SAP JCo 2.x-3.0
(Standalone)
Migration Guide

Release 2.x-3.0

SAP
Contents

Contents ...............................................................................................................................2
SAP JCo 3.0 (Standalone) Client...........................................................................................3
Features ............................................................................................................................3
Required Changes.............................................................................................................4
General Coding Changes...................................................................................................4
  Renaming of Java Packages.......................................................................................4
  Sequence of setValue() Operations.............................................................................5
  Replacing JCo Classes...............................................................................................5
Changes to the Client Role ..................................................................................................7
  Adapting Connection Management .............................................................................7
  Stateful Connections.................................................................................................10
  Exception Handling...................................................................................................11
  Changes to the Metadata Model ...............................................................................12
Changes to the Server Role...............................................................................................19
IDoc Support....................................................................................................................20
New Method Names in SAP JCo 3.0................................................................................20
SAP JCo 3.0 (Standalone) Client

Features

SAP JCo 3.0 is the recent version.

With version 3.0 a couple of SAP JCo functions have been enhanced or rebuilt. Version 3.0 offers the following benefits:

- JCo 3.0 offers an increased supportability because it complies with the SAP supportability standard
- The security of data transmission has been improved
- JCo 3.0 is more robust because it offers an enhanced resource management
- Metadata handling has been improved through
  - Reduced memory consumption by changed class design
  - Solved difficulties with Unicode/Non-Unicode communication
- The separation of application and technical concerns within runtime environment of JCo 3.0 offers additional benefits:
  - The application can now concentrate on business logic
  - Connection and repository (metadata) management is done within the JCo layer
  - The runtime environment integration is separated from JCo applications
- JCo 3.0 permits a uniform behavior of JCo applications in a specific runtime environment because it now offers
  - A single location for configuration data
  - A single resource management for all applications
- Version 3.0 supports several runtime environments (AS Java, JCo standalone, VMC)
- Both client and server are based on a more robust programming model.
- The RFC Library has been replaced by a Java RFC implementation, thus
  - reducing the risk of crashes (more robustness)
  - drastically reduced JNI overhead
- New features in JCo 3.0 are:
  - Stateful server communication
  - Monitoring capabilities
  - Full support for the ABAP statement IS SUPPLIED
  - Support for data type DECFLOAT
  - Support for bgRFC communication
  - Native support for CHANGING parameters: CHANGING parameters are no longer emulated by being both an IMPORT and an EXPORT parameter

New features will be available in JCo 3.0 only
**Required Changes**

Some clear alterations have been made to JCo API using SAP JCo Standalone Version 3.0. You will need to make some changes to the coding when switching from an older JCo version (2.x) to JCo 3.0.

Below you will find the necessary information about the type of changes that have been made and actions required for the migration.

Modifications are possibly required in the following areas:

- General changes
- Connection management
- Exception handling
- Metadata model
- Server adaptations

**Migration Scenarios**

The scope of the changes to be made depends largely on the application scenario used. There are basically three possible scenarios:

**Client Scenarios**

- Your application communicates as a client with an ABAP server. In this case you need to make changes in areas.
- Your client application has hard-coded metadata or metadata-relevant APIs. In this case you need to make changes in areas.

**Server Scenario**

- Your application was developed on a Java application server and executes server functions, that is it processes ABAP server calls.

Additionally, there are some general changes to be done regardless of the scenario you use.

**More Information**

For more detailed information on each of the changes to be made, see:

- General Coding Changes
- Changes to the Client Role
- Changes to the Server Role

**General Coding Changes**

**Renaming of Java Packages**

The SAP JCo standalone version 3.0 derives from SAP JCo as integrated component of SAP NetWeaver 7.1. Because both the old and the new JCo API is available there, it was necessary to change the naming convention for the new API Java packages to avoid redundancies.

The previous convention (which is still valid for the old API) `com.sap.mw.jco` has been changed to `com.sap.conn.jco`.
**Sequence of setValue() Operations**

The sequence of values for setValue() operations has been changed to adapt it to the more common convention.

The majority of these changes are recognized by the compiler, but unfortunately not those that occur the most frequently (String, int <-> int,String).

- For all imports with the new JCo API, change the package names from `com.sap.mw.jco*` to `com.sap.conn.jco*`.
- Check the sequence of the String- and int-values of all setValue() operations and adapt the value sequence is required.

**Example**

The changed value sequence for setValue() operations is made clear in the examples below:

**JCo 2.x**

```java
JCO.Structure myStruct=new JCO.Structure(repository.getStructureDefinition("FOOBAR"));
myStruct.setValue("Madonna", "ARTIST");
myStruct.setValue("Lucky Star", 1); // 1 is index of field SONG
myStruct.setValue(3.87, "LENGTH");
```

**JCo 3.0**

```java
JCoStructure myStruct=
JCo.createStructure(repository.getStructureDefinition("FOOBAR"))
;
myStruct.setValue(“ARTIST”, “Madonna”);
myStruct.setValue(1, “Lucks Star”); // 1 is index of field SONG
myStruct.setValue(“LENGTH”, 3.87);
```

**Replacing JCo Classes**

The new JCo API replaces the previous JCo classes (for example, JCO.Function, JCO.Table and so on) with interfaces in order to standardize access to the API for platform-specific use.

The naming of the newly-defined interfaces is very similar to that of the former JCo classes. Like the former classes, all the important interfaces have the same prefix, not in the form of an outer class, but simply as a naming convention. "JCO.-" becomes "JCo-".

**Procedure**

Replace all the former JCo classes with the corresponding interface.

**Example**

The following example shows JCo classes being replaced by interfaces:
JCo 2.x

```java
IRepository repository = JCO.createRepository("REPO", "SAP");
IFunctionTemplate template = repository.getFunctionTemplate("FUNCTION_MODULE");
IMetaData structureDefinition = repository.getStructureDefinition("MY_STRUCT");

JCO.Structure structure = JCO.createStructure(structureDefinition);
structure.setValue("Hello World", "TITLE");
structure.setValue("Good Morning", 1);

JCO.FieldIterator iterator = new JCO.FieldIterator(structure);
while (iterator.hasNextFields()) {
    JCO.Field field = iterator.nextField();
    System.out.println("FieldValue: "+field.getString());
}

JCO.Function function = template.getFunction();
JCO.ParameterList tables = function.getTablesParameterList();

JCO.Table tracks = tables.getTable("TRACKS");
if (!tracks.isEmpty()) {
    for (int i=0; i<tracks.getNumRows(); i++) {
        tracks.setRow(i);
        System.out.println(tracks.getString("ARTIST") + " : " +
                           tracks.getString("SONG"));
    }
}

JCO.AbapException[] exceptions = function.getExceptionList();
if (exceptions!=null) {
    for (int i=0; i<exceptions.length(); i++) {
        System.out.println("ExceptionName: "+exceptions[i].getKey());
    }
}
```

JCo 3.0

```java
JCoRepository repository = destination.getRepository();
JCoFunctionTemplate template = repository.getFunctionTemplate("FUNCTION_MODULE");
JCoRecordMetaData structureDefinition = repository.getStructureDefinition("MY_STRUCT");
```
JCoStructure structure=JCo.createStructure(structureDefinition);
structure.setValue("TITLE", "Hello World");
structure.setValue(1, "Good Morning");

JCoFieldIterator iterator=structure.getFieldIterator();
while(iterator.hasNextField())
{
    JCoField field=iterator.nextField();
    System.out.println("FieldValue: "+field.getString());
}

JCoFunction function = template.getFunction();
JCoParameterList tables=function.getTablesParameterList();

JCoTable tracks = tables.getTable("TRACKS");
if (!tracks.isEmpty())
{
    for (int i=0; i<tracks.getNumRows(); i++)
    {
        tracks.setRow(i);
        System.out.println(tracks.getString("ARTIST") + ":
        " +
        tracks.getString("SONG"));
    }
}

AbapException[] exceptions=function.getExceptionList();
if (exceptions!=null)
    for (int i=0; i<exceptions.length(); i++)
        System.out.println("ExceptionName: "+exceptions[i].getKey());

Changes to the Client Role

Adapting Connection Management

Connection management has been modified completely for the new JCo API. Until now the connection setup at the client side has used different JCo classes as a direct connection or Connection Pool.

In the new JCo API, connection management is no longer a task for the client application, but is managed centrally by a destination model.

A Destination Manager transfers a logical destination to the client, who in turn can trigger a certain function to be carried out on this destination.

Using this connection model you can avoid any high level configuration.

Procedure

Perform the required changes in the code to adapt correctly to the connection management.
Example
The following example shows the changes in the code at connection setup:

```java
JCo.addClientPool("FOO", 10, "000", "hugo", "******", "EN", "appserver", 
"00");
JCO.Repository repository = JCO.createRepository("MyRepository", SID);

IFunctionTemplate template = 
repository.getFunctionTemplate("BAPI_COMPANY_GETLIST");

if(template != null)
{
    JCO.Client client = null;
    try
    {
        JCO.Function function = template.getFunction();
        client = JCO.getClient("FOO");
        client.execute(function);

        JCO.ParameterList exports=function.getExportParameterList();
        JCO.ParameterList tables=function.getTablesParameterList();

        JCO.Structure bapiReturn = exports.getStructure("RETURN");
        System.out.println("BAPI_COMPANY_GETLIST RETURN: " + 
        bapiReturn.getString("MESSAGE"));

        JCO.Table companies = tables.getTable("COMPANY_LIST");
        if (companies.getNumRows()>0)
        {
            for (int i=0; i<companies.getNumRows(); i++)
            {
                companies.setRow(i);
                System.out.println(companies.getString("COMPANY")+
                ":  "+companies.getString("NAME1"));
            }
        }
    }
    catch (JCO.AbapException ex)
    {
        System.out.println("Caught a function module exception: \n" + ex);
    }
    catch (JCO.Exception ex)
    {
        System.out.println("Caught an exception: \n" + ex);
    }
    finally
```

```
{
    JCO.releaseClient(client);
}
```

**JCo 3.0**

```java
JCoDestination foo = JCoDestinationManager.getDestination("FOO");
JCoRepository repository = foo.getRepository();

JCoFunctionTemplate template =
    repository.getFunctionTemplate("BAPI_COMPANY_GETLIST");
if(template != null)
{
    try
    {
        JCoFunction function = template.getFunction();
        function.execute(foo);

        JCoParameterList exports=function.getExportParameterList();
        JCoParameterList tables=function.getTablesParameterList();

        JCoStructure bapiReturn = exports.getStructure("RETURN");

        System.out.println("BAPI_COMPANY_GETLIST RETURN: " +
                           bapiReturn.getString("MESSAGE"));

        JCoTable companies = tables.getTable("COMPANY_LIST");

        if (companies.getNumRows()>0)
        {
            for (int i=0; i<companies.getNumRows(); i++)
            {
                companies.setRow(i);
                System.out.println(companies.getString("COMPANY") +
                                    ": " +companies.getString("NAME1"));
            }
        }
    }
    catch (AbapException ex)
    {
        System.out.println("Caught a function module exception: \n" + ex);
    }
    catch (JCoException ex)
    {
        System.out.println("Caught an exception: \n" + ex);
    }
```
The JCo Runtime receives the connection parameters through the interface DestinationDataProvider. You can register an environment-specific implementation of DestinationDataProvider to enable integration into the application environment. You will find detailed information in the SAP JCo 3.0 documentation.

Stateful Connections

In SAP JCo Release 2.x any function call was executed in a stateful connection by default. For this reason it was not necessary to define a stateful connection explicitly.

As of JCo 3.0 the statements JCoContext.begin(destination) and JCoContext.end(destination) are required for a stateful connection.

Consequently the application only has to define the scope for the stateful communication. JCo runtime takes care of the connection management: all calls to a particular destination between begin and end will be invoked on the same connection.

Example

The following example shows the code differences between 2.x and 3.0:

JCo 2.x

```java
JCO.Function bapiFunction1 = ...
JCO.Function bapiFunction2 = ...
JCO.Function bapiTransactionCommit = ...
JCO.Function bapiTransactionRollback = ...
JCO.Client client = ...
try {
  try {
    client.execute(bapiFunction1);
    client.execute(bapiFunction2);
    client.execute(bapiTransactionCommit);
  }
  catch(JCO.AbapException ex) {
    client.execute(bapiTransactionRollback);
  }
} catch(JCO.Exception ex) {
  [...]}
```

JCo 3.0

```java
}]
```
JCoDestination destination = ...
JCoFunction bapiFunction1 = ...
JCoFunction bapiFunction2 = ...
JCoFunction bapiTransactionCommit = ...
JCoFunction bapiTransactionRollback = ...
	ry {
    JCoContext.begin(destination);
    try {
        bapiFunction1.execute(destination);
        bapiFunction2.execute(destination);
        bapiTransactionCommit.execute(destination);
    } catch(AbapException ex) {
        bapiTransactionRollback.execute(destination);
    }
} catch(JCoException ex) {
    [...]
} finally {
    JCoContext.end(destination);
}

**Exception Handling**

As a result of changes to exception handling, instead of enhancing of
java.lang.RuntimeException, java.lang.Exception has been introduced as a new
parent class.

When using RuntimeExceptions, it was not mandatory that exceptions be added to a throws
clause. However, when using 'normal' exceptions you must always decide whether an
exception is to be handled explicitly or if it is to be added to the throws clause.

**Procedure**

To adapt your code, either set try-catch [-finally] blocks, or add your exceptions to a
throw clause.

**JCo 2.x**

```java
public void executeFunction(JCO.Function function, String where) {
    JCO.Client client = JCO.getClient(where);
    client.execute(function);
}
```

**JCo 3.0 (Option 1)**
Changes to the Metadata Model

Some changes have also been made with the new JCo API in the area of the metadata model.

Separating Metadata Types

The most important new feature is the separation of the metadata types.

JCo function parameter lists as well as JCo records like tables and structures were previously assigned to a common metadata class. In the new API, record and parameter list metadata is assigned to two different metadata types in order to avoid any confusion. As a result of this separation, each record and parameterList has a reference to its own metadata object.

Enhancement of Metadata Objects

With the new JCo API, metadata objects contain both Unicode and non-Unicode information. This makes the usage of metadata easier and helps to avoid any errors when configuring the Repository.
It is of course still recommended that you exercise caution with the general configuration of the Repository. For example, you should be sure to avoid using the same name for different functions in different systems.

Reducing Metadata Objects in the Memory

Only one object is stored in the memory for a particular metadata type. This object referenced by the corresponding function parameter lists or tables. The result is a reduced memory consumption, particularly for nested tables.

Consequences

- A consequence of these changes is that specific metadata operations (for example, addinfo()) are no longer usable for the metadata of record and list objects, if it is already in use.

- Also, the ParameterList.appendValue() method no longer exists.

- However, it is still possible to create JCoFunction and its JcoParameterLists on the fly.

- Additionally, it is now possible to lock a metadata object after you have created it to avoid any unwanted changes.

Note:

The following activities are only relevant if you want to use an application with hard-coded metadata or with metadata-relevant APIs.

Examples

The following examples show the differences between JCo 2.x and 3.0 in three different areas:

- Retrieving metadata when only the data container is available
- Creating a parameterList on the fly.
- Setting up hard-coded metadata

Retrieving Metadata

JCo 2.x

```java
JCO.Function function = template.getFunction();

JCO.ParameterList exports=function.getExportParameterList();
for (int i=0; i<exports.getFieldCount(); i++)
    System.out.println("Parameter: " + exports.getName(i) + " length " + exports.getLength(i) + " is " + (exports.isOptional(i)?"not ":"not ") + "optional");

JCO.ParameterList tables=function.getTableParameterList();
JCO.Table theTable = tables.getTable("THE_TABLE");
for (int i=0; i<theTable.getFieldCount(); i++)
    System.out.println("Field: " + theTable.getName(i) + " length " + theTable.getLength(i) + " offset " + theTable.getOffset(i));
```
JCo 3.0

```
JCoFunction function = template.getFunction();
JCoParameterList exports = function.getExportParameterList();
JCoListMetaData listMetaData = exports.getListMetaData();
for (int i = 0; i < listMetaData.getFieldCount(); i++)
    System.out.println("Parameter: " + listMetaData.getName(i) +
        " length " + listMetaData.getByteLength(i) +
        " is " + (listMetaData.isOptional(i) ? "" : "not ") +
        "optional");

JCoParameterList tables = function.getTableParameterList();
JCoTable theTable = tables.getTable("THE_TABLE");
JCoRecordMetaData recordMetaData = theTable.getRecordMetaData();
for (int i = 0; i < recordMetaData.getFieldCount(); i++)
    System.out.println("Field: " + recordMetaData.getName(i) +
        " length " + recordMetaData.getByteLength(i) +
        " offset " + recordMetaData.getByteOffset(i));
System.out.println("Table length: " + recordMetaData.getRecordLength());
```

ParameterList on the fly

JCo 2.x

```
JCO.ParameterList imports = new JCO.ParameterList();

//no exports, tables and exceptions for this function
JCO.Function function = new JCO.Function("JCO_HELLO_WORLD", imports, null, null);
imports.appendValue("PARAM", JCO.TYPE_CHAR, 30, "Hello World!");
```

JCo 3.0

```
JCoListMetaData importsMeta = JCo.createListMetaData("IMPORT");
importsMeta.add("PARAM", JCoMetaData.TYPE_CHAR, 30, 60, 0, null, 0, null, 0, null, null);
importsMeta.lock();

//no exports, changings, tables and exceptions for this function
JCoFunction function = JCo.createFunctionTemplate("JCO_HELLO_WORLD",
```
imports, null, null, null, null).getFunction();

JCoParameterList imports = function.getImportParameterList();
imports.setValue("PARAM", "Hello World!");

**Hard-Coded Metadata**

**JCo 2.x**

JCO.MetaData bapiReturnUC = new JCO.MetaData("BAPIRETURN", 9);
JCO.MetaData bapiReturn = new JCO.MetaData("BAPIRETURN", 9);

JCO.MetaData companyListUC = new JCO.MetaData("BAPI0014_1", 2);
JCO.MetaData companyList = new JCO.MetaData("BAPI0014_1", 2);

bapiReturnUC.addInfo("TYPE", IMetaData.TYPE_CHAR, 1, 2, 0, 0, null, null, 0, null, null);
bapiReturnUC.addInfo("CODE", IMetaData.TYPE_CHAR, 5, 10, 2, 0, null, null, 0, null, null);
bapiReturnUC.addInfo("MESSAGE", IMetaData.TYPE_CHAR, 220, 440, 12, 0, null, null, 0, null, null);
bapiReturnUC.addInfo("LOG_NO", IMetaData.TYPE_NUM, 20, 40, 452, 0, null, null, 0, null, null);
bapiReturnUC.addInfo("LOG_MSG_NO", IMetaData.TYPE_CHAR, 6, 12, 492, 0, null, null, 0, null, null);
bapiReturnUC.addInfo("MESSAGE_V1", IMetaData.TYPE_CHAR, 50, 100, 504, 0, null, null, 0, null, null);
bapiReturnUC.addInfo("MESSAGE_V2", IMetaData.TYPE_CHAR, 50, 100, 604, 0, null, null, 0, null, null);
bapiReturnUC.addInfo("MESSAGE_V3", IMetaData.TYPE_CHAR, 50, 100, 704, 0, null, null, 0, null, null);
bapiReturnUC.addInfo("MESSAGE_V4", IMetaData.TYPE_CHAR, 50, 100, 804, 0, null, null, 0, null, null);
bapiReturnUC.setTabLength(904);

bapiReturn.addInfo("TYPE", IMetaData.TYPE_CHAR, 1, 1, 0, 0, null, null, 0, null, null);
bapiReturn.addInfo("CODE", IMetaData.TYPE_CHAR, 5, 5, 1, 0, null, null, 0, null, null);
null, null, 0, null, null);

bapiReturn.addInfo("MESSAGE", IMetaData.TYPE_CHAR, 220, 220, 6, 0, null, null, 0, null, null);

bapiReturn.addInfo("LOG_NO", IMetaData.TYPE_NUM, 20, 20, 226, 0, null, null, 0, null, null);

bapiReturn.addInfo("LOG_MSG_NO", IMetaData.TYPE_CHAR, 6, 6, 246, 0, null, null, 0, null, null);

bapiReturn.addInfo("MESSAGE_V1", IMetaData.TYPE_CHAR, 50, 50, 252, 0, null, null, 0, null, null);

bapiReturn.addInfo("MESSAGE_V2", IMetaData.TYPE_CHAR, 50, 50, 302, 0, null, null, 0, null, null);

bapiReturn.addInfo("MESSAGE_V3", IMetaData.TYPE_CHAR, 50, 50, 352, 0, null, null, 0, null, null);

bapiReturn.addInfo("MESSAGE_V4", IMetaData.TYPE_CHAR, 50, 50, 402, 0, null, null, 0, null, null);

bapiReturn.setTabLength(452);

companyListUC.addInfo("COMPANY", IMetaData.TYPE_CHAR, 6, 12, 0, 0, null, null, 0, null, null);

companyListUC.addInfo("NAME1", IMetaData.TYPE_CHAR, 30, 60, 12, 0, null, null, 0, null, null);

companyListUC.setTabLength(72);

companyList.addInfo("COMPANY", IMetaData.TYPE_CHAR, 6, 6, 0, 0, null, null, 0, null, null);

companyList.addInfo("NAME1", IMetaData.TYPE_CHAR, 30, 30, 6, 0, null, null, 0, null, null);

companyList.setTabLength(36);

JCO.MetaData exportsMetaUC=new JCO.MetaData("EXPORTS", 1);
exportsMetaUC.addInfo("RETURN", IMetaData.TYPE_STRUCTURE, 452, 904, 0, 0, null, null, 0, bapiReturnUC, null);

JCO.MetaData exportsMeta=new JCO.MetaData("EXPORTS", 1);
exportsMeta.addInfo("RETURN", IMetaData.TYPE_STRUCTURE, 452, 452, 0, 0, null, null, 0, bapiReturn, null);

JCO.MetaData tablesMetaUC=new JCO.MetaData("TABLES", 1);
tablesMetaUC.addInfo("COMPANY_LIST", IMetaData.TYPE_TABLE, 36, 72, 0, 0,
null, null, 0, companyListUC, null);

JCO_MetaData tablesMeta=new JCO_MetaData("TABLES", 1);
tablesMeta.addInfo("COMPANY_LIST", IMetaData.TYPE_TABLE, 36, 36, 0, 0, null, null, 0, companyList, null);

JCO_Client client = JCOgetClient("FOO");
try
{
    client.connect();
    boolean isUnicode = client.getAttributes().getPartnerBytesPerChar()==2;
    JCO_Function bapiCompanyGetList=null;
    if (isUnicode)
        bapiCompanyGetList=JCO.createFunction("BAPI_COMPANY_GETLIST", null,
JCO.createParameterList(exportsMetaUC),
JCO.createParameterList(tablesMetaUC));
    else
        bapiCompanyGetList=JCO.createFunction("BAPI_COMPANY_GETLIST", null,
JCO.createParameterList(exportsMeta),
JCO.createParameterList(tablesMeta));
    client.execute(bapiCompanyGetList);
    JCO_ParameterList tables= bapiCompanyGetList.getTableParameterList();
    JCO_Table companies=tables.getTable("COMPANY_LIST");
    for (int i=0; i<companies.getNumRows(); i++)
    {
        companies.setRow(i);
        System.out.println(companies.getString("COMPANY")+
            ": " +companies.getString("NAME1");
    }
} catch(Exception e)
{
    System.err.println("Exception occurred: "+e.toString());
} finally
{
    JCO.releaseClient(client);
}

JCo 3.0

JCoRecordMetaData bapiReturn = JCo.createRecordMetaData("BAPIRETURN", 9);
JCoRecordMetaData companyList = JCo.createRecordMetaData("BAPI0014_1", 2);

bapiReturn.add("TYPE", JCoMetaData.TYPE_CHAR, 1, 0, 2, 0, 0, null, null, null);

bapiReturn.add("CODE", JCoMetaData.TYPE_CHAR, 5, 1, 10, 2, 0, null, null, null);

bapiReturn.add("MESSAGE", JCoMetaData.TYPE_CHAR, 220, 6, 440, 12, 0, null, null, null);

bapiReturn.add("LOG_NO", JCoMetaData.TYPE_NUM, 20, 226, 40, 452, 0, null, null, null);

bapiReturn.add("LOG_MSG_NO", JCoMetaData.TYPE_CHAR, 6, 246, 12, 492, 0, null, null, null);

bapiReturn.add("MESSAGE_V1", JCoMetaData.TYPE_CHAR, 50, 252, 100, 504, 0, null, null, null);

bapiReturn.add("MESSAGE_V2", JCoMetaData.TYPE_CHAR, 50, 302, 100, 604, 0, null, null, null);

bapiReturn.add("MESSAGE_V3", JCoMetaData.TYPE_CHAR, 50, 352, 100, 704, 0, null, null, null);

bapiReturn.add("MESSAGE_V4", JCoMetaData.TYPE_CHAR, 50, 402, 100, 804, 0, null, null, null);

bapiReturn.setRecordLength(452, 904);
bapiReturn.lock();

companyList.add("COMPANY", IMetaData.TYPE_CHAR, 6, 0, 12, 0, 0, null, null, null);

companyList.add("NAME1", IMetaData.TYPE_CHAR, 30, 6, 60, 12, 0, null, null, null);

companyList.setRecordLength(36, 72);
companyList.lock();

JCoListMetaData exportsMeta=JCo.createListMetaData("EXPORTS", 1);
exportsMeta.add("RETURN", JCoMetaData.TYPE_STRUCTURE, 452, 904, 0, null, 0, bapiReturn, null);

JCoListMetaData tablesMeta=JCo.createListMetaData("TABLES", 1);
tablesMeta.add("COMPANY_LIST", JCoMetaData.TYPE_TABLE, 36, 72, 0, null, 0, companyList, null);
JCoFunctionTemplate
bapiCompanyGetListTemplate=JCo.createFunctionTemplate("BAPI.Company.GetList",
null, exportsMeta, null, tablesMeta, null);

try
{
    JCoDestination foo=JCoDestinationManager.getDestination("FOO");
    JCo.Function
    bapiCompanyGetList=bapiCompanyGetListTemplate.getFunction();
    bapiCompanyGetList.execute(foo);

    JCoParameterList tables= bapiCompanyGetList.getTableParameterList();
    JCoTable companies=tables.getTable("COMPANY_LIST");
    for (int i=0; i<companies.getNumRows(); i++)
    {
        companies.setRow(i)
        System.out.println(companies.getString("COMPANY")+
                           ":  "+companies.getString("NAME1");
    }
}
catch (Exception e)
{
    System.err.println("Exception occured: "+e.toString());
}

Changes to the Server Role
The server concept has been changed in various ways for release 3.0:

- Previously, each implementation represented an enhancement of the JCO server class. As of release 3.0 different interfaces (classes previously) will be implemented independently.

- As a result, a constructor that transfers the necessary parameters (such as Gateway Host, Gateway Service, Program ID and Repository) to the JCO.Server super class must also not be used.

- Instead of overwriting the protected void method handleRequest(JCo.Function function), a specific interface JcoServerFunctionHandler is implemented.

- Multiple instances will no longer be created for a JCoServer implementation, but just one instance.

- The mapping of logical names (e. g. gateway services) to ports has to be done manually with JCo 3.0.
Enter the corresponding values in the file `services.txt`. The file is located in the following directory:

- UNIX: `/etc/services`
- Windows: `/Windows/system32/drivers/etc/services`

Further Information
You can find detailed information and example programs for the server role in the SAP JCo 3.0 documentation.

IDoc Support
The IDoc class library 3.0 enables IDoc-based communication via SAP JCo.
Contrary to former versions, the IDoc class library 3.0 is based on a destination model for the communication with partner systems.

Further Information
You can find further information and programming examples for the IDoc class library 3.0 in the SAP JCo documentation.

New Method Names in SAP JCo 3.0
In this section you can find an overview of methods that have been renamed with SAP JCo 3.0. The table compares the old and the new names.

Renaming of Methods with SAP JCo 3.0

<table>
<thead>
<tr>
<th>Old Name</th>
<th>New Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMetaData.getTabName()</td>
<td>JCoMetaData.getRecordTypeName()</td>
</tr>
<tr>
<td>IMetaData.getMetaData()</td>
<td>JCoMetaData.getRecordMetaData()</td>
</tr>
<tr>
<td>IMetaData.getTabName()</td>
<td>JCoRecordMetaData.getRecordTypeName()</td>
</tr>
<tr>
<td>IRepository.getTableDefinition()</td>
<td>JCoRecordMetaData.getStructureDefinition()</td>
</tr>
<tr>
<td>IMetaData.getOffset()</td>
<td>JCoMetaData.getByteOffset() / getUnicodeByteOffset()</td>
</tr>
<tr>
<td>IMetaData.getLength()</td>
<td>JCoMetaData.getByteLength() / getUnicodeByteLength()</td>
</tr>
<tr>
<td>IMetaData.setTabLength()</td>
<td>JCoRecordMetaData.setRecordLength()</td>
</tr>
</tbody>
</table>