> <password> <language>	Replace the variables <client>, <username>, <password>, and <language> with the user logon information of your SAP R/3 system: • Replace <client> with your SAP R/3 client number (e.g. *800*) • Replace <username> with your SAP R/3 user id • Replace <password> with the password of the user id • Replace <language> with a language string valid for your SAP R/3 system (e.g. *E*) **Example:** -userinfo 800 muster ides E
-connectinfo <hostname> <systemno>	Replace the variables <hostname> and <systemno> with the system logon information of your SAP R/3 system: • Replace <hostname> with the IP address or hostname of your SAP R/3 system • Replace <systemno> with the number of the SAP R/3 instance you want to connect to **Example:** -connectinfo sapr3.boeblingen.de.ibm.com 0

Parameter	Explanation
-useserialized <true/false>	Replace <true/false> with either *true* or *false*. Specifying *true* means that a connection to SAP R/3 is only established if the meta information for the item to generate cannot be found on the local harddisk. Usually this variable will be set to *true* to avoid unnecessary, time consuming calls to SAP R/3. If the meta information in SAP R/3 has been changed then this variable must be set to *false* to refresh the meta information stored locally. **Example:** -useserialized true
-templatepath <input_path_for_templates>	Replace <input_path_for_templates> with the name of the path where the template files used for Java proxy generating are stored. In VisualAge for Java the template files are located under the VisualAge for Java product directory in the path *\ide\tools\com-ibm-sap-bapi\sys* **Example:** -templatepath d:\ibmvjava\ide\tools\com-ibm-sap-bapi\sys
-serializedpath <input_path_for_serialized_beans>	Replace <input_path_for_serialized_beans> with the name of the path where you want to put the serialized meta information. **Example:** -serializedpath f:\sapgen\ser\
-generatepath <output_path_for_generated_stuff>	Replace <output_path_for_generated_stuff> with the name of the path where you want to put the generated Java proxy files. **Example:** -generatepath f:\sapgen\java\

An actual call to create an RFC proxy bean could look like this:

```
java com.ibm.sap.bapi.generator.GeneratorRfcModule
     -rfcmodulename STFC_PERFORMANCE -embedsapdoc false
     -packagename foo.sap.bapi.generated
     -userinfo 800 muster ab30 en
     -connectinfo cbsapsrv.boeblingen.de.ibm.com 0
     -useserialized true
     -templatepath f:\sapgen\sys\
     -serializedpath f:\sapgen\ser\
     -generatepath f:\sapgen\java\rfc\
```

Restrictions

The command line interface of the SAP Generator Framework is not national language enabled. The only supported language is English.

RELATED CONCEPTS

Advantages of using proxy beans
Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules

Access Builder for SAP R/3
Base Connector for SAP R/3
Common RFC Interface for Java

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules
Invoking BAPIs and RFCs dynamically

RELATED REFERENCES

Naming conventions for generated classes
Naming conventions for proxy beans

Tracing the SAP R/3 connection of your application

When you develop an SAP R/3 application you may need to trace the connection to the SAP R/3 system. You can track down certain problems much easier with a trace than you can using the debugger. When you run your application outside of the VisualAge for Java Integrated Development Environment (IDE), tracing is the only way to get detailed information on the programs flow. You enable tracing in one of the following two ways:

1. Set the trace level using the method setRFCTraceLevel() defined by com.sap.rfc.IRfcConnection. This is the easy way and is not middleware dependent. Please note that you have to call setRFCTraceLevel() before opening the connection to make the trace work.
2. If you use JNI middleware you gain full tracing flexibility using the JniSettings class. Please read the section below to figure out how this can be accomplished.

Specifing trace options using JniSettings
The class JniSettings takes the following command line arguments:

```
[-tracefile      file]
[-alloctracefile file]
[-rfctracemode   {0 | 1} ]
[-sapgui         {0 | 1 | 2} ]
[-abapdebug      {0 | 1}   ]
or
-reset
```

The parameters have the following meaning

-tracefile file
Defines the fully qualified file name of the trace file. If the path information is omitted it is stored in the current directory. By default, no trace file is written.

-alloctracefile file
Defines the fully qualified file name of the allocation trace file. If the path information is omitted it is stored in the current directory. By default, no allocation trace file is written.

-rfctracemode mode
This option controls the RFC tracing of the SAP R/3 application server. Valid values are 0 or 1. The meanings are

- 0 = RFC Trace Off (default)
- 1 = RFC Trace On

-sapgui
This option controls the use of SAPGUI between RFC calls. Valid values are 0, 1 or 2. The meanings are

- 0 = without SAPGUI (default)
- 1 = with visible SAPGUI
- 2 = with invisible SAPGUI

-abapdebug
This option controls the use of the ABAP debugging facility for the invoked RFC modules. Valid values are 0 or 1:

- 0 = without ABAP/4 debugger (default)
- 1 = with ABAP/4 debugger

-reset This option resets all settings to default values.

Starting JniSettings in VisualAge for Java

1. In the Workbench, navigate to the com.ibm.sap.bapi.jni package. You can find this package in the IBM Access Builder for SAP R3 Libraries project.
2. Set the command line arguments of JniSettings in the Properties window of the class.
3. Select the JniSettings class and click **Run**.
4. Copy the JNISETTINGS.SER file to the resource directory of your project
   In VisualAge for Java each project has its own resource directory. The configuration of JniSettings is stored in a file called JNISETTINGS.SER. This file can be found in the root of the resource directory. When running JniSettings in VisualAge for Java the JNISETTINGS.SER file is created and stored in the resource directory of the IBM Access Builder for SAP R3 Libraries project. Your application or applet will use a JNISETTINGS.SER file located in your project resource directory.

Starting JniSettings from the command line
Switch to the directory from which you run your application and type the following command:

```
java com.ibm.sap.bapi.jni.JniSettings <options>
```

You will not have to copy any files in this case.

General information on trace files
By default the native library ivjsij35.dll (or libivjsij35.so on Unix systems) is used. For performance reasons this is a performance optimized library that does not contain any trace code. When you enable one of the trace file options the native library ivjsid35.dll (or libivjsid35.so on Unix systems) is used instead.

The trace files are written from scratch each time the JNI native library is reloaded. This typically occurs when you restart the Java application.

The tracefile contains the flow of execution inside the JNI native library. Import and export parameters are shown in hexdump format.
Warning: This file may get very big!

The allocation tracefile contains very low-level information and should only be specified when a strange error has to be analyzed.

RELATED TASKS

Using the proxy beans to access SAP Business Objects
Calling RFC modules
Running your application
Running applets with JNI code in AppletViewer
Running applets with JNI code in Netscape Communicator
Running applets with JNI code in Java Plug-In
Deploying your application

Configuring SAP R/3 to call an RFC Server application from ABAP/4

You can call an external RFC Server function from ABAP/4 as follows:

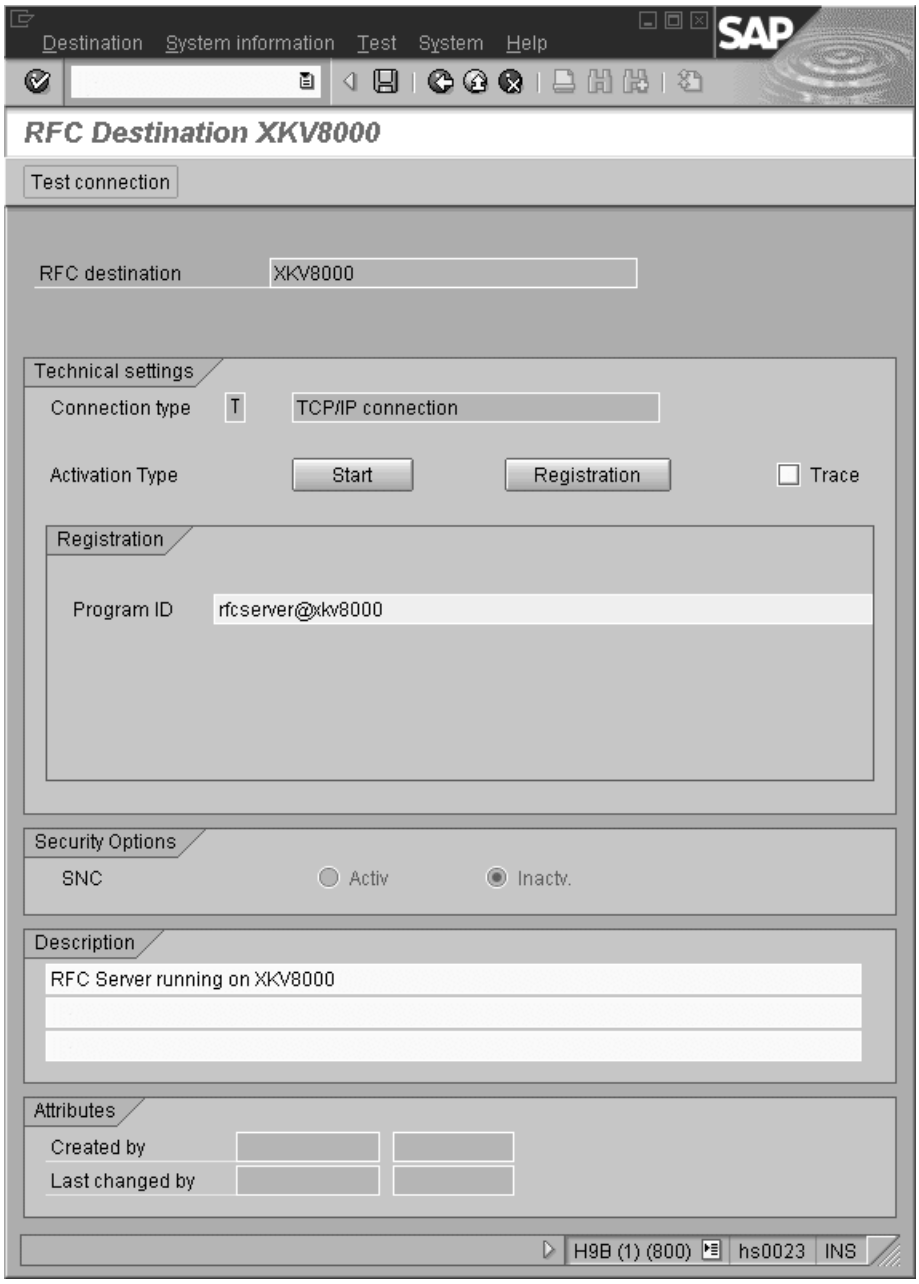
```
CALL FUNCTION 'MyServerFunction'
 DESTINATION 'XKV8000'
 EXPORTING ...
 IMPORTING ...
 TABLES ...
 EXCEPTIONS ...
 ...
```

The name of the function must refer to one of the modules contained in your RFC Server.

The value specified for DESTINATION must be a valid RFC destination. An RFC destination can be created in SAP R/3 under **Tools->Administration->Administration->Network->RFC Destination** (Transaction SM59). After clicking the **Create** toolbar button the following window is shown:

Select connection type (for example 'T' for a TCP/IP based RFC destination). Type a name for your destination and its description. Logon data must be provided if you want your RFC Server modules to be executed by another user than the one the ABAP/4 program uses. Then click Enter

to get the following window:



Click **Registration**. This allows you to enter a Progam ID under which your RFC Server can be reached on the SAP gateway. This must be the same name as the Program ID specified in your RfcServer object either set as a property or located in the destination entry of saprfc.ini.
Click Enter

to save the RFC Destination. Now your ABAP/4 program can use it.

RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

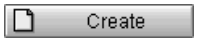
Developing an RFC Server application

Creating an ABAP/4 Report

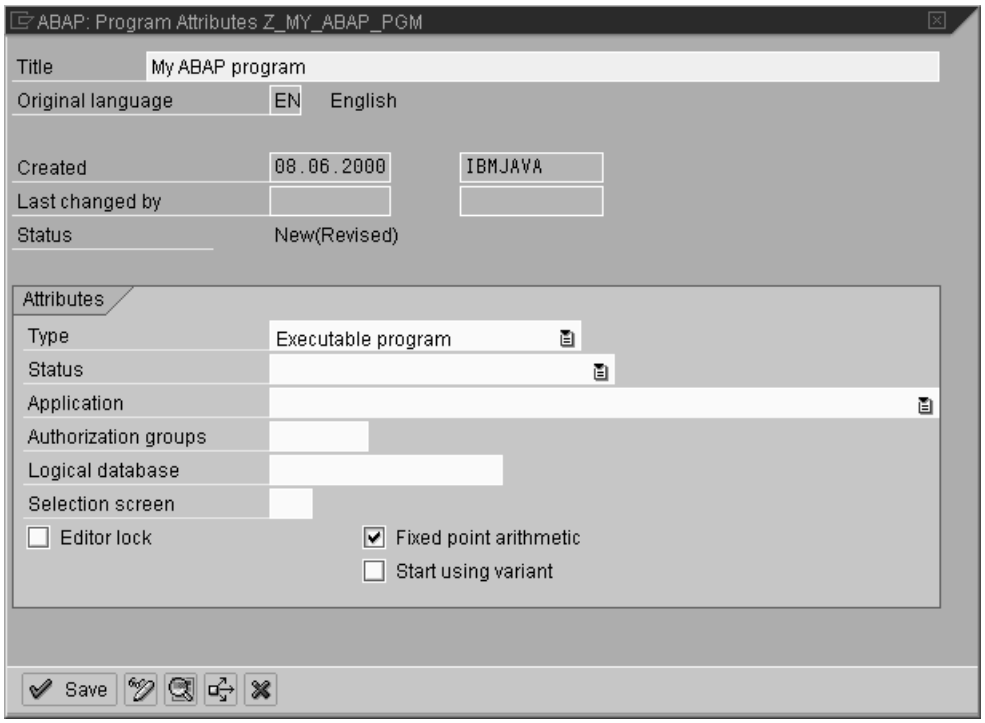
This topic shows you the steps you have to do for creating an ABAP/4 report.

After you have logged on to the SAP R/3 system select **Tools>ABAP Workbench>Development>ABAP Editor** from the navigation menu (Transaction SE38). The following dialog is shown:

Enter the name of your ABAP report. Choose 'Z' or 'Y' as the first letter. This convention means that your program belongs to the customer domain. All other starting letters are reserved by SAP internal programs. Click the Create

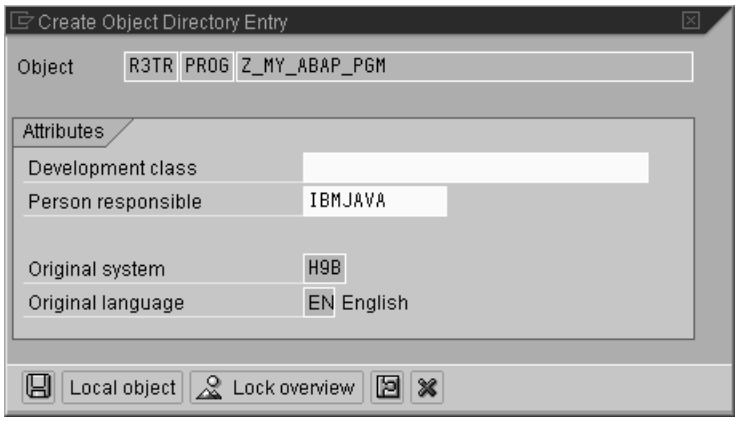


button and the `Program Attributes` dialog is shown:

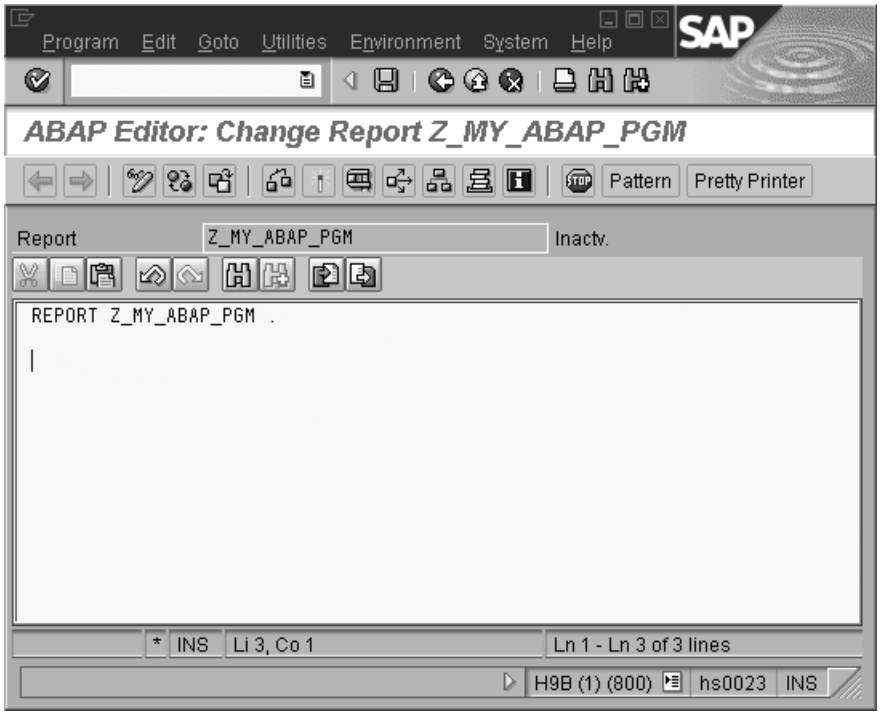


Specify the Type = Executable Program. Then click the Save

button to get to the `Create Object Directory Entry` dialog:



We intend our ABAP report to be a local object. This means, the report belongs to the `&TMP` class and cannot be transported to another SAP R/3 system. Therefore, click on the **Local object** button. Then your report will be created and the ABAP source code editor opens:



If your code resides in a text file on your local machine, open it with a text editor and copy its content into the clipboard. Then you can add the content of the clipboard to your ABAP source clicking the Insert

button.

Here are some more useful actions for working with your program:

- Click the **Check**

  button for performing a syntax check on your source code.
- Click the **Activate**

  button for activating your program.
- Click the **Test**

  button for running your program.

RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

Developing an RFC Server application

Using BAPI Helpvalues

A Java application developer can use the Helpvalues bean to request all valid values for import parameters of any BAPI method. These values can be provided to the user in any way. Then the user can choose one value to be used when performing this BAPI method.

The API of the Helpvalues bean provides two methods:

- One to query possible values of a simple import parameter and
- One to query possible values of a simple field inside a structure.

Distinguishing between these two approaches is necessary because the second alternative represents a rather complex scenario.

Query values for simple import parameters
Besides the common information as the Business Object's name or type and the name of the BAPI method only the nam of the parameter should be provided.

Query values for a simple field of a structure
Querying values for a structure's field is commonly more complex because there are often relations between several fields inside a structure.
Therefore it may be necessary to provide the names and values of related fields as filter conditions.

Besides the common information as the Business Object's name or type and the name of the BAPI method the following additional information should be provided:

- Name and type of the structure parameter
- Name of the field of interest
- An optional string array of <field name>/<value> pairs in the format "<field name>=<value>" specifying the context for the returned values

Providing a valid string array may reduce the number of possible values being returned to the caller because invalid combinations are filtered.

Invoking BAPIs and RFCs dynamically

To invoke BAPIs and RFCs dynamically you can use the autocreate functionality of the RFC module factory. Before reading this chapter make sure you understand the interfaces of the Common RFC Interface for Java.
The sample code below shows how to create an RFC module instance for the SAP BAPI "BAPI_COMPANYCODE_GETLIST":

```
// Get access to factory manager
FactoryManager aFactoryManager = FactoryManager.getSingleInstance() ;
// Get access to RFC module factory
IRfcModuleFactory aRfcModuleFactory = aFactoryManager.getRfcModuleFactory() ;
// Let RFC module factory create function module
IRfcModule bapiCompanycodeGetlist =
   aRfcModuleFactory.autoCreateRfcModule(aConnection, "BAPI_COMPANYCODE_GETLIST") ;
```

After this the input parameters have to be set and then the function module can be executed:

```
bapiCompanycodeGetlist.callReceive();
```

The return parameters of the call then can be retrieved using the standard mechanisms defined by the Common RFC Interface for Java. The code snippet below shows how to obtain the table “COMPANYCODE_LIST” from the function module.

```
ITable companycodeList = bapiCompanycodeGetlist.getTableParam("COMPANYCODE_LIST") ;
```

To access the result again the standard mechanisms defined by the Common RFC Interface for Java are used:

```
ITable companycodeList = bapiCompanycodeGetlist.getTableParam("COMPANYCODE_LIST") ;
int rowCount = table.getRowCount() ;
System.out.println ("Returned table has " + rowCount + " lines.");
IRow row = null ;
for (int i = 0; i < rowCount; i++)
{
   row = table.getRow(i) ;
   System.out.println(
      "\t" + row.getSimpleField("COMP_CODE").getString() +
      "\t" + row.getSimpleField("COMP_NAME").getString()) ;
}
```

If compared to working with proxies generated by Access Builder for SAP R/3 working with BAPIs and RFCs dynamically may seem cumbersome and difficult from a programming point of view. In addition using the autocreate functionality of IRfcModuleFactory has a major impact on runtime performance. The reason for this is that in the autocreate case for each creation of an RFC module the required meta information has to be obtained before the RFC module can be created. This results in many remote calls to the SAP R/3 system during the creation of a new RFC module - just to gather the meta information.
There are cases though where the generated proxy classes cannot be used. For these cases the approach shown here is the only choice.

RELATED CONCEPTS

Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules
Common RFC Interface for Java
Run-time classes for SAP R/3
BOR Access Classes

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules
Generating proxies from command line
Calling RFC modules
Running your application
Deploying your application

RELATED REFERENCES

ABAP/4 and Java data types
Package com.sap.rfc

Supporting a national language

You may want to adapt your application to a specific national language. If you integrate the Logon bean into your application you probably want the logon window to use the same national language.

The default language for the Logon bean is English. To use another language set the default locale to the appropriate localization information. For example, to set the localization information to German, specify:

```
java.util.Locale.setDefault( new java.util.Locale( de,DE))
```

Perform this task before you create an instance of the Logon bean; otherwise, you can invoke the updateAppearance() method of the Logon bean after setting the locale.

The national language information for the Logon bean is stored in the SapLogon.properties and the translated SapLogon_*.properties files. Depending on your language selection the correct properties file is used.

RELATED CONCEPTS

Logon bean

RELATED TASKS

Integrating the Logon bean into your application
Deploying your application

RELATED REFERENCES

Logon bean

Developing SAP R/3 applications using the InfoBus technology

The purpose of this topic is to give you a short overview about the basic concepts of the InfoBus technology and its use in your own SAP R/3 applications developed with Access Builder for SAP R/3. It does not compensate the reading of the detailed descriptions given in the official InfoBus 1.1 Specification documentation by Mark Colan, Lotus Development Corp. The following text is partly taken from this documentation.
For further information please refer to the official InfoBus homepage at Sun Microsystems. There you can find all the news and the official documents about the InfoBus technology and related topics.

Basic concepts of InfoBus
The InfoBus facilitates the dynamic exchange of data between multiple applications, applets, or servlets built from Java beans. The InfoBus is based on a notion of an information bus where components can plug into and asynchronously exchange data with any other component on the bus in a structured way, including arrays, tables, and database rowsets. It is designed for components working together in the same Java Virtual Machine (JVM). The current design does not directly deal with components running in separate JVMs, though in general it does provide facilities that allow building a bridge between different JVMs.

In contrast to a conventional communication between pure bean components the InfoBus mechanism is not based on an event/response model where the semantics of the interaction depend upon understanding a specific event or knowing specific callback routines. With the conventional Java beans model the interfaces of cooperating beans have to be well-known at development time or an introspection has to be done during run time to learn about the cooperating bean prior to calling any methods or reacting on any events. The InfoBus simply defines a small set of standard interfaces and specifies the protocol for the use of those interfaces. It forms a tightly typed contract between the cooperating beans and relies on these

methods and events to be supported by the participating beans. No introspection is required and the procedure calls are direct. The semantics of the data flow are based on interpreting the contents of the data spread over the InfoBus, not on the names or parameters from events, nor on the names or parameters of callbacks.

Exchanging and tracking data on the InfoBus
The beans within an InfoBus application can be divided into three types: data producers, data consumers and data controllers. Data producers act as the source of the data spread over the InfoBus, data consumers act as the receivers. Data controllers are specialized components that mediate the rendezvous between producers and consumers. They will not be discussed any further here. The data itself flows in named objects called data items.

To establish an InfoBus participation any Java component can connect to the InfoBus by implementing the InfoBusMember interface, obtaining an InfoBus instance and having the member to join it. Please note that there may be more than one InfoBus instance on one JVM. You may use the default InfoBus for the JVM or you may explicitly create your own.

Once an object is a member of an InfoBus, it receives bus notifications by implementing the InfoBusDataProducer interface or the InfoBusDataConsumer interface or both interfaces. The InfoBusDataConsumer receives announcement events about data availability. Similarly the InfoBusDataProducer receives data request events. The data producers announce the availabilty of new data items as they become ready, consumers solicit data from producers as they require that data. The rendezvous between them is arranged only by the name of the data. It is up to the application designer to designate the names for the data items that can be exchanged.

A data consumer should not need a detailed understanding of the data producer's internal data representation or structures to use and navigate through its data. This requires the data producer and the consumer to agree on a common encoding for the exchanged data. Therefore the InfoBus defines a small set of interfaces for various standard access protocols like ArrayAccess or RowsetAccess which are used to create data items with common access. Another intent of the InfoBus is for data consumers to require as little specialized understanding of data formats as possible. So a data item can be retrieved as a String or a Java object. The Java objects are typically object wrappers for primitive data types such as Integer or instances of other core classes such as Collection. Finally a data consumer can attempt to change the value of data items. It's up to the producer to enforce a policy on whether anyone can change the data.

Besides the data exchange protocol and the data access interfaces the InfoBus also defines an event-driven mechanism for monitoring changes to data items. According to it data consumers may register themselves on a data item to be notified about any changes of the data item's value. On a data item's value change each registered consumer is notified with a DataItemChangeEvent specifying the type of change and the data item that changed.

InfoBus support in Access Builder for SAP R/3
The Access Builder for SAP R/3 supports the InfoBus 1.1 specification. This support is explicitly provided in the JNI middleware layer.

If your generated Business Object beans are to share the SAP R/3 data they retrieve with other beans, applets, or applications on the InfoBus, you are limited to use JNI as middleware type. Since the generated beans represent general SAP

Business Objects for all kinds of middleware layers they do not directly provide the InfoBus interfaces. The InfoBus interfaces are only available through the following basic run-time classes for SAP R/3 that are stored inside those beans:

- **com.ibm.sap.bapi.Simple** (which implements the com.sap.rfc.ISimple interface)
- **com.ibm.sap.bapi.Structure** (which implements the com.sap.rfc.IStructure interface)
- **com.ibm.sap.bapi.Table** (which implements the com.sap.rfc.ITable interface)

You must cast any of the base data interface objects which the various factories or proxy beans return to the above enumerated object types before calling any of the specialized InfoBus related methods. The three classes **Simple**, **Structure** and **Table** are enabled as InfoBus data items and may be distributed over the InfoBus.

A concrete sample

The Catalog sample demonstrates data sharing via an InfoBus between two Java applications. One application - the **CatalogViewer** - is used to browse through a ProductCatalog obtained from an SAP R/3 system and fills an order bag with the selected items of a given quantity. The second application - the **OrderForm** - uses the **OrderTable** class to display the current order bag. It may place a complete order extended by a customer and an invoice number into SAP R/3. The shared data is held and managed in a **Table** object that is published on an InfoBus. There is no direct method invocation between the two applications, instead all communication is done using InfoBus mechanisms.

First of all, you have to create the basic **Table** object. In the Catalog sample this is done within the constructor of the class **CatalogViewer**:

```
 SimpleInfo[] structInfoOrderTable = {
  new SimpleInfo ("ITEMNAME", IFieldInfo.RFCTYPE_CHAR, 40, 0),
  new SimpleInfo ("ITEMNO",   IFieldInfo.RFCTYPE_CHAR, 18, 0),
  new SimpleInfo ("ITEMUNIT", IFieldInfo.RFCTYPE_CHAR,  3, 0),
  new SimpleInfo ("CURRENCY", IFieldInfo.RFCTYPE_CHAR,  5, 0),
  new SimpleInfo ("PRICE",    IFieldInfo.RFCTYPE_BCD,  12, 4),
  new SimpleInfo ("QUANTITY", IFieldInfo.RFCTYPE_INT,   6, 0)
 };
fieldOrderTable =
    (com.ibm.sap.bapi.Table) FactoryManager.getSingleInstance().getTableFactory().
    createTable( new ComplexInfo(structInfoOrderTable, fieldInfoBusObject) );
```

Although the table is still empty it is ready to use, and therefore the next step would be to designate a data item name for this object and announce it on the InfoBus. The data producer of the item has to do this. Now you may say that the **CatalogViewer** class is the data producer for the **Table** object, but then the application itself must implement the InfoBusDataProducer interface and register itself as a data producer to the InfoBus before it can announce the availability of the table data item. This implies that an InfoBus was previously instantiated and opened and the application has joined it as an InfoBusMember. To facilitate this process the **Table** class and the other base data classes provide some methods to propagate themselves as a DataItem, an InfoBusDataProducer, and an InfoBusMember simultaneously. This could be achieved through only one method call to **joinInfoBusAsMemberAndProducer**. That's what the **CatalogViewer** constructor does after creating the table:

```
fieldOrderTable.joinInfoBusAsMemberAndProducer (fieldInfoBusID, fieldInfoBusObject);
```

This call results in the **Table** object instantiating an InfoBus with the name fieldInfoBusID, opening it, and joining it as InfoBusMember. Then the **Table** object registers itself as a data producer and announces itself as an available data item with the name fieldInfoBusObject.

Please remember that this method is provided only for an easier use of the InfoBus. If you wish to do every single step by your own, for example because you want to participate in the InfoBus with only one data producer that offers multiple data items or you want to have more detailed control on the data flow, you are free to do so. For that the base data classes **Simple**, **Structure**, and **Table** provide the following methods which cover the different levels of InfoBus task integration:

1. The base data class is InfoBusMember, InfoBusDataProducer, and DataItem simultaneously:

   ```
   public void joinInfoBusAsMemberAndProducer (java.awt.Component componentForDefaultBusName,
       String dataItemName)
   ```

   This method uses the default InfoBus for the JVM by generating the default Name from componentForDefaultBusName (usually a this-pointer to the applet):

   ```
   public void joinInfoBusAsMemberAndProducer (String busName, String dataItemName)
   ```

   This method uses a named InfoBus with the Name busName.

2. The base data class is InfoBusDataProducer and DataItem simultaneously:

   ```
   public void joinInfoBusAsProducer (javax.infobus.InfoBusMember parentInfoBusMember,
       String dataItemName)
   ```

   This method uses the same InfoBus instance which the given parentInfoBusMember object has previously joined.

3. The base data class is a DataItem only:

   ```
   public void activateAsDataItem (String dataItemName,
       javax.infobus.InfoBusEventListener dataItemSource)
   ```

   This method takes the dataItemSource argument for returning references to its source object (usually the creating InfoBusDataProducer).

Back to the Catalog sample: By calling the **joinInfoBusAsMemberAndProducer** method the **Table** object as the InfoBusDataProducer announces the availability of the **Table** object as DataItem and all InfoBusDataConsumers registered at the InfoBus will receive an InfoBusItemAvailableEvent. In the Catalog sample the receiver of this event would be the **OrderTable** object which is an InfoBusDataConsumer that has previously joined the same InfoBus while processing its own constructor. The InfoBus 1.1 specification describes how to create an InfoBusDataConsumer and how to join the InfoBus.

The InfoBusItemAvailableEvent ends up in the **OrderTable**'s **dataItemAvailable**-method which you can see below:

```
public void dataItemAvailable (InfoBusItemAvailableEvent e)
{
    if (e.getDataItemName().equals(fieldInfoBusTable))
    {
        java.awt.datatransfer.DataFlavor[] prefDataFlavors =
            { new java.awt.datatransfer.DataFlavor("x-InfoBus/ScrollableRowsetAccess",
                                                   "x-InfoBus/ScrollableRowsetAccess") };
        Object dataItem = e.requestDataItem(this, prefDataFlavors);
        if ((dataItem == null) || !(dataItem instanceof ScrollableRowsetAccess))
            throw new ClassCastException("The InfoBus producer did not return a " +
                                         "ScrollableRowsetAcccess data item.");
        if (dataItem instanceof DataItemChangeManager)
```

```
            ((DataItemChangeManager) dataItem).addDataItemChangeListener(this);
        fieldInfoBusCursor = (ScrollableRowsetAccess) dataItem;
    }
}
```

First of all the OrderTable checks if the data item announced is really the one that it is interested in. The only matching criteria (the rendezvous) is the data item's name. If the names match the data item is requested by the requestDataItem method. The first parameter specifies a reference to the requesting InfoBusDataConsumer object, the second parameter is a java.awt.datatransfer.DataFlavor[] with which the consumer may specify his preferred data formats in descending order for the requested data item. Using a null-pointer indicates that the consumer does not prefer a special format. In the Catalog sample the **OrderTable** insists on getting a ScrollableRowsetAccess interface, otherwise throwing a ClassCastException if the InfoBusDataProducer does not return a data item of this kind. A complete description of the ScrollableRowsetAccess interface is documented in the Infobus 1.1 specification.

After having received the data item the **OrderTable** checks if it also supports the DataItemChangeManagerInterface. In this case the data item supports firing DataItemChangeEvents. As the **OrderTable** object also implements the DataItemChangeListener interface and is interested in any changes of the data item, it adds itself as a listener to all change events fired by the data item. Finally the **OrderTable** stores a reference to data item in a member variable for future use.

Back inside the **CatalogViewer**'s constructor it also registers itself as InfoBusDataConsumer at the InfoBus, requests the ScrollableRowsetAccess data item, and adds itself as DataItemChangeListener to the data item's listener list. From now on all changes to the ScrollableRowsetAccess data item, respectively the **Table** object, will cause the two DataItemChangeListener objects to receive a DataItemChangeEvent. So the **CatalogViewer** and the **OrderTable** are able to update their views of the shared data inside their requested data item.

Five different DataItemChangeEvents can be fired when a data item changes:

- DataItemAddedEvent
- DataItemDeletedEvent
- DataItemRevokedEvent
- DataItemValueChangedEvent
- RowsetCursorMovedEvent.

A DataItemChangeListener may react according to the type of event that occurred. Please note that in case of events indicating some sort of change to a **Table** object there is an additional property called "RowNumber" stored in those events whenever it is possible. The property indicates the 1-based row number that the change event is related to. For example, you can see the use of this property information in the **OrderTable**'s **dataItemDeleted** method where the "RowNumber"-property tells the number of the row that has been deleted.

```
public void dataItemDeleted(DataItemDeletedEvent event)
{
    Object rowNumber = event.getProperty ("RowNumber");
    if (rowNumber != null)
    {
        // idx holds the (zero-based) index of the row being deleted
        int idx = ((java.lang.Integer) rowNumber).intValue()-1;
        deleteRow (idx);
    }
}
```

When a DataItemRevokedEvent occurs the data consumer has to release all references to the specified data item and no further calls may be made to it any longer. In contrast to not calling any methods of the data item when the data item itself notifies its revocation, the consumer has to call the data item's release() method before he deletes his last reference to it.

At the end of every InfoBus participation an InfoBusDataProducer and an InfoBusDataConsumer should leave and close the InfoBus properly. In case of the Access Builder's base data classes **Simple**, **Structure**, and **Table** being an InfoBusDataProducer they provide the **leaveInfoBus** method for doing this. If they act only as a DataItem the **deactivateAsDataItem** method should be called. These methods also take care of firing the appropriate revoked-events making the corresponding objects no longer available on the InfoBus.

RELATED CONCEPTS

InfoBus homepage (Internet)
InfoBus 1.1 Specification documentation (Internet))
Run-time classes for SAP R/3

RELATED TASKS

Using the proxy beans to access SAP Business Objects
Running applets with JNI code in AppletViewer
Running applets with JNI code in Netscape Communicator
Running applets with JNI code in Java Plug-In

RELATED REFERENCES

Run-time classes for SAP R/3

Chapter 14. Samples

Sample: View documentation of Business Objects in IDES 4.5a

Objectives
To understand the typical features of Business Objects in an SAP R/3 system.
To see generated reference documentation.

Time Required
It might take you some time to understand all 176 Business Objects available in IDES 4.5a. If you just want to see the documentation structure and the features which a Business Object provides, you will probably want to allow about 5 to 10 minutes.

Description
The sample IDES 4.5a system provides 176 Business Objects. The documentation you see is fully generated by the Access Builder. In addition to the documentation file for each of these Business Objects, two other files are generated. The index file contains links to all Business Objects in alphabetical order. The BOR file contains a hierarchy of Business Objects by problem domain.

Instructions for viewing the sample documentation of Business Objects
You can directly view the documentation from the following links.

1. Hierarchy of Business Objects ordered by problem domain
2. Index of Business Objects ordered alphabetically
3. Employee provided as a Business Object with typical features

Sample: Retrieve company information with the proxy bean

Objectives
Use a proxy bean in your program.
Invoke a static and an instance BAPI method.

Time Required
You can easily run the sample in a few seconds. To rebuild the sample you probably want to allow about 10 minutes.

Description
The sample connects to the SAP R/3 system, retrieves detailed information about a given company, and prints the information to the output.

The sample uses

- CompanyCode Business Object
- getlist BAPI method of CompanyCode. This is a static method.
- getdetail BAPI method of CompanyCode. This is an instance method.

Instructions for running the sample
Before you can run this sample, you have to provide the correct middleware, user, and connection information to connect to your SAP R/3 system.

Perform the following steps:

1. Expand the Access Builder for SAP R3 Samples project

2. Expand the Default package for Access Builder for SAP R3 Samples package
3. Select the SampleCompanyCode class
4. Open the context menu of SampleCompanyCode class
5. Select the menu item **Properties**. The properties window of class SampleCompanyCode opens up
6. Select the Program tab
7. Enter the following command line arguments:

```
-conn            <JNI/Orbix> <additional params>
-userinfo        <client> <username> <password> <language>
-connectinfo     <hostname> <systemno>
```

   Example: `-conn JNI -userinfo 800 muster ides en -connectinfo 47.11.0.8 15`
8. Click **Ok** to leave the properties window.
9. Click the **Run**

   button.

After invocation you see the results in the command window or the VisualAge for Java console.

Note: The following samples are performing exactly the same operations on the SAP R/3 system as this sample. Therefore, if you compare the respective output you will notice that the output for those samples is exactly the same.

- Retrieve company information with the RFC module
- Retrieve company information without proxy bean

Sample: Retrieve company information without proxy bean

Objectives
Use the run-time classes for SAP R/3 directly.
Invoke a static and an instance BAPI method.
Compare the effort with that of using a proxy bean.

Time Required
You can easily run the sample in a few seconds. To rebuild the sample you probably want to allow about 30 minutes.

Description
The sample connects to the SAP R/3 system, retrieves detailed information about a given company, and prints the information to the output.

The sample uses

- Run-time classes for SAP R/3

Instructions for running the sample
Before you can run this sample, you have to provide the correct middleware, user, and connection information to connect to your SAP R/3 system.

Perform the following steps:

1. Expand the Access Builder for SAP R3 Samples project
2. Expand the Default package for Access Builder for SAP R3 Samples package

3. Select the SampleDirect class
4. Open the context menu of SampleDirect class
5. Select the menu item **Properties**. The properties window of class SampleDirect opens up
6. Select the Program tab
7. Enter the following command line arguments:

```
-conn            <JNI/Orbix> <additional params>
-userinfo        <client> <username> <password> <language>
-connectinfo     <hostname> <systemno>
```

   Example: `-conn JNI -userinfo 800 muster ides en -connectinfo 47.11.0.8 15`
8. Click **Ok** to leave the properties window.
9. Click the **Run**

button.

After invocation you see the results in the command window or the VisualAge for Java console.

Note: The following samples are performing exactly the same operations on the SAP R/3 system as this sample. Therefore, if you compare the respective output you will notice that the output for those samples is exactly the same.

- Retrieve company information with the RFC module
- Retrieve company information with the proxy bean

Sample: Retrieve company information with the RFC module

Objectives
Use an RFC proxy bean in your program.
Invoke an RFC module.

Time Required
You can easily run the sample in a few seconds. To rebuild the sample you probably want to allow about 20 minutes.

Description
The sample connects to the SAP R/3 system, retrieves detailed information about a given company, and prints the information to the output.

The sample uses

- RFC proxy bean BAPI_COMPANYCODE_GETLIST
- RFC proxy bean BAPI_COMPANYCODE_GETDETAIL

Instructions for running the sample
Before you can run this sample, you have to provide the correct middleware, user, and connection information to connect to your SAP R/3 system.

Perform the following steps:

1. Expand the Access Builder for SAP R3 Samples project
2. Expand the Default package for Access Builder for SAP R3 Samples package
3. Select the SampleRFCFunctions class

4. Open the context menu of SampleRFCFunctions class
5. Select the menu item **Properties**. The properties window of class SampleRFCFunctions opens up
6. Select the Program tab
7. Enter the following command line arguments:

```
-conn            <JNI/Orbix> <additional params>
-userinfo        <client> <username> <password> <language>
-connectinfo     <hostname> <systemno>
```

   Example: `-conn JNI -userinfo 800 muster ides en -connectinfo 47.11.0.8 15`
8. Click **Ok** to leave the properties window.
9. Click the **Run**

   button.

After invocation you see the results in the command window or the VisualAge for Java console.

Note: The following samples are performing exactly the same operations on the SAP R/3 system as this sample. Therefore if you compare the respective output you will notice that the output for those samples is exactly the same.

- Retrieve company information with the proxy bean
- Retrieve company information without proxy bean

Sample: Create a sales order manually

Objectives
Use proxy beans in your program.
Invoke a factory BAPI method.

Time Required
You can easily run the sample in a few seconds. To rebuild the sample you probably want to allow about 40 minutes.

Description
The sample logs on to the SAP R/3 system and creates a new sales order.

The sample uses

- SalesOrder Business Object proxy
- createfromdata BAPI method of SalesOrder. This is a factory method.

Instructions for running the sample
Before you can run this sample, you have to provide the correct middleware, user, and connection information to connect to your SAP R/3 system.

Perform the following steps:

1. Expand the Access Builder for SAP R3 Samples project
2. Expand the Default package for Access Builder for SAP R3 Samples package
3. Select the SampleCustomerOrderCreate class
4. Open the context menu of SampleCustomerOrderCreate class

5. Select the menu item **Properties**. The properties window of class SampleCustomerOrderCreate opens up
6. Select the Program tab
7. Enter the following command line arguments:

```
-conn            <JNI/Orbix> <additional params>
-userinfo        <client> <username> <password> <language>
-connectinfo     <hostname> <systemno>
```

   Example: `-conn JNI -userinfo 800 muster ides en -connectinfo 47.11.0.8 15`
8. Click **Ok** to leave the properties window.
9. Click the **Run**

   button.

After invocation you see the results in the command window or the VisualAge for Java console.

Sample: Retrieve company information visually

Objectives
Use the proxy beans in combination with the Visual Composition Editor of VisualAge for Java.
Use the Logon bean in the Visual Composition Editor.
Invoke a static and an instance BAPI method.

Time Required
You can easily run the sample in a few seconds. To rebuild the sample you probably want to allow about 10 minutes, it is entirely developed visually and does contain no line of hand-written code.

Description
The sample consists of a main window with two tables, one used for the table of companies and one for the detail structure of the selected company. The Logon bean is used to facilitate the logon to the SAP R/3 system.

- Logon bean
- CompanyCode Business Object
- getlist BAPI method of CompanyCode. This is a static method.
- getdetails BAPI method of CompanyCode. This is an instance method.

Instructions for running the sample
Perform the following steps:

1. Expand the Access Builder for SAP R3 Samples project
2. Expand the com.ibm.sap.bapi.demo.companycode package
3. Select the DemoCompanyCode class
4. Click the **Run**

   button.

After invocation continue with the following steps:

1. Click the **Logon** button to connect to the SAP R/3 system.

2. The SAP Logon window appears which is simply implemented by using the Logon bean. Fill in the information required in the user and system pages to logon to the SAP R/3 system. Press the **Logon** button.
3. If the logon is successful, the Logon window disappears and the list of company codes and company names are shown.
4. Select an item from the list and you get the details of this company.

Sample: Submit a sales order

Objectives
Invoke an instance and a factory BAPI method.
Use the LogonView bean in the Visual Composition Editor.
Use the proxy beans in combination with the Visual Composition Editor of VisualAge for Java.
Build a complex application with additional logic.

Time Required
You can easily run the sample in a few minutes. To rebuild the sample you probably want to allow about 1 - 2 hours.

Description
The sample allows you to logon to the SAP R/3 system, retrieve material information for given material codes, gather the items to be purchased, and create a new sales order.

The sample uses

- LogonView bean
- Customer, Material, and SalesOrder Business Objects
- getdetails BAPI method of Material. This is an instance method.
- createfromdata BAPI method of SalesOrder. This is a factory method.
- Prebuilt beans OrderItem, OrderBag, and OrderBagView

Instructions for running the sample
Perform the following steps:

1. Expand the Access Builder for SAP R3 Samples project
2. Expand the com.ibm.sap.bapi.demo.salesorder package
3. Select the OrderAppl class
4. Select the **Selected > Properties** menu item
5. Click the **Run**

   button.

Note that the following instructions use default values for a sample IDES application of SAP R/3 4.5a. You might have to override the values to fit your SAP R/3 system.

After invocation continue with the following steps:

1. You must fill in the appropriate information to connect to the SAP R/3 system in the user and system pages of the SAP Logon window. Click **Logon**. Note that the SAP Logon window is simply implemented by using the LogonView bean.
2. After successful logon the SAP Logon window disappears.

3. Click on the **Add Item** button to open the **Add Item** selection dialog.
4. Type a valid material code in the entry field following `Product`. In the sample IDES application of SAP R/3 4.5a you might use `M-06` (uppercase!) for a "Flatscreen MS 1460 P".
5. Tab to the entry field following `Quantity` and enter the quantity you would like to purchase. Leaving the `Product` entry field will issue the getdetail method of the generated Material class to invoke the BAPI_MATERIAL_GETDETAIL function that retrieves this information from the SAP R/3 system.
6. Click **Add** to add this item to the order bag. You may repeat these two steps of retrieving product information and adding the item to the order bag.
7. To submit the order you click **Purchase**. A new instance of the SalesOrder class is created with the given information. The appropriate constructor invokes the BAPI_SALESORDER_CREATEFROMDATA function that creates this order in the SAP R/3 system

Sample: Retrieve company information with EJB beans

Objectives
The CompanyCodeEjb sample demonstrates a minimal EJB Client/Server environment.

Time required
To deploy the EJB beans and run the sample you will need approximately 20 minutes.

Before you begin
The following features must be added before running this sample:

- IBM Access Builder for SAP R/3 Libraries
- Connector for SAP R/3
- Connector for SAP R/3 Samples
- IBM EJB Development Environment
- IBM WebSphere Test Enviroment
- IBM Common Connector Framework

You should be able to run the CompanyCode Sample Application and retrieve a list of CompanyCodes from an SAP R/3 system.

If you would like to understand the implementation details of the sample you should also be familiar with Sun's Enterprise JavaBeans specification.

Description
The client application instantiates a CompanyCodeEjb on the EJB server and calls its method getList() retrieving a list of all available CompanyCodes in the associated SAP R/3 system. Then it transfers the resulting table to the client and prints all its entries to the console (to stdout).

Instructions for running the sample

- Creating the EJB group
- Deploying the EJB beans
- Running the sample

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

RELATED REFERENCES

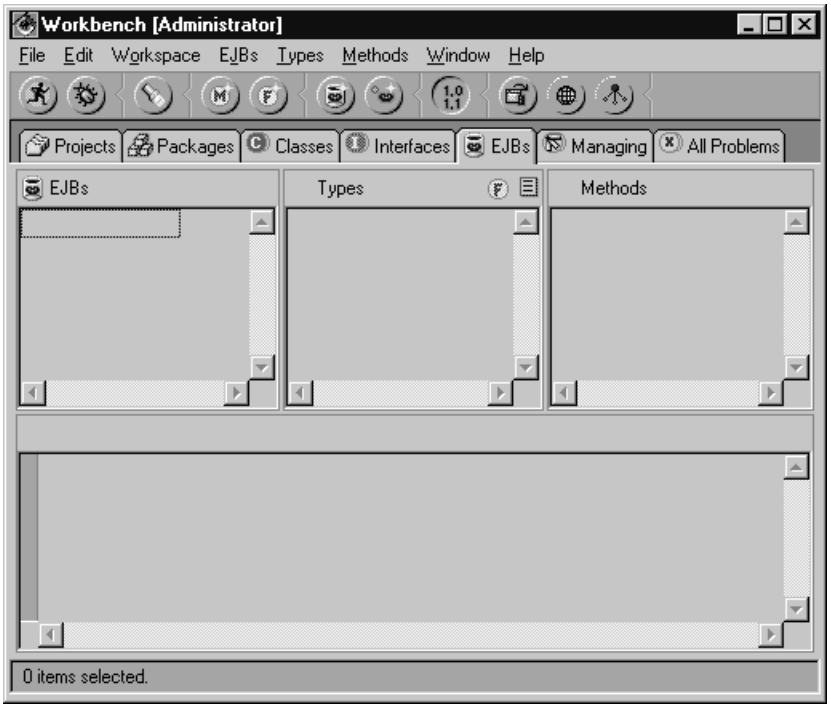
Naming conventions for generated classes

Creating the EJB group

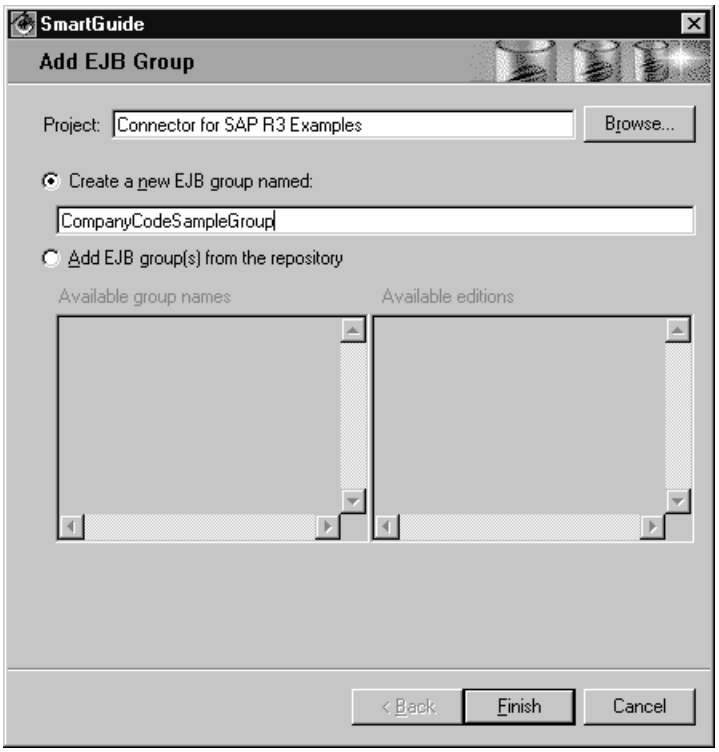
First, you create an open edition of the related projects to store the classes that are generated in the deployment process. In addition, you create a new EJB group that holds the deployed EJB beans.

Perform the following steps:

1. In the Workbench browser window, click on the Projects pane, select the Connector for SAP R/3 project and click the right mouse button. Then select **Manage > Create Open Edition** from the pop-up menu.
2. Then select the Connector for SAP R3 Samples project and click the right mouse button. Then select **Manage > Create Open Edition** from the pop-up menu. The two project are open editions now and ready to hold the new EJB beans that will be generated in this sample.
3. In the Workbench browser window, click on the EJB beans tab. The EJB beans page appears.



4. From the **EJB beans** menu, select **Add - EJB Group**. The Add EJB Group SmartGuide appears.



5. Besides the **Project** field, click the **Browse** button and select the project Connector for SAP R/3 Samples, then click **OK**.
6. Ensure that the **Create a new EJB group named** radio button is selected and type the name you want to assign to the new group into the entry field, for example CompanyCodeSampleGroup.
7. Click **Finish**. The new group is created and is displayed in the EJB beans pane.

Next step: Deploying the EJB beans

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

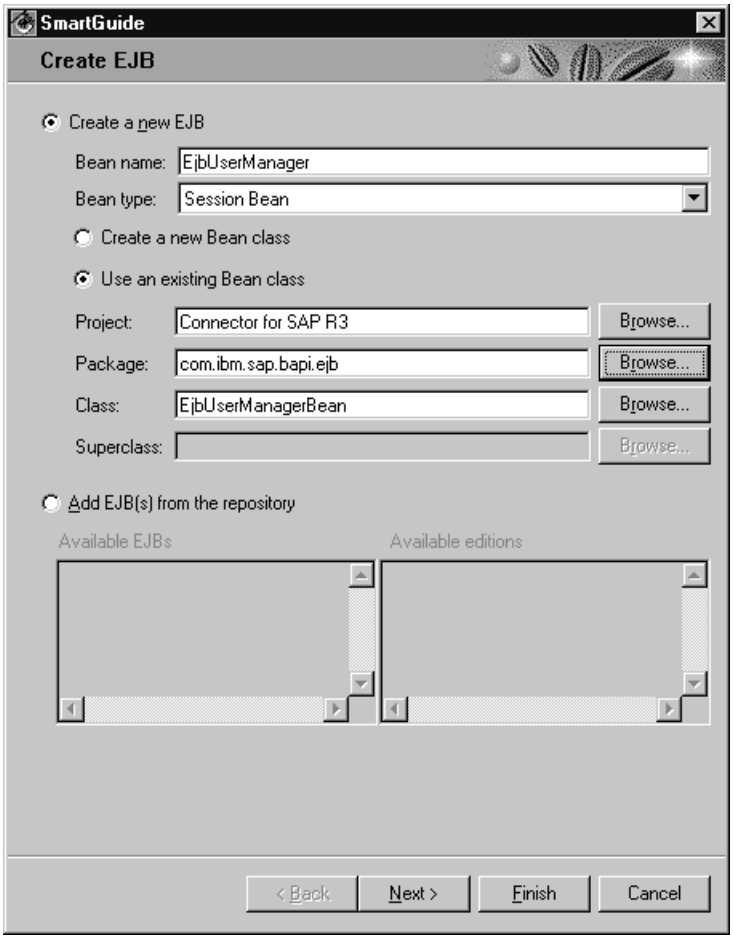
RELATED REFERENCES

Naming conventions for generated classes

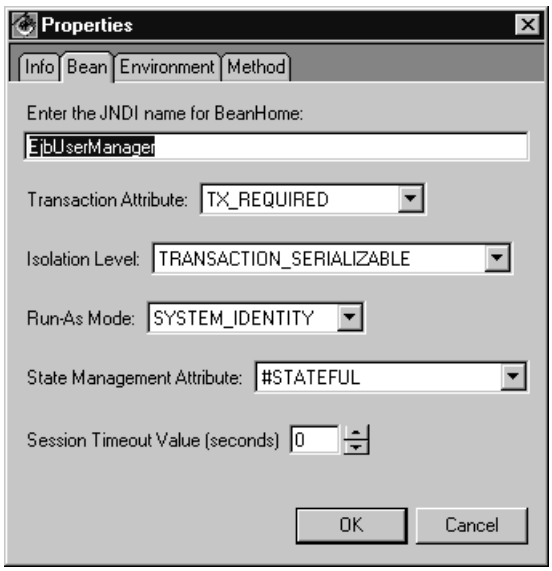
Deploying the EJB beans

To run the CompanyCodeEjb sample you must first deploy all required EJB beans, that is EjbUserManager, EjbTable, and CompanyCodeEjb. Please follow these steps:

1. First, you add the EjbUserManager to the previously created EJB group. In the EJB beans pane, select the created EJB group and click the right mouse button. Then select **Add > EJB** from the pop-up menu. The Create EJB SmartGuide appears.

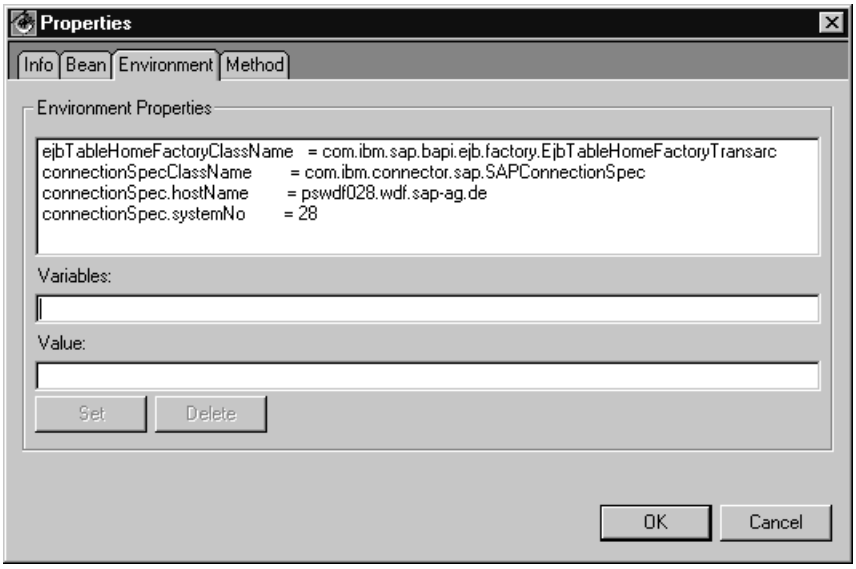


2. Ensure that the **Create a new EJB** radio button is selected and type EjbUserManager in the **Bean name** field.
3. The **Bean type** list must be set to **Session Bean**.
4. Please select the **Use an existing Bean class** radio button.
5. Click **Browse** next to the **Project** field and select the project that contains the EjbUserManager bean class: Connector for SAP R/3, then click **OK**.
6. Click **Browse** next to the **Package** field and select the package that contains the EjbUserManager bean class: com.ibm.sap.bapi.ejb, then click **OK**.
7. Click **Finish**. The EjbUserManager bean appears under the selected EJB Group.
8. Select the created EjbUserManager object and click the right mouse button. Then select **Generate - Deployed Code** from the pop-up menu. Now the process that generates the specific java classes for deploying the EjbUserManager starts.
9. Click **Yes** when asked if the creation of an edition of the package com.ibm.sap.bapi.ejb should be attempted and the generation process continues. When it is finished the Enterprise Java Server classes, the stubs, skeletons, and helper classes for the EjbUserManager are generated.
10. Again select the created EjbUserManager object and click the right mouse button. After selecting **Properties** from the pop-up menu the Bean page of the Properties window appears.



11. Leave all entry fields unchanged except the field **State Management Attribute**. Make sure that in the **State Management Attribute** list **#STATEFUL** is selected, then click **OK**. Now the EJB deployment descriptor properties for EjbUserManager are set.
12. Next, the EjbTable must be added to our EJB group. In the EJB beans pane, select the EJB group CompanyCodeSampleGroup and click the right mouse button. Then select **Add > EJB** from the pop-up menu. The Create EJB SmartGuide is appears.
13. Ensure that the **Create a new EJB** radio button is selected and type EjbTable in the **Bean name** field.
14. Ensure that in the **Bean type** list **Session Bean** is selected.
15. Please select the **Use an existing Bean class** radio button.
16. Click the **Browse** button next to the **Project** field and select the project that contains the EjbTable bean class: Connector for SAP R/3, then click **OK**.
17. Click the **Browse** button next to the **Package** field and select the package that contains the EjbTable bean class: com.ibm.sap.bapi.ejb, then click **OK**.
18. Click **Finish**. The EjbTable bean appears under the selected EJB Group.
19. Select the created EjbTable object and click the right mouse button. Then select **Generate - Deployed Code** from the pop-up menu to generate the necessary java classes.
20. Select **Properties** from the pop-up menu of the created EjbTable object. The Bean page of the Properties window appears.
21. Leave all entry fields unchanged except the field **State Management Attribute**. Ensure that **#STATEFUL** is selected in the **State Management Attribute** list, then click **OK**. The EJB deployment descriptor properties for EjbTable are set now.
22. Finally the CompanyCodeEjb must be added to your EJB group. Select **Add > EJB** from the pop-up menu of the CompanyCodeSampleGroup. The Create EJB SmartGuide appears.
23. Ensure that the **Create a new EJB** radio button is selected and type CompanyCodeEjb into the Bean name field.
24. In the **Bean type** list **Session Bean** must be selected.
25. Please select the **Use an existing Bean class** radio button.

26. Click the **Browse** button next to the **Project** field and select the project that contains the CompanyCodeEjb bean class: Connector for SAP R/3 Samples, then click **OK**.
27. Click the **Browse** button next to the **Package** field and select the package that contains the CompanyCodeEjb bean class: com.ibm.sap.bapi.demo.ejb.generated, then click **OK**.
28. Click **Finish**. The CompanyCodeEjb bean appears under the selected EJB Group.
29. select **Generate - Deployed Code** from the pop-up menu of the CompanyCodeEjb object. The process that generates the specific java classes for deploying the CompanyCodeEjb starts now.
30. Click **Yes** when asked if the creation of an edition of the package com.ibm.sap.bapi.demo.ejb.generated should be attempted and the generation process continues.
31. Select **Properties** from the pop-up menu of the created CompanyCodeEjb object. The Bean page of the Properties window appears.
32. Leave all entry fields unchanged except the field **State Management Attribute**. In the **State Management Attribute** list **#STATEFUL** must be selected.
33. In the Properties window, click the Environment tab. The Environment page appears.



34. Now you have to set several environment properties. To set the environment properties, specify the variable in the **Variables** field and its associated value in the **Values** field, and then click **Set**. Please set the ConnectionSpec properties as needed for your SAP R/3 target system. The following two examples show the minimal environment to be set:

Variable	**Value (example)** **Description**
ejbTableHomeFactoryClassName	com.ibm.sap.bapi.ejb.factory. EjbTableHomeFactoryTransarc Classname for the class to create the EjbTableHome instance.

Variable	Value (example) Description
connectionSpecClassName	com.ibm.connector.sap.SAPConnectionSpec Classname for the backend specific ConnectionSpec implementation for the CCF.
connectionSpec.hostName	pswdf028.wdf.sap-ag.de Hostname of the SAP R/3 system to which the EJB bean should connect.
connectionSpec.systemNo	28 System number of the SAP R/3 system to which the EJB bean should connect.

or alternatively with load balancing functionality:

Variable	Value (example) Description
ejbTableHomeFactoryClassName	com.ibm.sap.bapi.ejb.factory.EjbTableHomeFactoryTransarc Classname for the class to create the EjbTableHome instance.
connectionSpecClassName	com.ibm.connector.sap.SAPConnectionSpec Classname for the backend specific ConnectionSpec implementation for the CCF.
connectionSpec.loadBalancing	true Hostname of the SAP R/3 system to which the EJB bean should connect.
connectionSpec.systemName	DIP System name (system ID) of the SAP R/3 system to which EJB bean should connect.
connectionSpec.msgServer	pswdf028.wdf.sap-ag.de Message server of the SAP R/3 system to which the EJB bean should connect.
connectionSpec.groupName	PUBLIC Group name of the SAP R/3 system to which the EJB bean should connect.

35. After setting all relevant properties for your environment click **OK**. The EJB deployment descriptor properties for CompanyCodeEjb are set now.

Next step: Running the sample

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

RELATED REFERENCES

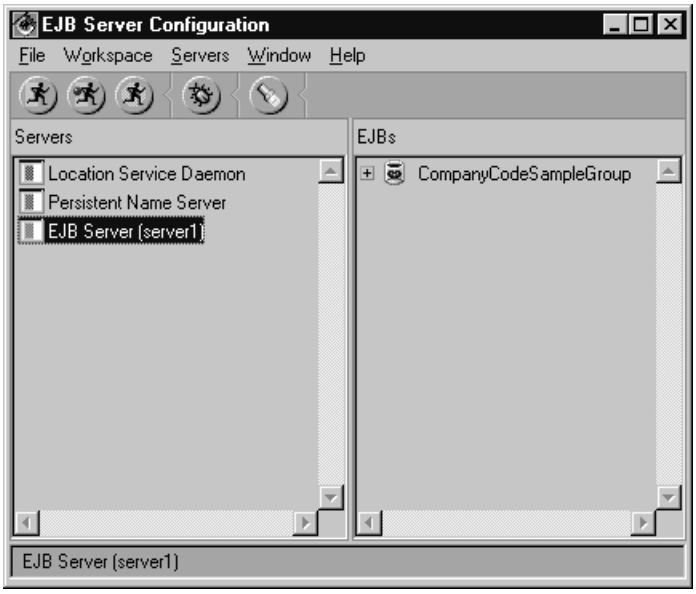
Naming conventions for generated classes

Running the sample

Follow these steps to run the sample in the EJB Test Environment:

1. In the Workbench browser window, click the EJB beans tab. The EJB beans page appears.

2. Select **Add To - Server Configuration** from the pop-up menu of the CompanyCodeSampleGroup. The Enterprise JavaServer Configuration browser appears.



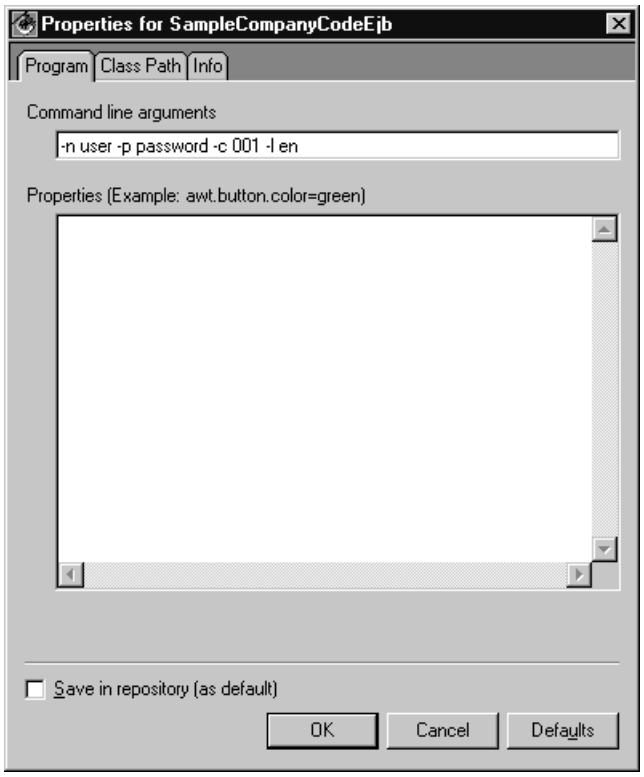
An EJB Server entry is added in the Servers pane. The EJB Server contains your EJB group.

3. In the Servers pane, select the **Location Service Daemon** and the **Persistent Name Server**, then click the right mouse button and select **Start Server**. The Console window appears in the background and monitors the two servers.
4. In the Console, check that the **Location Service Daemon** and the **Persistent Name Server** are listening by selecting each server in the All Programs pane and looking for the following messages in the Standard Out pane:

```
Location service daemon listening...
NameServer is listening...
```

The Name Service servers are running now.

5. Then the EJB Server can be started. In the Enterprise JavaServer Configuration browser, select the **EJB Server** in the Server's pane, click the right mouse button, and select **Start Server** from the pop-up menu. In the Servers pane, an icon appears beside the **EJB Server** to indicate that it is running and the Console window now also monitors the EJB server. The EJB server is now ready to receive requests from a client application.
6. In the Workbench browser, select the Projects pane and open the Connector for SAP R3 Samples project. Select the class com.ibm.sap.bapi.demo.ejb.SampleCompanyCodeEjb. Select **Properties** from the pop-up menu of this class. The Class Path page of the Properties window appears.
7. In Properties window, click the Program tab. The Program page appears.



8. Edit the **Command line arguments** field specifying your SAP R/3 logon information. Use the following switches for the different arguments:

   ```
   -n userName
   -p password
   -c client
   -l language
   -x codepage
   ```

   After specifying the **Command line arguments**, click **OK**. The SampleCompanyCodeEjb is now ready to run.

9. In the Workbench browser window, select the Projects pane and open the Connector for SAP R3 Samples project. Select the class com.ibm.sap.bapi.demo.ejb.SampleCompanyCodeEjb and click the right mouse button. Then select **Run > Run main**

10. In the Console, check the output of the **SampleCompanyCodeEjb** process by selecting it in the All Programs pane and look into the Standard Out pane. The output should look like this:

    ```
    Retrieving initial JNDI context...
    Retrieving EjbUserManagerHome interface...
    Creating new EjbUserManager...
    Setting UserInfo for user muster...
    Retrieving CompanyCodeEjbHome interface...
    Creating new CompanyCodeEjb...
    Calling CompanyCode.getlist()...
    Transferring companyCodeListTable...
    CompanyCodeListTable has xx entries.
    Entry No.1: 0001, SAP A.G.;
    Entry No.2: 1000, IDES AG;
    Entry No.3: 2000, IDES UK;
    Entry No.4: 2100, IDES Portugal;
    Entry No.5: 2200, IDES France;
    .....
    ```

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

RELATED REFERENCES

Naming conventions for generated classes

Sample: Retrieve company information with EJB beans visually

Objectives
This example demonstrates the usage of enterprise beans generated by the Access Builder for SAP R/3. It demonstrates how to create an EJB application visually using additional helper classes for an easy and effective way of application development.

Time required
You can run the sample in about 10 minutes. To rebuild the sample you probably want to allow about 30 minutes.
Before you begin
The following features must be added before running this sample:

- IBM Access Builder for SAP R/3 Libraries
- Connector for SAP R/3
- Connector for SAP R/3 Samples
- IBM WebSphere Test Enviroment
- IBM Common Connector Framework

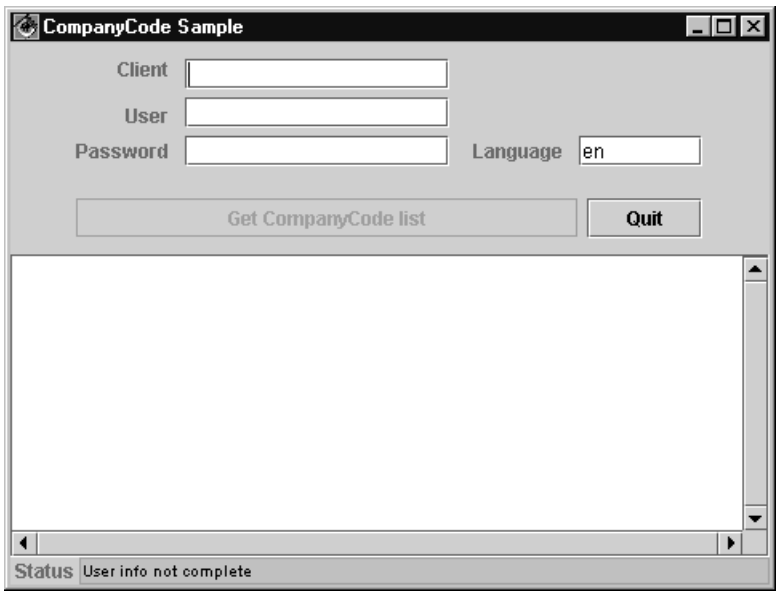
Description
The resulting program provides a graphical user interface which allows you to retrieve a list of company codes from a SAP R/3 system.

Instructions for running the sample
Note: Before you can run the sample ensure that the EJB beans are already generated, deployed, and started on the EJB server. Refer to the Retrieve company information with EJB beans sample to perform these steps.

To start the sample

- Select the class com.ibm.sap.bapi.demo.ejb.SampleCompanyCodeEjbStart located in the Connector for SAP R3 Samples project.
- Click the **Run**

button
A window prompting for client, userid, password, and language appears:

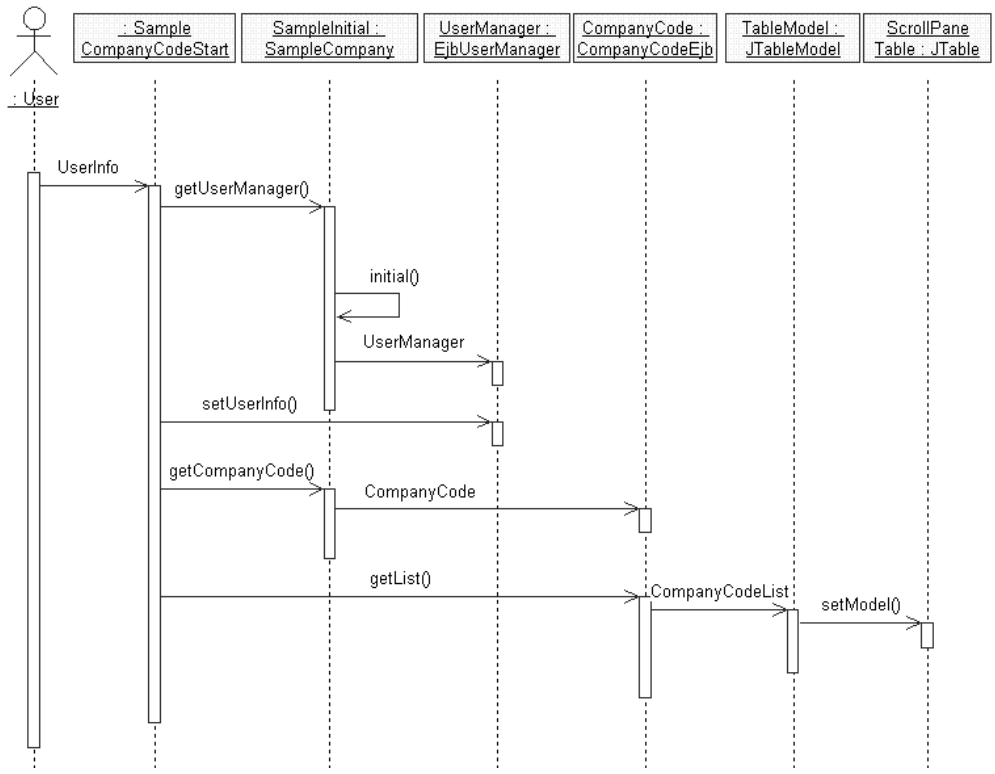


- Specify values for **client**, **user**, **password** and **language**. Then click the **Get CompanyCode list** button to retrieve the list of company codes from your SAP R/3 system.

Application logic
The following list contains objects and classes which implement the sample application

- **SampleInitial** (com.ibm.sap.bapi.demo.ejb.SampleCompanyCodeEjbInitial) contains initializing code, for example for requesting name services and creating proxies for **UserManager** and **CompanyCode**.
- **UserManager** (com.ibm.sap.bapi.ejb.EjbUserManager) manages user information which is necessary to establish connections to the target SAP R/3 system. Refer to CCF Connector for SAP R/3 for more information about this topic.
- **CompanyCode** (com.ibm.sap.bapi.demo.ejb.generated.CompanyCodeEjb) this class was generated by the Access Builder for SAP R/3 tool and contains the code to accomplish the functionality of a company code Business Object.
- **CompanyCodeList** (com.ibm.sap.bapi.demo.ejb.generated.Bapi0002_1Table) holds the company code data returned by **CompanyCode**.
- **TableModel** (com.ibm.sap.bapi.util.JTableModel) adapts the data to the **ScrollPaneTable**.
- **ScrollPaneTable**(com.sun.java.swing.JTable) a table view of the company code data, its meta data is automatically provided by **TableModel.**

The following diagram illustrates the application flow between these objects:



Instructions for building the sample

Follow these steps to create the example using the Visual Composition Editor:

1. Create a new class which is a subclass of JFrame.
2. Compose the class visually. This opens the Visual Composition Editor.
3. The JFrame component appears on the free-form surface. This is the main window of the aplication.
4. Open the properties of the JFrame and set the `beanName` and `title` property. Set the `layout` property to `BorderLayout`.
5. Add a JPanel to the free-form surface. Set the `layout` property to `GridBagLayout`.
6. Add the components (JLable, JTextField and JPasswordField) for **client**, **user**, **password** and **language** to this panel as shown in the picture above.
7. Add two JButton components to the JPanel. Set their `text` property to `Get CompanyCode List` and `Quit`.
8. Add the JPanel to the `north` area of the main window.
9. Add another JPanel to the free-form surface. Set the `layout` property to BoxLayout.
10. Add a JLabel to the JPanel. Set the `text` property to Status
11. Add a JTextfield next to the JLabel.Set the editable property to false and the beanName to StatusTextField. **Note:** The generated variable for this text field will be referenced in the program code.
12. Add the JPanel to the `south` area of the main window.
13. Add a JTable component to the `center` area of the main window. Now, the graphical components for the sample application are provided. We need some other beans, which do the functional work for us.

14. Click **Choose Bean** (bean type: **Class** and class name: com.ibm.sap.bapi.demo.ejb.SampleCompanyCodeEjbInitial) and place the Bean inside the free-form surface.
15. Click **Choose Bean** (bean type: **Variable** and class name: com.ibm.sap.bapi.ejb.EjbUserManager) and place the variable inside the free-form surface.
16. Click **Choose Bean** (bean type: **Variable** and class name: com.ibm.sap.bapi.ejb.CompanyCodeEjb) and place the variable inside the free-form surface.
17. With a right mouse-click on the CompanyCodeEjb - Bean select `Tear-Off Property` and the property embeddedTable(ITable). This will create a new bean variable.
18. Click **Choose Bean** (bean type: **Class** and class name: com.ibm.sap.bapi.util.JTableModel) and place the bean inside the free-form surface. Now, the GUI components and beans are ready to be connected.
19. Connect the `userManager` attribute of the SampleCompanyCodeEjbInitial bean with the `this` attribute of the EjbUserManager bean.
20. Connect the `actionPerformed` event of the **get CompanyCode list** button with the `setUserInfo` method of the of EjbUserManager bean. A dotted line appears since the connection is not yet complete.
21. Connect the `text` attributes of the TextField components for **client**, **user**, **password** and **language** with the corresponding parameter values of the preceding connection. This completes the connection.
22. Connect the `actionPerformed` event of the **get CompanyCode list** button with the `getCompanyCode()` method of the SampleCompanyCodeEjbInitial bean. This method will return a result of type CompanyCodeEjb. Connect the result with the `this` attribute of the CompanyCodeEjb bean.
23. Connect the `actionPerformed` event of the **get CompanyCode list** button with the `getlist()` method of CompanyCodeEjb bean.
24. Connect the `embeddedTable` property of the Bapi0002_1Table bean with the `table` property of the JTableModel bean.
25. Connect the `actionPerformed` event of the **get CompanyCode list** button with the `setModel()` method of JTable. This connection needs an input parameter. Connect the `this` property of JTableModel with the parameter value.
26. Connect the `actionPerformed` event of the **get CompanyCode list** button with the `repaint` method of JTable.
27. Create a method which performs a check on the text fields (see code below).

```
public void logon_DataCheck()
{
   if (getClientTextField().getText().length() != 0 && getUserTextField().getText().length()
         && getPasswordTextField().getText().length() != 0)
    {
       getGetListButton().setEnabled(true);
       getStatusTextField().setText("");
    }
    else
    {
       getGetListButton().setEnabled(false);
       getStatusTextField().setText("User info not complete");
    }
}
```

28. Connect the `keyReleased` event of the text fields **client**, **user**, **password** and **language** with the method created in the preceding step.

29. Connect the `actionPerformed` event of the **Quit** button with the `dispose` method of the main window.

Sample: Creating sales orders with EJB beans in WebSphere Application Server Enterprise Edition

Objectives
Learn how to deploy SAP Business Object proxy enterprise beans in WebSphere Application Server Enterprise Edition. Demonstrate a one-phase-commit scenario in an EJB Client/Server environment.

Time required
To deploy the EJB beans and run the sample you will need approximately 3 hours.

Before you begin

- WebSphere Application Server Enterprise Edition must be installed and configured properly.
- Access to a SAP R/3 system Release 4.5B or later is required. This restriction is due to the fact that one-phase-commit is not supported by the BAPI_SALESORDER_CREATEFROMDAT1 BAPI in releases lower than 4.5B (the BAPI performs an implicit COMMIT WORK in releases lower than 4.5B).
- The following features must be added before running this sample:
  - Access Builder for SAP R/3 - Libraries
  - Connector for SAP R/3
  - Connector for SAP R/3 Samples
  - IBM EJB Development Environment
  - IBM Common Connector Framework

Description
This is a sample application that shows how to deal with sales orders on an SAP R/3 system in a one-phase-commit scenario. The sample's business logic gets deployed on a WebSphere Application Server (Enterprise Edition). The server application is accessed by Java based clients.

Instructions for running the sample

- Creating the EJB-JAR file
- Building the sample application
- Loading the application into the System Manager
- Creating and configuring a server group on which the application is to run
- Running the application
- Working with the client GUI

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

RELATED TASKS

Deploying EJB beans into WepSphere Application Server Enterprise Edition 3.02
Setting up the CCF infrastructure
Generating proxy beans for SAP Business Objects and RFC modules
Deploying your application

Creating the EJB-JAR file

This section describes how to create the EJB-JAR file containing the enterprise bean proxy for the SalesOrder Business Object. This file serves as input for the deployment process. You have to perform the following steps:

1. Start VisualAge for Java
2. Create an EJB group
   - Create open editions of the Connector for SAP R/3 and the Connector for SAP R/3 Samples projects.
   - Switch to the EJB beans page by clicking on the EJB beans tab.
   - Select **EJBs > Add > EJB Group** to create a new EJB group.
   - In the Project field, select the project Connector for SAP R/3 Samples.
   - Ensure that the **Create a new EJB group** radio button is selected and type the name you want to assign to the new group into the entry field, for example SalesOrderEjbGroup.
   - Click **Finish**. The new group is created and is displayed in the EJB beans pane.
3. Add the EjbTable to our EJB group.
   - From the pop-up menu of the created EJB group select **Add > EJB**. The Create EJB SmartGuide appears.
   - Ensure that the **Create a new EJB** radio button is selected and type `EjbTable` in the Bean name field.
   - Ensure that in the Bean type list **Session Bean** is selected.
   - Select the **Use an existing Bean class** radio button.
   - In the Project field select the project that contains the EjbTable bean class: Connector for SAP R/3.
   - In the Package field select the package that contains the EjbTable bean class: com.ibm.sap.bapi.ejb.
   - Click **Finish**.
   - Click the right mouse button on the created EjbTable object and select **Properties** from the pop-up menu. The Bean page of the Properties window is shown.
   - Ensure that **TX_NOT_SUPPORTED** is selected in the Transaction Attribute list.
   - Ensure that **#STATEFUL** is selected in the State Management Attribute list, then click **OK**. The EJB deployment descriptor properties for EjbTable are set now.
4. Add the SalesOrderEjb to our EJB group.
   - From the pop-up menu of the created EJB group select **Add > EJB**.
   - Ensure that the **Create a new EJB** radio button is selected and type `SalesOrderEjb` into the Bean name field.
   - In the Bean type list **Session Bean** must be selected.
   - Select the **Use an existing Bean class** radio button.
   - In the Project field select the project that contains the SalesOrderEjb bean class: Connector for SAP R/3 Samples.
   - In the Package field select the package that contains the SalesOrderEjb bean class: com.ibm.sap.bapi.demo.ejb.cb.generated.
   - Click **Finish**.
   - From the pop-up menu of the created SalesOrderEjb object select **Properties**.

- Ensure that **TX_NOT_SUPPORTED** is selected in the Transaction Attribute list.
- Ensure that **#STATEFUL** is selected in the State Management Attribute list.
- In the Properties window, click the Environment tab. The Environment page is shown.
- You have to set several environment properties. Please set the ConnectionSpec properties as needed for your SAP R/3 target system. The following two examples show the minimal environment to be set:

Variable	Value
ejbTableHomeFactoryClassName	com.ibm.sap.bapi.ejb.factory. EjbTableHomeFactoryCBroker
connectionSpecClassName	com.ibm.connector.sap.SAPConnectionSpec
connectionSpec.hostName	Hostname of the SAP R/3 system to which the EJB bean should connect
connectionSpec.systemNo	System number of the SAP R/3 system to which the EJB bean should connect

or alternatively with load balancing functionality:

Variable	Value
ejbTableHomeFactoryClassName	com.ibm.sap.bapi.ejb.factory. EjbTableHomeFactoryCBroker
connectionSpecClassName	com.ibm.connector.sap.SAPConnectionSpec
connectionSpec.loadBalancing	true
connectionSpec.systemName	System name (system ID) of the SAP R/3 system to which the EJB bean should connect
connectionSpec.msgServer	Message server of the SAP R/3 system to which the EJB bean should connect
connectionSpec.groupName	Group name of the SAP R/3 system to which the EJB bean should connect

Note: This sample requires a SAP R/3 system of Release 4.5B or later.

- After setting all relevant properties for your environment click OK. The EJB deployment descriptor properties for SalesOrderEjb are set now.

5. Export the EJB-JAR file by selecting **Export > EJB-JAR** from the pop-up menu of the EJB group. Specify the directory and name of the EJB-JAR file, for example Drive:\SalesOrderEjbDirectory\SalesOrderEjb.jar.

Next step: Building the sample application

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

RELATED TASKS

Deploying EJB beans into WepSphere Application Server Enterprise Edition 3.02
Setting up the CCF infrastructure
Generating proxy beans for SAP Business Objects and RFC modules
Deploying your application

Building the sample application

This section describes how to deploy the enterprise beans into WebSphere Application Server Enterprise Edition. This is done by using the **cbejb** tool.

1. Set the CLASSPATH system environment variable to include the following classes:
   - The SalesOrder enterprise proxy bean and helper classes generated with the Access Builder for SAP R/3. You find them in VisualAge for Java in the com.ibm.sap.bapi.demo.ejb.cb.generated package (Project: Connector for SAP R/3 Samples). You should export the package as JAR file.
   - Access Builder for SAP R/3 Libraries (Drive:\IBMVJava\eab\runtime30\ivjsap20.jar where Drive:\IBMVJava is the location of your VisualAge for Java installation)
   - Infobus (Drive:\IBMVJava\eab\runtime30\infobus.jar)
   - The Common Connector Framework (CCF)
2. Open a comand prompt
3. Change to the directory that contains the EJB-JAR file.
4. Run the **cbejb** tool with the commandline option **ccf** on the EJB-JAR file (this may take a while...):

   ```
   cbejb -ccf SalesOrderEjb.jar
   ```

Next step: Loading the application into the System Manager

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

RELATED TASKS

Deploying EJB beans into WepSphere Application Server Enterprise Edition 3.02
Setting up the CCF infrastructure
Generating proxy beans for SAP Business Objects and RFC modules
Deploying your application

Loading the application into the System Manager

This procedure installs the sample application family on the System Manager host of your Component Broker network. The System Manager can then make the application files available automatically wherever you configure the application to run.

This example uses the Load application action of the Component Broker System Manager user interface to install the application, a technique normally only used in an application development environment. (Normally, applications are installed into Component Broker by an installation tool provided with the application family package.)

To install the application, use the System Manager user interface to complete the following steps:

1. Change the user level to expert: **View > View Level > Control**

2. Click the Host Images folder, and select sm_host_name, where sm_host_name is the name of the System Manager host
3. From the pop-up menu of sm_host_name, select **Load Application**. The Load Application dialog is shown..
4. Select the SalesOrderEjbFamily.ddl file:

   ```
   Drive:\SalesOrderEjbDirectory\Working\Nt\PRODUCTION\SalesOrderEjbFamily.ddl
   ```
5. Click OK. This starts the System Manager which will load the application family and displays an Action console window. When you see a message indicating that the action has completed, you can close the Action console window.

Next step: Creating and configuring a server group on which the application is to run

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

RELATED TASKS

Deploying EJB beans into WepSphere Application Server Enterprise Edition 3.02
Setting up the CCF infrastructure
Generating proxy beans for SAP Business Objects and RFC modules
Deploying your application

Creating and configuring a server group on which the application is to run

This step is not part of the normal procedure for installing an application. It is used here on the assumption that there is no server group on which the sample application can run.

Normally, you might have a server or server group already defined, and you don't have to create a new one.

To create a server group on which the sample application is to run, complete the following steps:

1. Select **Tasks -> Create Server**
2. Select **Sample Application Zone**
3. Select **Sample Configuration**
4. Select **defaultServerGroup**
5. Provide `myserver` as name for the server, click **Add**
6. Click **Finish**

To configure the new server group, complete the following steps:

1. Select **Tasks -> Configure Server**
2. Add EjbTableApp and SalesOrderEjbApp to the right window
3. Select **Sample Application Zone**
4. Select **Sample Configuration**
5. Add **defaultServerGroup** to the right window
6. Click **Finish**

To activate the configuration and bind the JNDI names of the enterprise beans in the Component Broker namespace, complete the following steps:

1. Expand the **Management Zones > Sample Application Zone > Configurations**
2. Select **Activate** from the pop-up menu of **Sample Configuration**
3. Wait until activation completes
4. Open a command prompt and change to the directory containing the EJB-JAR file (Drive:\SalesOrderEjbDirectory)
5. Bind the JNDI name of the SalesOrderEjb enterprise bean in the Component Broker namespace:

   ```
   ejbbind SalesOrderEjb.jar com.ibm.sap.bapi.demo.ejb.cb.generated.SalesOrderEjb
   ```

6. Bind the JNDI name of the EjbTable enterprise bean in the Component Broker namespace:

   ```
   ejbbind SalesOrderEjb.jar com.ibm.sap.bapi.ejb.EjbTable
   ```

Ensure that the Connector for SAP R/3 dynamic link libraries are in the library path. You find the libraries for the respective platforms in the following directories:

- AIX
  - Drive:\IBMVJava\eab\lib\libivjsid20.so
  - Drive:\IBMVJava\eab\lib\libivjsij20.so
- Solaris
  - Drive:\IBMVJava\eab\solaris\libivjsid20.so
  - Drive:\IBMVJava\eab\solaris\libivjsij20.so
- Windows NT
  - Drive:\IBMVJava\eab\bin\ivjsij20.dll
  - Drive:\IBMVJava\eab\bin\ivjsid20.dll

Please also add the respective SAP RFC library to the library path. For Windows NT this DLL is named librfc32.dll and can be found in Drive:\IBMVJava\eab\bin\. For the other platforms please refer to the documentation of the SAP client.

Next step: Running the application

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

RELATED TASKS

Deploying EJB beans into WepSphere Application Server Enterprise Edition 3.02
Setting up the CCF infrastructure
Generating proxy beans for SAP Business Objects and RFC modules
Deploying your application

Running the application

Before running the sample, ensure that the Component Broker sample has been configured as described in the preceding sections.

To run the client, perform the following steps:

1. Start VisualAge for Java
2. Create a new project

3. Import the Drive:\CBroker\lib\somojor.zip JAR file into the new project, where Drive:\CBroker is the location of your Component Broker installation
4. Expand the Connector for SAP R/3 Samples project
5. Export the package com.ibm.sap.bapi.demo.ejb.cb as JAR file in a new directory Drive:\SalesOrderEjbClientDirectory\SalesOrderEjbClient.jar
6. Export the package com.ibm.sap.bapi.demo.ejb.cb.generated as JAR file: Drive:\SalesOrderEjbClientDirectory\SalesOrderGenerated.jar
7. Open a command prompt and change to the directory containing the JAR file
8. Copy ioser.dll from Drive:\CBroker\bin into that directory (or make sure the DLL is in the library path)
9. Set the classpath to include the following classes:
   - The client JAR (Drive:\SalesOrderEjbClientDirectory\SalesOrderEjbClient.jar)
   - The generated SalesOrderEjb client JARs: Drive:\SalesOrderEjbDirectory\Working\Nt\Production\SalesOrderEjbClient.jar and Drive:\SalesOrderEjbDirectory\Working\Nt\Production\EjbTableClient.jar
   - The SalesOrder enterprise proxy bean and helper classes generated with the Access Builder for SAP R/3 (Drive:\SalesOrderEjbClientDirectory\SalesOrderGenerated.jar)
   - Access Builder for SAP R/3 Libraries (Drive:\IBMVJava\eab\runtime30\ivjsap20.jar where Drive:\IBMVJava is the location of your VisualAge for Java installation)
   - Infobus (Drive:\IBMVJava\eab\runtime30\infobus.jar)
   - The somojor.zip file provided by ComponentBroker (Drive:\CBroker\lib\somojor.zip)
   - Sun Swing Version 1.0.3
10. Ensure that the Connector for SAP R/3 dynamic link libraries are in the library path. You find the libraries for the respective platforms in the following directories:
    - AIX
      - Drive:\IBMVJava\eab\lib\libivjsid20.so
      - Drive:\IBMVJava\eab\lib\libivjsij20.so
    - Solaris
      - Drive:\IBMVJava\eab\solaris\libivjsid20.so
      - Drive:\IBMVJava\eab\solaris\libivjsij20.so
    - Windows NT
      - Drive:\IBMVJava\eab\bin\ivjsij20.dll
      - Drive:\IBMVJava\eab\bin\ivjsid20.dll

    Please also add the respective SAP RFC library to the library path. For Windows NT this library is named librfc32.dll and can be found in Drive:\IBMVJava\eab\bin\. For the other platforms please refer to the documentation of the SAP client.
11. Run the client:

    ```
    java -Dcom.ibm.CORBA.BootstrapHost=hostname -Dcom.ibm.CORBA.BootstrapPort=port com.ibm.sap.bapi.
    ```

    where hostname and port specify your Component Broker host and userid, password, client, and language is the SAP R/3 logon information.

Next step: Working with the client GUI

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

RELATED TASKS

Deploying EJB beans into WepSphere Application Server Enterprise Edition 3.02
Setting up the CCF infrastructure
Generating proxy beans for SAP Business Objects and RFC modules
Deploying your application

Working with the client GUI

The client displays a window which is divided into four parts:

- Entry fields for organization, channel, division, PO number, and customer and buttons to create a sales order, commit, or rollback.
  The **Create SO** button is used to create a new sales order (BAPI_SALESORDER_CREATEFROMDAT1 call) using the information from the entry fields and the item table described below.
  The **Commit** button is used to commit all created but not yet committed sales orders.
  The **Rollback** button is used to discard all uncommitted sales orders.
- An item table and entry fields for material (e.g. M-01, M-15, P-109) and quantity. Use the entry fields together with the **Add Item**, **Remove Item**, and **Clear Table** buttons to modify the item table.
- The table of sales order documents. Click the **Getlist** button to get the current list of sales order documents (BAPI_SALESORDER_GETLIST call) for the selected customer and organization. Note that only committed sales orders will be displayed.
- A text area showing the actions you perform and providing status information.

A typical usage scenario could be the following:

1. Enter the following information valid for your SAP R/3 system:
   - organization
   - channel
   - division
   - PO number
   - customer
2. Click **Getlist** to see a list of all sales orders for this customer
3. Add items to a new sales order
   - Type material and quantity in the respective entry fields
   - Click the **Add Item** button
   - Repeat this step for all desired materials
4. Click **Create SO** button to create a new sales order
5. You may repeat steps 3. and 4. to create further sales orders
6. Click either
   - **Commit** to commit the changes in the SAP R/3 system
   - **Rollback** to discard the changes.

At any time click **Getlist** to see the list of committed sales orders.

RELATED CONCEPTS

CCF Connector for SAP R/3
Enterprise bean proxies for Business Objects

RELATED TASKS

Deploying EJB beans into WepSphere Application Server Enterprise Edition 3.02
Setting up the CCF infrastructure
Generating proxy beans for SAP Business Objects and RFC modules
Deploying your application

Sample: Process purchase requisitions with EJB beans

Objectives
The Purchase Requisition sample demonstrates the usage of the SAP R/3 Business Object EJB beans which the IBM Access Builder for SAP R/3 tool generates. The application is based on the purchase requisition business process in SAP R/3.

Time Required
This is a rather complex sample that demonstrates a real world scenario. To understand and exploit the sample you propably need about an hour.

Description
In the following, you find more information about

- The business process purchase requisition and the related Business Objects
- Internals of the application

Instructions for running the sample
The following steps explain how to

- Run the sample
- Create and delete a purchase requisition
- Show information about purchase requisitions
- Change a purchase requisition
- Release a purchase requisition item

Business process

The following section describes an application scenario that is based on the SAP R/3 purchase requisition business process.

A purchase requisition is used to communicate purchasing requirements of a requesting department to a corporate department. The reason of creating a purchase requisition could be a lack of material or the need of a service.

A purchase requisition contains several items. Each item represents one single request of a material or a service. As a result of a purchase requisition item a request for quotation or a purchase order is generated. If a purchase requisition meets certain conditions (for example the value exceeds $10,000), it needs to be approved before it can be processed. After approving the purchase requisition the corresponding items can be released.

Relating to the SAP R/3 purchase requisition business process this application scenario focuses on standard functions like

- Creating a purchase requisition

- Displaying information about a purchase requisition
- Changing a purchase requisition
- Deleting a purchase requisition item
- Releasing a purchase requisition item

with some restrictions

- Only purchase requisition items of a standard procurement type are created
- Only material requests are covered. Material items can be selected from a product catalog.
- Purchase requisition items are created with a single account assignment. It entails the specification of an account and a cost center per item.

SAP R/3 Business Objects used in this application are

- PurchaseRequisition
- PurchaseReqItem

The following table contains the attributes of a purchase requisition item provided by this scenario and information about how each attribute is related to the above listed functions

Attribute	Description	Create	Can be changed?
PREQ_ITEM	item number	M	
PREQ_NAME	name of requisitioner	M	
MATERIAL	material number	M	
MAT_GRP	material group	S	
C_AMT_BAPI	price per unit	S	
CURRENCY	currency	S	
PUR_GROUP	purchasing group	A	
SHORT_TEXT	text (descriptor of the requested material)	S	
PLANT	plant	M	
QUANTITY	quantity	M	X
UNIT	unit	S	
DELIV_DATE	requested delivery date	M	X
G_L_ACCT	G/L account number	M	
COST_CTR	cost center	M	

M = must be specified,

S = specified by selecting a material from a product catalog

A = determined while creating a purchase requisition

Application internals

The sample uses proxy EJB beans for the SAP R/3 Business Objects. The proxy EJB beans can be deployed to any EJB container. This allows a multi tier access to a proxy bean which is instantiated on an EJB server. The main characteristics of these beans are

- Methods that map to the corresponding BAPIs of SAP R/3 Business Objects
- Contained command object for each BAPI call
- Set and get methods for BAPI parameters (simple, structure, and table types)

Proxy beans and other helper classes generated for the Purchase Requisition Application are located in package com.ibm.sap.bapi.demo.purchreq.generated. These classes are

- Proxy EJB beans for the SAP R/3 Business Object PurchaseRequisition (BUS2105) and PurchaseReqItem (BUS2009)
- Classes which allow (cached) access to SAP R/3 table parameters for each environment option (for example BapiebancTable, BapiebancEjbTable, BapiebancTableRow)
- Command classes which correspond to BAPI calls (for example BAPI_REQUISITION_GETITEMS)

The design of the application is based on the interaction between the application model and the GUI components. The GUI classes implement certain listener interfaces which the application model disposes. Thereby the GUI components will be notified about changes in the application model. On the other side a GuiController object receives the GUI events and calls the appropriate methods in the application model.

Application Model
The application model consists of several classes which represent the elements of the process of a purchase requisition. These classes are located in the package com.ibm.sap.bapi.demo.purchreq.base. The following provides a short description of each class.

ApplModel	
	• The center of the application model • Only one instance (Singleton) • Creates the client side stubs for the EJB beans • Contains and manages objects of other application classes • Holds logon data to establish R3 connections
BapiException	
	• Indicates an error in a BAPI call
CreateRequisition	
	• Represents a purchase requisition which can be created • Contained as a single instance in the ApplModel object • Contains a list of purchase requisition items • Provides methods to maintain the list (adding and removing items)
CreateRequisitionEvent	
	• Holds the data of a change request to the CreateRequisition object
CreateRequisitionListener	
	• Must be implemented by a class to obtain notifications about changes in the CreateRequisition object • Is implemented by a corresponding GUI class to enable synchronization of its contents with the CreateRequisition object
Helper	
	• Implements several static helper functions • Enables tracing
Item	
	• Represents a single purchase requisition item • Serves as a wrapper for two table rows, since the item data entails general and account information

ItemTable	
	• Represents a table of purchase requisition items • Enables selection of items according to different search criteria • Contained as a single instance in the ApplModel object
ItemTableListener	
	• Must be implemented by a class to obtain notifications about changes in the ItemTable object • Is implemented by a corresponding GUI class to enable synchronization of its contents with the ItemTable object
ProductCatalogTreeNode	
	• A tree node in the product catalog hierarchy • Is returned from a ProductCatalogServer
PurchaseReqItem	
	• A wrapper class for the generated PurchaseReqItem classes • Delegates method calls to an internal PurchaseReqItem object
UpdateRequisition	
	• Represents a purchase requisition which can be changed • Contained as a single instance in the ApplModel object • Contains a list of purchase requisition items; only the changed items are sent to SAP R/3 • Provides methods to change a selected item
UpdateRequisitionEvent	
	• Holds the data of a change request to the CreateRequisition object
UpdateRequisitionListener	
	• Must be implemented by a class to obtain notifications about changes in the UpdateRequisition object • Is implemented by a corresponding GUI class to enable synchronization of its contents with the UpdateRequisition object

GUI Components

The GUI part of the application consists of several classes based on the JFC graphic library and their related model classes. They can be found in the package com.ibm.sap.bapi.demo.purchreq.gui.

Class	Description
CreateDialog	
	• Dialog window for creating purchase requisitions • Communicates with the CreateRequisition object in ApplModel • Contains a product catalog tree view, a detail panel, and an item table
CreateRequisitionTableModel	
	• Controls the item table in CreateDialog
DeleteDialog	
	• Window for confirming a delete item request

Class	Description
DetailPanel	• Panel used to display and enter data for a purchase requisition item • Contained in CreateDialog and UpdateDialog
ErrorLogDialog	• A window for displaying error log information
GuiController	• Coordinates GUI activity • Receives GUI events and calls the appropriate methods in the application model
GuiException	• Indicates an error in a GUI component (for example input error, wrong date format,...)
ItemTableModel	• Controls the item table in MainFrame
LogonFrame	• Frame window used to enter logon information • Is controlled by a class which extends LogonModel
LogonModel	• Abstract class which enables logon processing in a separate thread
LogonModelImpl	• An implementation of the LogonModel for this application
MainFrame	• The main window • Contains tool bar and menu bar to control the application • Contains an item table to select purchase requisition items
ReleaseDialog	• Window for confirming a release item request
RunLogoff	• Implements Runnable to enable logoff processing in a separate thread
UpdateDialog	• Window for changing purchase requisitions • Communicates with the UpdateRequisition object in ApplModel • Contains a detail panel and an item table
UpdateRequisitionTableModel	• Controls the item table in UpdateDialog

Running the sample

Before you can start the application inside VisualAge for Java you must perform the following steps

1. Create an EJB group (or use an existing one)
2. Add all EJB beans which are necessary for running the Purchase Requisition Application to this EJB group (see below)
3. Configure EJB beans (properties, environment, ...)
4. Create deployed code
5. Add the EJB group to the server configuration
6. Start server (Location Service Daemon, Persistence Name Service, and the EJB server including your EJB group)

You can find a detailed description of these steps in the Retrieve company information with EJB beans sample. Additional information is provided in the chapter CCF Connector for SAP R/3.

Besides the EjbTable and EjbUserManager EJB beans you must also add and deploy the following EJB beans:

JNDI Name	EJB Class Name
PurchaseRequisitionEjb	com.ibm.sap.bapi.demo.ejb.generated.PurchaseRequisitionEjb
PurchaseReqItemEjb	com.ibm.sap.bapi.demo.ejb.generated.PurchaseReqItemEjb
ProductCatalogServerEjb	com.ibm.sap.bapi.demo.purchreq.catalog.ProductCatalogServerEjb

Ensure that the EJB beans are deployed as Stateful Session Beans and the environment variables are correctly set (as described in in the chapter CCF Connector for SAP R/3).

After you have completed the above steps you can execute the aplication:

1. Select the class com.ibm.sap.bapi.demo.purchreq.ApplClient in the project Connector for SAP R/3 examples
2. Enter valid command line parameter for an SAP R/3 product catalog and its version in the properties page, for example -cat WB00000001 -catv 002
3. Compute the project path
4. Run the application

RELATED CONCEPTS

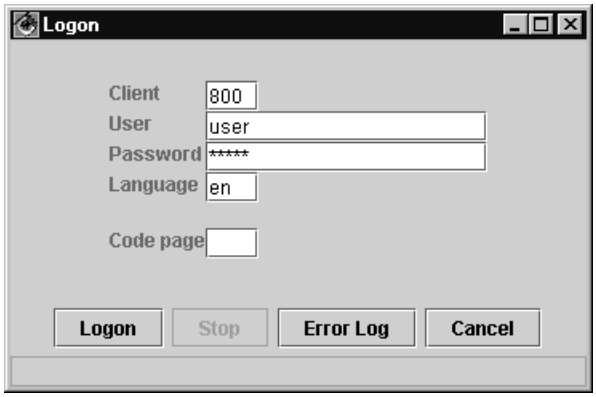
CCF Connector for SAP R/3

Providing logon information

The Purchase Requisition Application needs logon information to connect to SAP R/3. The logon data only consists of user information since connection data must be specified in the deployment descriptors of the associated EJB beans.

User info

- client (mandatory)
- user (mandatory)
- password (mandatory)
- language (optional)
- code page (optional)



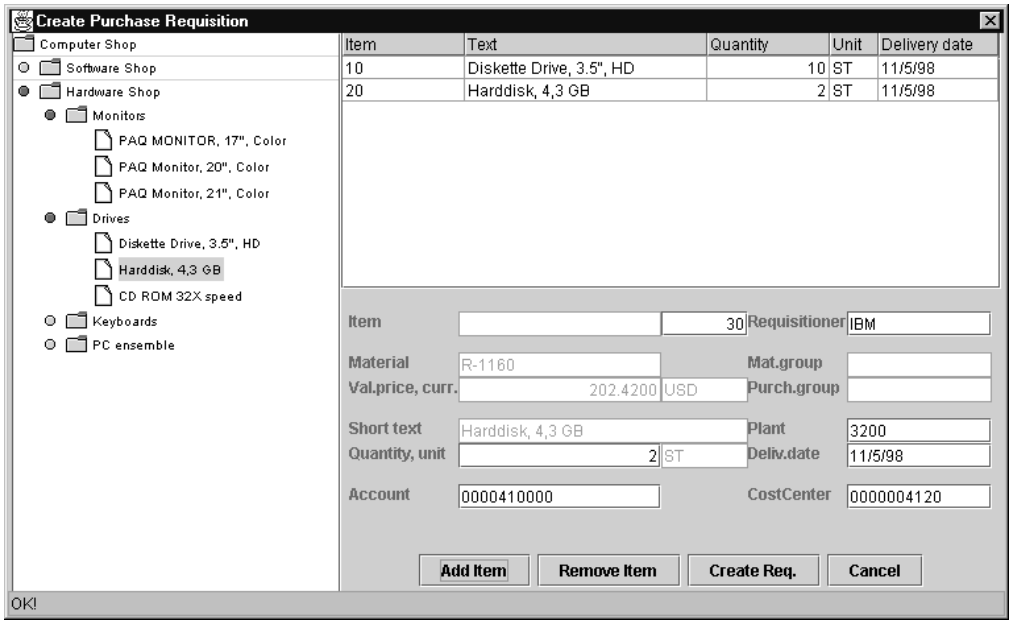
Creating or deleting purchase requisitions

To create a purchase requisition:

1. Click the Create Requisition

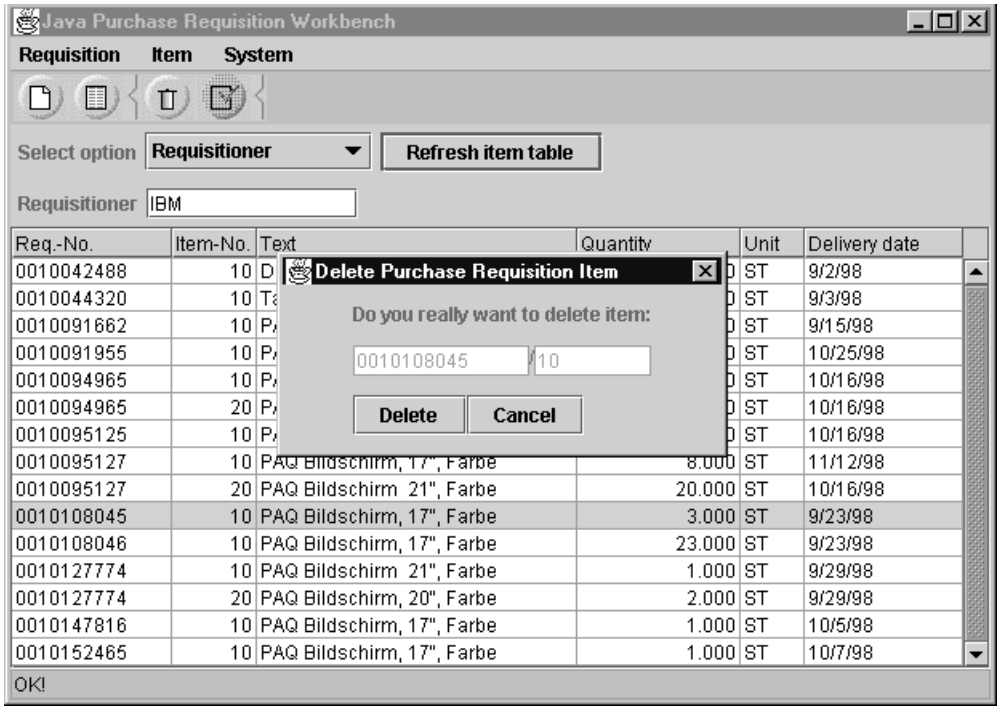
   toolbar button in the main window.
2. Select a material from the product catalog, its data is shown in the dialog on the right side.
3. Complete the data of your purchase requisition item (item number, requisitioner, plant, quantity, delivery date, account, cost center).
   - The selected material must be maintained in the specified plant.
   - Account and cost center must refer to the same company code.
4. Click **Add Item** to add the item to the table.
5. Add additional items in the same way or remove an item by selecting it and clicking **Remove Item**.
6. If the item list is completed click **Create Req.** to create the purchase requisition in SAP R/3.



To delete a purchase requisition item

1. Select the purchase requisition item
2. Click Delete Item

3. Confirm the delete request

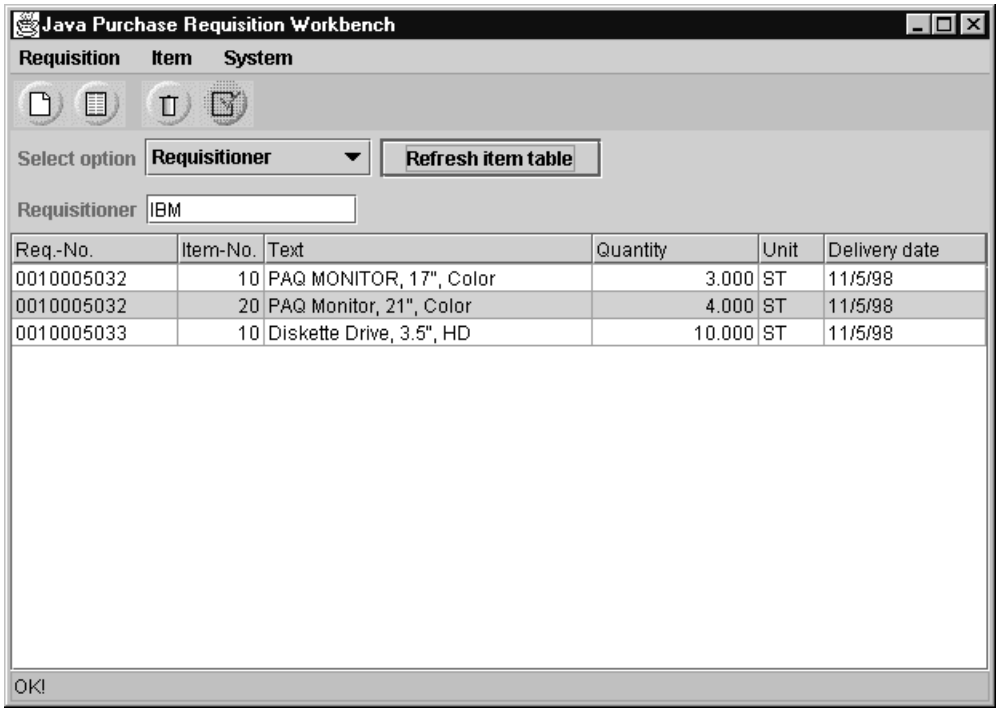


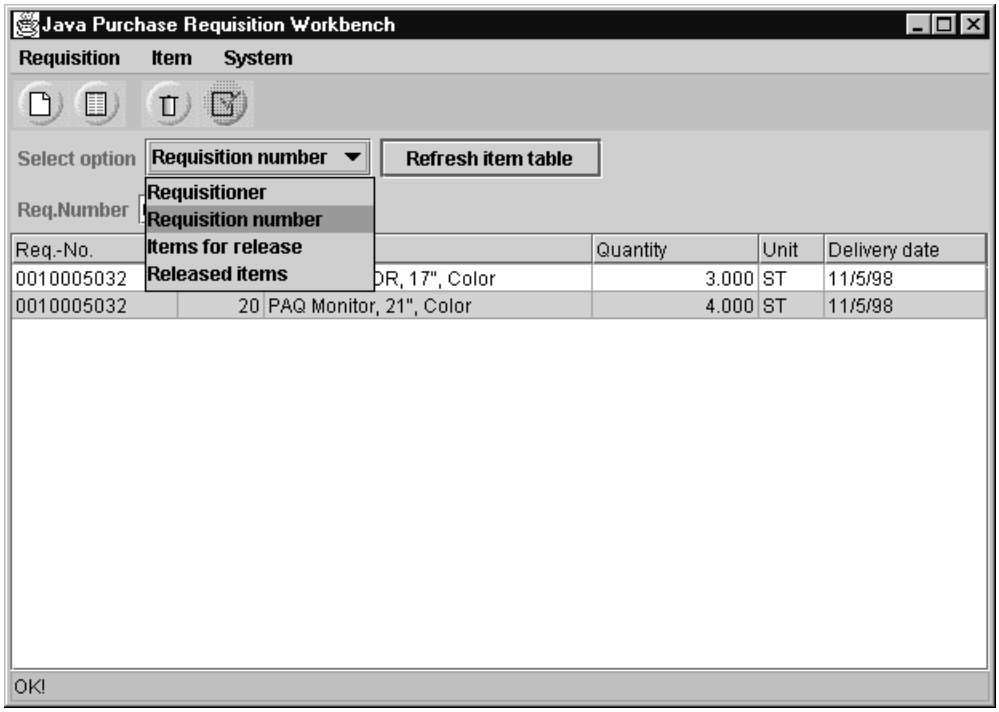
Showing information about purchase requisitions

After a successful logon the main window of the purchase requisition application is shown. A menu and a toolbar enable you to perform the base operations as described in the following topics.

Moreover, this panel allows you to display information about available purchase requisition items. You can specify the following selection criteria:

- **Requisitioner:** after refreshing the item table the purchase reqisition items for the specified requisitioner are displayed.
- **Requisition number:** only the purchase requisition for the specified requisition number is displayed.
- **Items for release:** only the purchase requisition items for the specified release group and code are displayed.
- **Released Items:** only released purchase reqisition items are displayed.





Changing purchase requisitions

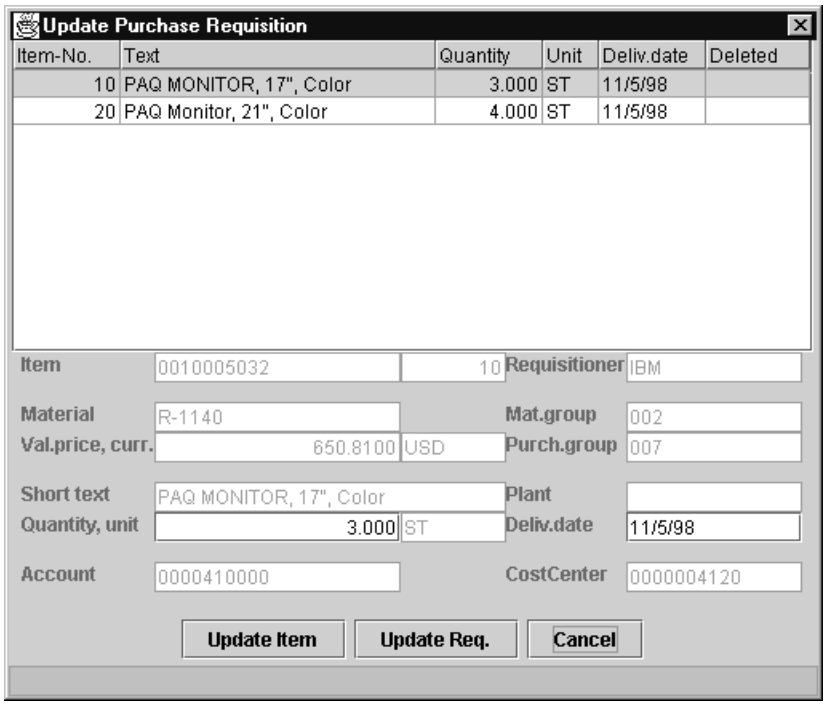
To update a purchase requisition:

1. Click the Update Item

   toolbar button in the main window.

2. Select the item you want to change, its data is shown in the update dialog.
3. Change values (only quantity and delivery date can be changed).
4. Click **Update Item** to refresh the table view.
5. Change other items in the same way.
6. Click **Update Req.** to change the purchase requisition permanently. This invokes the corresponding BAPI call.



Releasing purchase requisitions

Purchase requisitions can only be released if some preconditions are met:

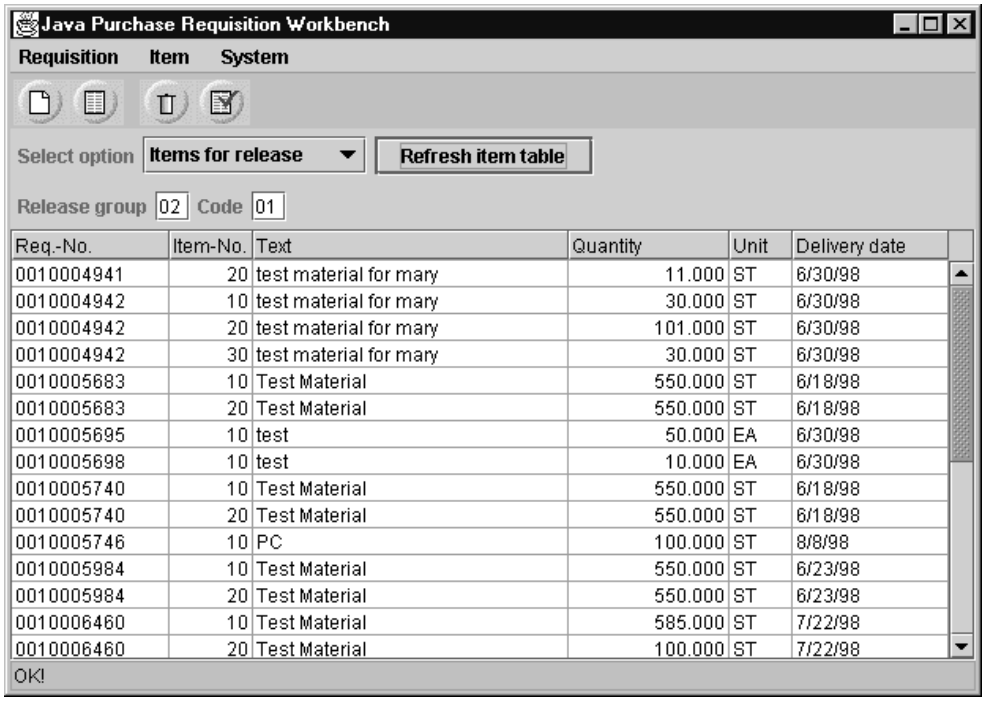
- Release conditions must be employed. A release condition determines the release strategy in accordance with which a requisition is to be released. If a requisition does not meet the conditions for a release strategy, it is automatically released for further processing.
- You need a release code and group. Ask your system administrator for this information.

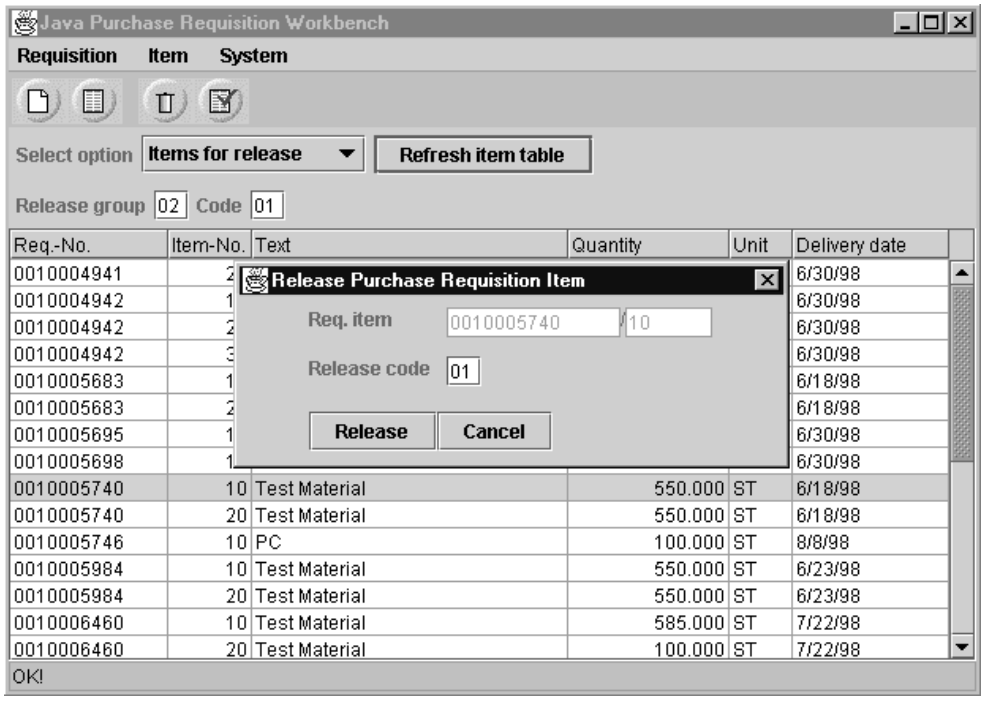
To release a purchase requisition item:

- Select option **Items for release**.
- Enter valid release group and code.
- Click **Refresh item table**.
- Select the item you want to release.
- Click the Release Item

toolbar button.

- Confirm the release request.





Sample: Receiving stock prices from web using an RFC Server

Objectives
This sample demonstrates how to install and run an RFC Server for SAP R/3.

Time required
You need approximateley 20 minutes to run this program.

Before you begin
Ensure that you have SAPGUI access to an SAP R/3 system and sufficient authorization to develop ABAP/4 programs. Ask your system administrator for the `gateway host` and `gateway service` of your SAP R/3 system. You will need this information to configure the RFC Server for SAP R/3.
If you want to use web access find out how your local machine on which the RFC Server for SAP R/3 should run is connected to the internet. If a proxy is employed find out its host name, port and if necessary user name and password.

Description
The sample allows you to receive web based stock prices in an ABAP client program. The names and the numbers of the stocks to be received are stored in a profile file on the server side. You can specify its file name or alternatively an option to receive stock prices according to a stock index (for example DAX or EUROSTOXX). If you have no web access you can receive stock prices from a related profile file stored on the server.

Instructions for running the sample
Running this sample entails the following steps:

- **Creating an RFC destination for your local machine**. This task is described in the topic ″Configuring SAP R/3 to call an RFC Server application from ABAP/4″. Choose any name for the RFC destination but remember it.
- **Creating an ABAP/4 report**. This task is described in the topic “Creating an ABAP/4 report”. Copy the ABAP source code contained in the text file `ABStockPriceSample.abap` located in the directory `<Visual Age Install Dir>\Ide\project_resources\Access Builder for SAP R3 Samples` to your newly created ABAP/4 report. Choose a report name starting with ’Y’ or ’Z’. Edit the source and change the name of the RFC destination corresponding to your RFC destination.
- **Modifying saprfc.ini file**. The `saprfc.ini` file must be located in the directory `<Visual Age Install Dir>\Ide\Program`. Therefore you can copy and rename the file `saprfc_sample.ini` located the directory `<Visual Age Install Dir>\Ide\project_resources\Access Builder for SAP R3`. Then add the following lines to it:

  ```
  DEST=ABRFCSERVER
  TYPE=R
  PROGID=my_progid
  GWHOST=my_gateway_host
  GWSERV=my_gateway_service
  RFC_TRACE=0
  ```

  Replace the my_* placeholders with your values. The value for DEST is referenced in the sample java source. Don’t change it. The PROGID value is the one specified in your RFC destination.
- **Specifying proxy info**. If you want to receive stock prices from the web, you have to provide some proxy information in a property file named `proxy.properties.` You will find the file in the directory `<Visual Age Install Dir>\Ide\project_resources\Access Builder for SAP R3`. You have to specify the following properties:
  `proxySet=true proxyHost=my_proxy proxyPort=my_proxy_port proxyAuth=my_user:my_password` Replace the my_* placeholders with your values. The colon in proxyAuth is crucial, don’t miss it. The property proxySet is a general switch. You can set it to false if a proxy is not required.

- **Checking file locations.** The following files must be located in the project's current working directory `<Visual Age Install Dir>\Ide\Project_resources\Access Builder for SAP R3 Samples`:

  ```
  dax30.csv
  estoxx.csv
  mystocks.csv
  daxvalue.csv
  proxy.properties
  ```

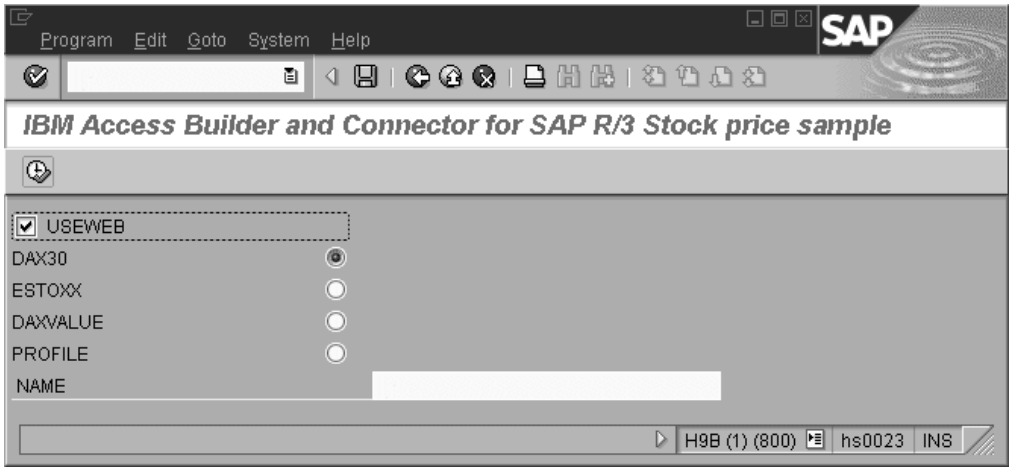
  The file `saprfc.ini` must be located in the directory `<Visual Age Install Dir>\Ide\Program`.
- **Checking project's class path.** Switch to the Workbench window and select the class StockPriceServer in package com.ibm.sap.bapi.demo.rfcserver, which contains the main entry point of the sample RFC Server. Get the property page and ensure that at least the following projects are located in the class path:

  ```
  IBM Access Builder for SAP R3 Libraries
  Infobus
  Netscape Security
  ```
- **Starting the RFC Server.** You can simply perform this by selecting the class StockPriceServer and clicking on the Run icon. The console will show you some status information when the server is running.

Now you are ready to run the ABAP/4 report. Switch to your SAPGUI session and select your ABAP/4 report in the ABAP editor. Click the Execute

or the Test

button to run the report. The following panel appears:



Specify your options as follows:

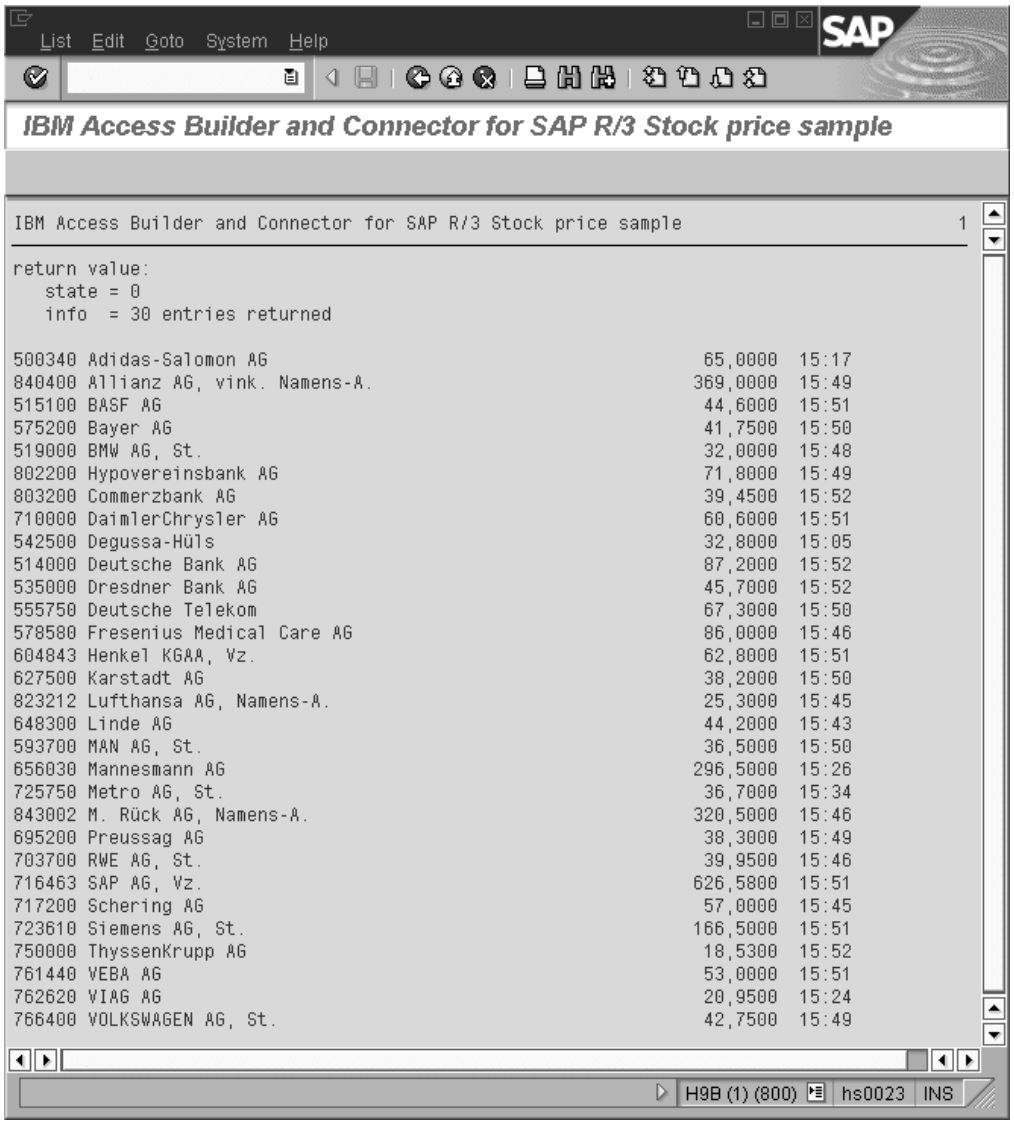
- USEWEB, indicates that you want to receive real time stock prices from web.
- DAX30, stock prices according to the DAX30 index will be received.
- ESTOXX, stock prices according to the EUROSTOXX index will be received.
- DAXVALUE, only the current DAX value will be received.
- PROFILE, stock prices according to entries added to a profile file will be received. This file is stored in csv-format (comma-separated-values) and is

located on the server side in the current directory (for example mystocks.csv). You can simply add such a file with your own stock portfolio.

- NAME, name of the profile. Omit any extension. It will be appended on the server side.

Then click the Execute

button to do the RFC call to your RFC server. You will receive a result similar to this:



RELATED CONCEPTS

RFC Server for SAP R/3

RELATED TASKS

Developing an RFC Server application
Configuring SAP R/3 to call an RFC Server application from ABAP/4
Creating an ABAP/4 report

Chapter 15. Appendix

ABAP/4 and Java data types

The run-time classes for SAP R/3 automatically map ABAP/4 data types to Java data types. When you use the generated beans you simply work with native Java types and you do not need to be aware of the mapped ABAP/4 data types. When you use the run-time classes for SAP R/3 to invoke BAPI methods dynamically you must specify the appropriate ABAP/4 data types for any Simple, Structure, or Table within the corresponding SimpleInfo and ComplexInfo classes.

The following table shows the mapping of ABAP/4 data types to Java data types:

RFCTYPE_CHAR	java.lang.String
RFCTYPE_DATE	java.util.Date
RFCTYPE_BCD	java.math.BigDecimal
RFCTYPE_TIME	java.util.Date
RFCTYPE_BYTE	byte []
RFCTYPE_NUM	java.math.BigInteger
RFCTYPE_FLOAT	double
RFCTYPE_INT	int
RFCTYPE_INT2	short
RFCTYPE_INT1	byte
RFCTYPE_STRUCTURE	com.sap.rfc.IStructure*
RFCTYPE_ABAP4OBJECT	java.lang.String**

* Tables and structures are complex data types and are mapped to instances of classes that adhere to the interfaces ITable and IStructure respectively.
** The ABAP/4 object type is usually referenced by an address and is mapped to an instance of java.lang.String which holds a text representation of the address.

The data types RFCTYPE_DATE_1, RFCTYPE_DATE_2, RFCTYPE_ITAB, RFCTYPE_IUNKNOWN, RFCTYPE_NULL, RFCTYPE_SAPAUTOMATION, RFCTYPE_STUB, RFCTYPE_WIDE_2, RFCTYPE_WIDE_4, RFCTYPE_WCHAR and RFCTYPE_WSTRING are not supported.

RELATED TASKS

Tracing the SAP connection of your application

RELATED REFERENCES

Package com.ibm.connector.sap
Package com.ibm.sap.bapi
Package com.ibm.sap.bapi.bor
Package com.ibm.sap.bapi.connectionmanager
Package com.ibm.sap.bapi.exception
Package com.ibm.sap.bapi.logon
Package com.ibm.sap.bapi.rfcserver

Package com.ibm.sap.bapi.util
Package com.ibm.sap.bapi.util.helpvalues
Package com.ibm.sap.bapi.util.logon
Package com.sap.rfc
Package com.sap.rfc.exception

Naming conventions for enterprise bean proxies

In addition to the general naming conventions, the parameter access methods are mapped according to the following naming scheme:

get methods

```
public <ParameterType> get<MethodName>_<ParameterName>() ;
```

for example `public java.lang.String getGetdetail_AccountAssignment();`

set methods

```
public void set<MethodName>_<ParameterName>(<ParameterType>
<ParameterName>) ;
```

for example `public void setGetdetail_AccountAssignment(java.lang.String accountAssignment)`

Generated source files

To generate the enterprise bean proxies for Business Objects, the Access Builder for SAP R/3 produces a number of Java files. The produced files follow the naming schemes below.

<SapBoName>EjbBean.index
The <SapBoName>EjbBean.index file contains a list of file names of all Java classes belonging to this proxy EJB bean.

<SapBoName>EjbBean.java
The <SapBoName>EjbBean.java file contains a class definition of the proxy EJB bean containing all methods belonging to this Business Object.

<SapBoName>Ejb.java
The <SapBoName>Ejb.java file contains the remote interface definition for the proxy EJB bean.

<SapBoName>EjbHome.java
The <SapBoName>EjbHome.java file contains the home interface definition for the proxy EJB bean.

<RFC_MODULE_NAME>.java
The <RFC_MODULE_NAME>.java files contain the BAPI command objects which the proxy EJB bean uses under the covers to invoke a specific BAPI method. There is one <RFC_MODULE_NAME>.java file generated for each method of the SAP Business Object.

<RfcStructureName>Structure.java
Each <RfcStructureName>Structure.java file contains the class definition of an RFC structure used as a method parameter. This class contains methods for accessing all fields of the structure. It depends on the respective BAPI method which structure proxy files are generated. <RfcStructureName> is a placeholder for the SAP R/3-internal name of the structure.

<RfcTableName>Table.java

<RfcTableName>TableRow.java

<RfcTableName>EjbTable.java

Each <RfcTableName>Table.java file contain the class definition of an RFC table parameter used as a method parameter. This class contains methods for navigating through the tables. For each <RfcTableName>Table.java file there exists one <RfcTableName>TableRow.java file which contains a class definition to access the fields of a table row and one <RfcTableName>EjbTable.java file for smart table caching.

It depends on the respective BAPI method which table proxy files are generated. <RfcTableName> is a placeholder for the SAP R/3-internal name of the table.

RELATED CONCEPTS

Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules

RELATED REFERENCES

Naming conventions for generated classes
Naming conventions for proxy beans
Naming conventions for RFC module proxy beans
BO and RFC proxy beans
Package com.ibm.connector.sap
Package com.ibm.sap.bapi
Package com.ibm.sap.bapi.bor
Package com.ibm.sap.bapi.connectionmanager
Package com.ibm.sap.bapi.exception
Package com.ibm.sap.bapi.logon
Package com.ibm.sap.bapi.rfcserver
Package com.ibm.sap.bapi.util
Package com.ibm.sap.bapi.util.helpvalues
Package com.ibm.sap.bapi.util.logon
Package com.sap.rfc
Package com.sap.rfc.exception

Naming conventions for proxy beans

When generating proxy beans for Business Objects the Access Builder for SAP R/3 produces a number of Java files. The produced files follow the naming schemes below:

<SapBoName>.index

The <SapBoName>.index file contains a list of file names of all Java classes belonging to this Business Object.

<SapBoName>.java

The <SapBoName>.java file contains a class definition of the proxy containing all methods belonging to this Business Object.

<SapBoName><MethodName>Params.java

The <SapBoName><MethodName>Params.java files contain the parameter

container class definition for a specific method of the business object proxy. For each method of the SAP BO one parameter container class is generated.

<RfcStructureName>Structure.java
Each <RfcStructureName>Structure.java file contains the class definition of an RFC structure used as a method parameter. This class contains methods for accessing all fields of the structure. It depends on the respective BAPI method which structure proxy files are generated. <RfcStructureName> is a placeholder for the SAP R/3-internal name of the structure.

<RfcTableName>Table.java
<RfcTableName>TableRow.java
Each <RfcTableName>Table.java file contains the class definition of an RFC table parameter used as a method parameter. This class contains methods for navigating through the tables. For each <RfcTableName>Table.java file there exists one <RfcTableName>TableRow.java file which contains a class definition to access the fields of a table row. It depends on the respective BAPI method which table proxy files are generated.
<RfcTableName> is a placeholder for the SAP R/3-internal name of the table.

RELATED CONCEPTS

Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules

RELATED REFERENCES

Naming conventions for generated classes
Naming conventions for enterprise bean proxies
Naming conventions for RFC module proxy beans
BO and RFC proxy beans
Package com.ibm.connector.sap
Package com.ibm.sap.bapi
Package com.ibm.sap.bapi.bor
Package com.ibm.sap.bapi.connectionmanager
Package com.ibm.sap.bapi.exception
Package com.ibm.sap.bapi.logon
Package com.ibm.sap.bapi.rfcserver
Package com.ibm.sap.bapi.util
Package com.ibm.sap.bapi.util.helpvalues
Package com.ibm.sap.bapi.util.logon
Package com.sap.rfc
Package com.sap.rfc.exception

Naming conventions for RFC module proxy beans

When generating RFC module proxy beans for RFC modules and BAPIs the Access Builder for SAP R/3 produces a number of Java files. The produced files follow the naming schemes below.

<RFC_MODULE_NAME>.index
The <RFC_MODULE_NAME>.index file contains a list of all file names belonging to this RFC proxy.

<RFC_MODULE_NAME>.java
The <RFC_MODULE_NAME>.java file contains a class definition of the RFC proxy bean containing all parameters belonging to the respective remote function call or BAPI, as well as access methods for the parameters, and an execute method for invoking the RFC module on the SAP R/3 system.

<RfcStructureName>Structure.java
Each <RfcStructureName>Structure.java file contains the class definition of an RFC structure used as a method parameter. This class contains methods for accessing all fields of the structure. It depends on the respective BAPI method which structure proxy files are generated. <RfcStructureName> is a placeholder for the SAP R/3-internal name of the structure.

<RfcTableName>Table.java
<RfcTableName>TableRow.java
Each <RfcTableName>Table.java file contains the class definition of an RFC table parameter used as a method parameter. This class contains methods for navigating through the tables. For each <RfcTableName>Table.java file there exists one <RfcTableName>TableRow.java file which contains a class definition to access the fields of a table row. It depends on the respective BAPI method which table proxy files are generated.
<RfcTableName> is a placeholder for the SAP R/3-internal name of the table.

RELATED CONCEPTS

Proxy beans for Business Objects
Enterprise bean proxies for Business Objects
Proxy beans for RFC modules

RELATED TASKS

Generating proxy beans for SAP Business Objects and RFC modules

RELATED REFERENCES

Naming conventions for generated classes
Naming conventions for proxy beans
Naming conventions for enterprise bean proxies
BO and RFC proxy beans
Package com.ibm.connector.sap
Package com.ibm.sap.bapi
Package com.ibm.sap.bapi.bor
Package com.ibm.sap.bapi.connectionmanager
Package com.ibm.sap.bapi.exception
Package com.ibm.sap.bapi.logon
Package com.ibm.sap.bapi.rfcserver
Package com.ibm.sap.bapi.util
Package com.ibm.sap.bapi.util.helpvalues
Package com.ibm.sap.bapi.util.logon
Package com.sap.rfc
Package com.sap.rfc.exception

Notices

Note to U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.
IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

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

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.
This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Lab Director
IBM Canada Ltd.
1150 Eglinton Avenue East
Toronto, Ontario M3C 1H7
Canada

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.
The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

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 and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:
(C) (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. (C); Copyright IBM Corp. 1997, 2000. All rights reserved.

Programming interface information

Programming interface information is intended to help you create application software using this program.

General-use programming interfaces allow the customer to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

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

- AIX
- AS/400
- DB2
- CICS
- CICS/ESA
- IBM
- IMS
- Language Environment
- MQSeries
- Network Station
- OS/2
- OS/390
- OS/400
- RS/6000
- S/390
- VisualAge
- VTAM
- WebSphere

Lotus, Lotus Notes and Domino are trademarks or registered trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Tivoli Enterprise Console and Tivoli Module Designer are trademarks of Tivoli Systems Inc. in the United States, or other countries, or both.

Encina and DCE Encina Lightweight Client are trademarks of Transarc Corporation in the United States, or other countries, or both.

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

ActiveX, Microsoft, SourceSafe, Visual C++, Visual SourceSafe, Windows, Windows NT, Win32, Win32s and the Windows logo are trademarks or registered trademarks of Microsoft Corporation in the United States, or other countries, or both.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Intel and Pentium are trademarks of Intel Corporation in the United States, or other countries, or both.

Other company, product, and service names, which may be denoted by a double asterisk(**), may be trademarks or service marks of others.