SAP Java Connector

SAP AG
SAP Java Connector – Overview

- Enables communication between any SAP System and Java (supports communication with SAP R/2 and SAP R/3 3.1 and later)
- Supports inbound (Java calls ABAP) as well as outbound (ABAP calls Java) communication
- Support for Unicode and Non-Unicode Systems
- Hides all difficult parts like codepages, data type conversions, connection pooling, etc. from the programmer
- Hides RFC details from programmers
- Hides DDIC details from programmers
- Consistent and easy-to-learn class design and API
- Highly optimized source code leads to unprecedented performance
Presentation – Content Overview

1. Overview
2. Architecture
3. Scenarios
4. Design
5. Examples
Java API comprises:
- Dynamic metadata lookup and caching
- JCO.Function object offers ready-to-use containers for parameters and/or tables
- Service APIs useful to build higher level applications on top
Java Applications only use SAP JCo’s Java API

Pluggable SAP JCo Middleware Implementations

RFC Middleware uses RFC Library through Java Native Interface (JNI) Layer
Presentation – Content Overview

1. Overview
2. Architecture
3. Scenarios
4. Design
5. Examples
SAP Java Connector – Scenarios

Standalone Java Application as external RFC Client
- Numerous projects (marketplaces)

Java Server as external RFC Server
- SAP Business Connector
- Various projects (marketplaces)

Middleware based on SAP JCo
- JRA for J2EE Application Server
- RFCAdapter
- ObjectFactory
- ObjectBuilder

Interface between SAP J2EE and SAP Web AS
SAP Java Connector – Design

Small and fast

- Minimizes number of internally used objects
- Does not use Java codepage converters
- Avoids using multidimensional arrays
- Minimizes calls between Java and JNI layer
- Manipulates Java member variables directly instead of using function calls in JNI layer
Consistent and easy-to-learn API

Facts about function parameters

- SAP Web AS Functions have Import-, Export-, and Table Parameters
- Parameters can be atomic (Integer, Float, String, etc.)
- Parameter can also be composite (heterogeneous Structures, Tables)

Common representation of parameters

- A parameter list can be considered to be a heterogeneous structure
- A heterogeneous structure in turn can be seen as a table with just a single row
A JCO.Function object
- Represents a SAP Function Module in Java
- Contains meta-data about the call interface and all data types, i.e. (Import-., Export-, Table Parameters, Name, etc.)
- Contains ready-to-use data containers for parameters and / or tables

Function objects are build from templates
- Templates contain all meta-data that completely specify a function
- But they do not have any preinitialized containers yet

Function templates and therefore JCO.Functions can be obtained dynamically from a repository for a given SAP system
- Client side caching ensures efficiency
- There is one repository for each dedicated SAP System
Consistent and easy-to-learn API

Simple Repository API

```java
IRepository repository = JCO.createRepository(...);

JCO.Function function =
    repository.getFunctionTemplate(“WebAS_FUNCTION”).
    getFunction();
```

Consistent API for access to atomic parameters, structures, tables

```java
JCO.ParameterList import =
    function.getImportParameterList();

String s = import.getString(“PARAM1”);
int i = import.getInt(“PARAM1”);
JCO.Structure struct = import.getStructure(“PARAM3”);
JCO.Table table = import.getTable(“PARAM4”);
```
Performance

How to avoid extensive object creation?

- Why not using a heterogeneous buffer (e.g. one dimensional array) divided into cells, each cell holding the field‘s value?

- Since strings are by far the most occurring field types, a character array would be the natural choice for a buffer.

What are the advantages/disadvantages?

- Minimal number of objects
- Fast access to string fields
- Fast access from middleware (e.g. JNI Layer)
- Non-string fields would need some nasty binary encoding / decoding mechanism
Performance

```
byte[] type
int[] length
int[] offset

buffer[offset[column] + row * row_length]
```
Performance

- Metadata for functions and parameters will be fetched at first request and stored in the repository cache.

- Connection pooling
  Avoids expensive `connect()`, `disconnect()` calls.

- Performance measuring: number of calls, sent and received bytes, time measuring for middleware calls, marshalling and unmarshalling, total request time.
SAP Java Connector – Design

Attributes

Throughput

Connection

PoolManager

Pool

Client

Server

BasicRepository

IRepository

Repository

IMetaData

FunctionInterface

IFunctionInterface

Function

Record

MetaData

ParameterList

Structure

Table

Field

FieldIterator
SAP Java IDoc Class Library – Overview

Goal: Offer business level access to binary data

The technical level of an IDoc

Defined by the three record types compatible with the IDoc Interface
- Control record (containing the routing and processing information)
- Data record (containing the business data)
- Status record (containing status information)

The business level of an IDoc

Defined by the segments of an IDoc Interface. Segments are hierarchically ordered structures used to interpret the data stream of the data record. An IDoc Type is defined by the relevant
- Segments
- Attributes of these segments (e.g. minimum and maximum usage, hierarchical sequence)
Schematic example: IDoc: MATMAS01 (Material Master)

Segment type | Data stream
--- | ---
E1MARAM | 005P-100 19941107Baller 19990324D020033 KCVDPALSQBG FERTM001
| | 005DPumpe GG IDESNORM 100-200 DE
| | 005EPump GG IDESNORM 100-200 EN
| | 0051000VDPALSQG PD1011 0 M0.00 EXE 0.000 0.000 0.000
| | 0050001DL 0.000 0.000 0.000
| | 0050002DL 0.000 0.000 0.000
| | 0051300DBGVA PD1011 0 M0.00 EXE800.000 0.000 0.000
| | 0050001D 0.000 0.000 0.000
| | 005PF 5 1 20500000000009 IC0.000 0.000 0.000 0.000
| | 005PF 00001205000000000009 ICX
| | 0051000 S1292.7 1292.7 1 7920S1292.7 1292.7 1
| | 0050001 5.000 0.000 0.000 PCE El
| | 0051000101 X 000000000.000 0.000 0.000
| | 005CA CTX11CTX21CTX31
| | 005MVKE E1000 P-100 100010
| | 005* Vor Einbau, elektrischen Anschluß oder Inbetriebnahme bitte
| | 005 Betriebsanleitungen unbedingt beachten.
SAP Java IDoc Class Library – Features

Provides structured, high-level access to IDocs in Java
- Easy navigation through IDoc segment trees (generic)
- Easy access to single data fields (generic)
- Access to all available IDoc meta data

Supports effective and robust programming
- Typesafe creation and modification of IDocs, segments and fields
- Optional syntax checking (3 different checks)
- Creation of IDoc-HTML files for easy debugging

Supports effective and robust programming
- Supports IDoc base types and customer extension types
- Interface based architecture allows own extensions and implementations
import com.sap.mw.jco.*;

public class Quicky {
    static final String SID = "B20";  // system ID used throughout the example
    IRepository repository;           // the repository we gonna be using

    public Quicky() {
        try {
            // Add a connection pool for the specified system
            JCO.addClientPool(SID,       // alias for this pool
                10,       // max. number of connections
                "000",     // SAP client
                "bbking",  // userid
                "lucille", // password
                "EN",     // language
                "pwdf024", // host name
                "06"      // system number
            );
            // Create a new repository
            repository = JCO.createRepository("MYRepository", SID);
        } catch (JCO.Exception ex) {
            System.out.println("Caught an exception: \n" + ex);
        } //try
    }
}
// Retrieves and prints list of sales orders
public void salesOrders() {

    JCO.Client client = null;

    try {
        // Get a function template from the repository
        JCO.Function function = repository.getFunctionTemplate(
            "BAPI_SALESORDER_GETLIST").getFunction();

        // Fill in input parameters
        JCO.ParameterList input = function.getImportParameterList();
        input.setValue("0000001200", "CUSTOMER_NUMBER");
        input.setValue("1000", "SALES_ORGANIZATION");
        input.setValue("0", "TRANSACTION_GROUP");

        // Get a client from the pool
        client = JCO.getClient(SID);

        // Execute function the remote system
        client.execute(function);

        // Print return message
        JCO.Structure ret=function.getExportParameterList().getStructure("RETURN");
        System.out.println("RETURN MESSAGE: " + ret.getString("MESSAGE");

        // Get a reference to the table which contains the orders
        JCO.Table orders=function.getTableParameterList().getTable("SALES_ORDERS");
// Print results
if (orders.getNumRows() > 0) {
    // Loop over all rows
    do {
        // Loop over all columns in the current row
        for (JCO.FieldIterator e = orders.fields(); e.hasMoreElements();) {
            JCO.Field field = e.nextField();
            System.out.println(field.getName() + ":\t" + field.getString());
        }
    } while(orders.nextRow());
} else {
    System.out.println("No results found");
}//if

} catch (Exception ex) {
    System.out.println("Caught an exception: " + ex);
} finally {
    // Always release client !!!
    JCO.releaseClient(client);
}//try

// Main program
public static void main(String[] argv) {
    Quicky quicky = new Quicky();
    quicky.salesOrders();
}

© SAP AG 2002, SAP JCo / 24
import com.sap.mw.jco.*;

public class QuickyServer {

    static public class Listener extends JCO.Server {
        // Constructor takes gateway host, service, program ID, and repository
        public Listener(String gwhost, String gwserv, String program_id, IRepository repos) {
            super(gwhost, gwserv, program_id, repos);
        }
        // Handles incoming requests
        protected void handleRequest(JCO.Function function) {
            if (function.getName().equals("BAPI_COMPANYCODE_GETLIST")) {
                JCO.Table codes = function.getTableParameterList().getTable("COMPANYCODE_LIST");
                codes.deleteAllRows();
                codes.appendRow();
                codes.setValue("XXXX", "COMP_CODE");
                codes.setValue("This is field comp_name", "COMP_NAME");
            } else {
                throw new JCO.AbapException("FUNCTION_NOT_SUPPORTED",
                        "The requested service is not supported");
            }
        }
    }
}

//Listener
String POOL = "T20"; // System ID
IRepository repository; // Our repository
static final int MAX_LISTENERS = 3; // Max. listener

Listener listeners[] = new Listener[MAX_LISTENERS]; // Listeners

public QuickyServer() {
    JCO.addClientPool(POOL, MAX_LISTENERS, "400", "tarzan", "jane", "EN",
                       "iwdf5020.wdf.sap-ag.de", "T20", "SPACE");
    repository = JCO.createRepository("QUICKY", POOL);
}

public void startThemAll() {
    for (int i = 0; i < MAX_LISTENERS; i++) {
        listeners[i] = new Listener("iwdf5020", "sapgw00", "QUICKY", repository);
        try {
            listener[i].start();
        } catch (Exception ex) {
            System.out.println("Could not start listener !\n" + ex);
        }//try
    }//for
}

public static void main(String[] argv) {
    QuickyServer qserver = new QuickyServer();
    qserver.startThemAll();
}