FUJITSU

WHITE PAPER

BeanConnect™ V3.0 Technical Overview

Issue September 2013

Pages 13

Summary

With the product BeanConnect™ 3.0, Fujitsu offers software which complies with the JCA 1.6 specification.

The Java EE Connector Architecture (JCA) 1.6 is part of the Java Platform, Enterprise Edition 6 (Java EE 6) and defines a standard architecture for accessing resources of an Enterprise Information System (EIS). The Connector Architecture facilitates the integration of Java applications with heterogeneous EIS systems.

The product BeanConnect™ 3.0 connects applications based on a Java EE application server to applications of the transaction systems openUTM (Fujitsu) and CICS (IBM).

This paper describes the BeanConnect™ 3.0 functionality and its application.

Contents

Introduction

Over the years most companies have invested large amounts in Enterprise Information Systems (EISs) such as ERP systems, legacy applications, mainframe-based databases and transaction systems. In order to protect these investments, the EIS systems must be integrated into today's environment of web-based application, and this presents a great challenge.

The manufacturers of the EIS systems offer proprietary interfaces to their systems which, however, are not necessarily suitable for supporting Enterprise Application Integration (EAI). The application server vendors therefore would have to provide separate interfaces for each EIS system. At the same time the application developers would have to implement and manage features in the application at system level, for example security, transactions and connection spooling.

This is where the Java EE Connector Architecture comes in. This specification enables interfaces to be defined which make it easier to integrate an EIS system into an application server.

Overview of JCA 1.6

Challenges for EIS integration

The integration of EIS systems presents a number of challenges:

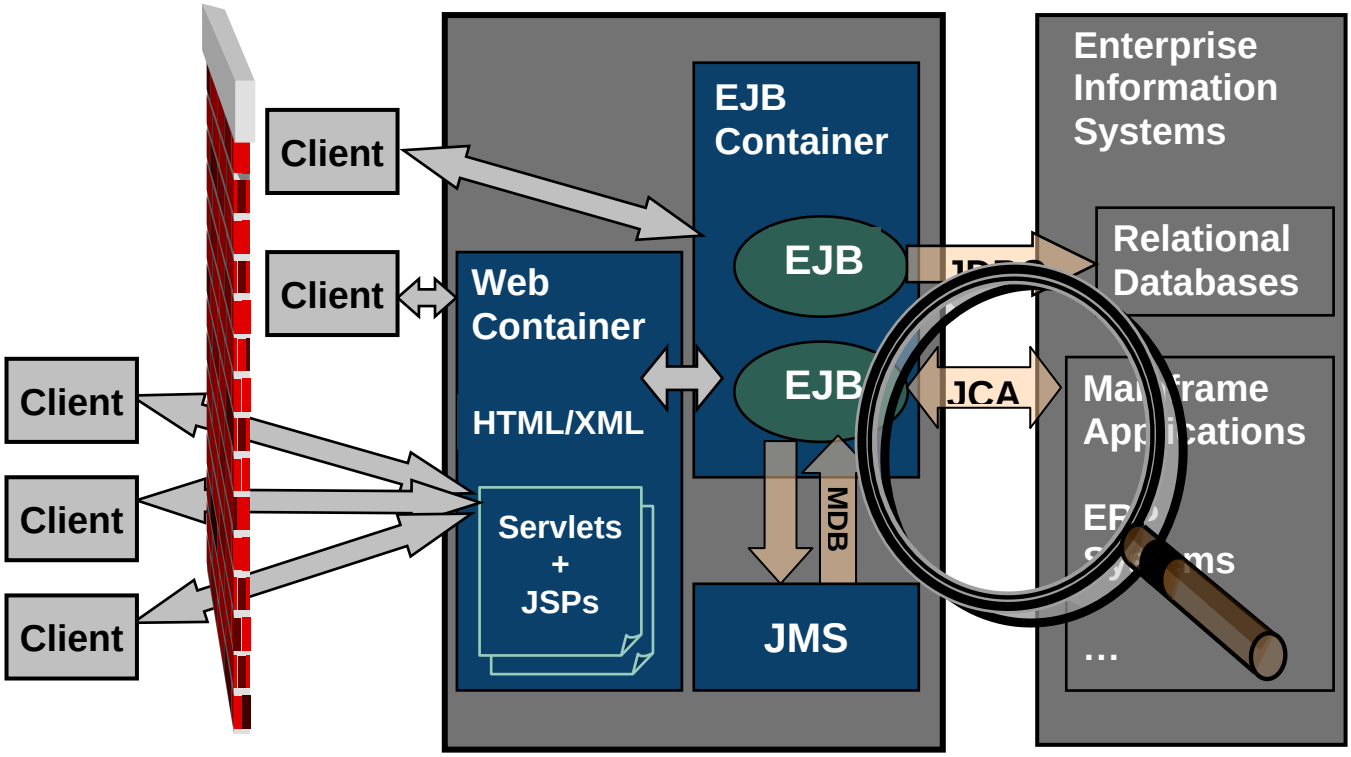
- The backend EIS systems are extremely complex and heterogeneous. The application programming is therefore very much dependent on the particular EIS system used. This consequently results in greater complexity and effort when applications are integrated. Tools to facilitate the integration of EIS systems are therefore particularly important.
- Transaction and security administration also make the integration of backend EIS systems more complex.
- The web-based frontend architecture requires a high level of scalability to permit a large number of clients to access the EIS systems in parallel.

The Java EE platform and the Connector Architecture

These challenges are addressed by the Connector Architecture. With its model of reusable components in which Enterprise JavaBeans (EJB) and Java Server Pages (JSP) are employed, the Java EE platform offers an ideal basis for generating and configuring a multitier application which is both platform- and vendor-independent. The Java EE platform is based on the claim familiar to us from Java: "Write Once, Run Anywhere" and is very widespread in the industry.

The Connector Architecture adds a simplified EIS integration to the Java EE platform. The aim here is to utilize the strengths of the Java EE platform (component model, transactions, security, etc.) in order to handle EIS integration.

The Connector Architecture defines a general interface between the application servers and the EIS systems. In concrete terms this means: When an EIS vendor implements a Resource Adapter which complies with JCA for their system, the EIS system can be integrated into any application server which complies with Java EE. This approach results in simplified application integration, which permits the benefits of scalable standard JAVA EE platform architectures to be utilized.



JCA 1.6 contracts

It must be possible to implement the interfaces of the Connector Architecture in the application server and in the EIS-specific Resource Adapter. A Resource Adapter is an EIS-specific system library which provides the connection to the EIS system. Resource Adapters are similar to JDBC drivers: The interface between a Resource Adapter and an EIS system is EIS-specific. The interface between a Resource Adapter and an application server, on the other hand, is standardized.

The Connector Architecture defines seven types of so-called system contracts and one Common Client Interface (CCI):

- Connection Management Contract
  This contract enables the application server to set up connections to an EIS system. This also includes pooling the connections made available by the Resource Adapter on the application server.
- Transaction Management Contract
  This enables transactions on the application server to be propagated to the EIS system.
- Security Management Contract
  The access data for the EIS system can be configured on the application server.
- Lifecycle Management Contract
  Management of the lifecycle of a Resource Adapter (startup/shutdown).
- Work Management Contract
  A Resource Adapter typically consists of multi-threaded software. The threads required here are made available by the application server via this contract and managed in a thread pool.
- Message Inflow Contract
  This contract allows the application server to be addressed from the EIS system, too.
- Transaction Inflow Contract
  An EIS transaction can be imported into the application server via this contract.
- Common Client Interface (CCI)
  A general client interface.
  The CCI is typically used by business integration tools.
- Generic Work Context Contract (new in JCA 1.6)
  This contract enables the Resource Adapter to forward context information for Inbound Communication received from the EIS to the application server.
- Security Work Context (new in JCA 1.6)
  The Security Work Context enables the Resource Adapter, by means of the Generic Work Context Contract, to forward security information for the inbound communication, received from the EIS, to the application server.

Overview of BeanConnect

Motivation

With openUTM, Fujitsu offers an excellent product which provides very good performance features in terms of transactions, scalability and stability. Furthermore, openUTM supports OSI-TP, which makes it ideal for use in heterogeneous, transaction-oriented environments.
On the basis of this technological experience, BeanConnect offers a Resource Adapter implementation which complies with JCA and permits access to openUTM- and CICS-based applications in the JAVA EE application server environment.

Components

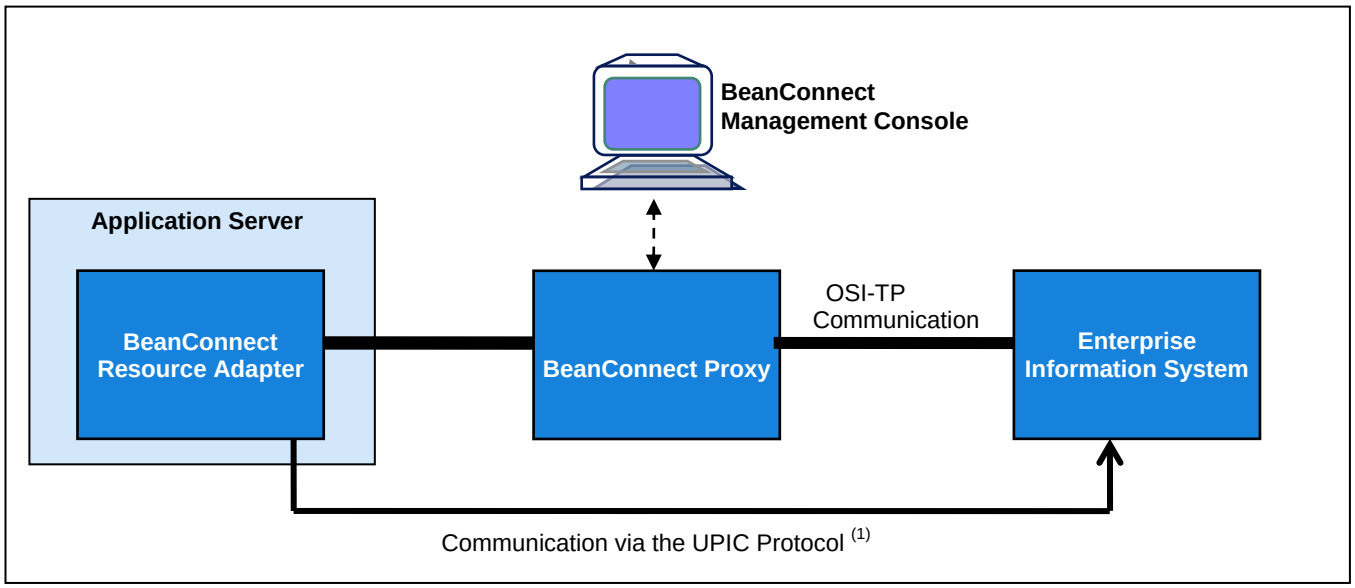
The BeanConnect software comprises the following components:

- BeanConnect Resource Adapter
  The BeanConnect Resource Adapter implements the JCA 1.6 interfaces.
  As an adapter which complies with JCA it is deployed on the application server and therefore executes on the application server.

  For application servers which solely support JCA 1.5, a separate BeanConnect Resource Adapter is delivered which implements the JCA 1.5 interfaces only.
- BeanConnect Proxy
  The BeanConnect Proxy provides the functionality of a protocol machine and functions for transaction control. It can be regarded as an intelligent gateway. It communicates with the Resource Adapter which runs on the application server on the one side and with the EIS partner on the other.
- BeanConnect Management Console
  The BeanConnect Management Console (MC) is a Java-based graphical user interface for configuring and administering BeanConnect Proxies which run on the same system or on remote systems.

  In addition to graphical administration, administration via scripts is possible, wherewith administration activities may be automated. Jython is used as the script language.

  During configuration the Management Console is used to store all the data which is required for communication between the BeanConnect Resource Adapter and the EIS and which is accessed at runtime.
  The BeanConnect Proxy can be started up/shut down, monitored and employed for diagnostic support using the Management Console.



(1): Communication using the UPIC protocol, i.e. without the proxy, is possible in the case of EIS openUTM only.

Details on configuration:

- No software for using BeanConnect need be connected on the EIS system (i.e. BeanConnect permits **non-invasive integration**).
- The application server with the BeanConnect Resource Adapter, the BeanConnect Proxy and the Management Console can run together either on a single system or on separate systems.

Data exchange

BeanConnect, as a JCA 1.6 Resource Adapter, supports both outbound communication (i.e. from the application server to the EIS system) and inbound communication (i.e. from the EIS system to the application server). These two communication types can be used both synchronously, i.e. with waiting for a response, and asynchronously, i.e. without a response option.
Two interfaces each are offered for inbound and for outbound communication:

- The BeanConnect-specific API (**B**ean**C**onnect **I**nterface, BCI for short)
- The **C**ommon **C**lient **I**nterface specified by JCA, CCI for short.

Transactions

When the OSI-TP protocol is used for data exchange, transactions can be propagated from the application server to the EIS system (= outbound communication) and imported from the EIS system to the application server (= inbound communication), in both cases transparently for the EJB programmer.

Security

Different access controls are supported depending on the communication scenario.

Outbound communication:

For the authorization check, user IDs and passwords can be forwarded to the EIS system for checking purposes.
Two different options are offered:

- **Container-based security**:
  The user and password for the check can be defined by the EIS in the configuration.
- **Application-based security:**
  At the EJB programmer's API the class net.fsc.jca.communication.PasswordCredential is used to define the user and password, and this is transferred as a parameter when the getConnection() method is called.

Inbound communication:

If the EIS partner forwards user ids and passwords for the inbound communication, these are propagated through the Security Work Context (JCA 1.6) to the application server, provided that the BeanConnect Proxy's data check was successful.

Encoding (code conversion)

Depending on the application scenario, different code conversion methods are used to convert the data which is exchanged which has to be presented in different ways in the Java environment and in the EIS.

- For **printable characters** (strings) it is possible to enable automatic encoding on the basis of **code tables that are supplied** or to use **user-defined code conversion tables** for special cases which are defined by the EJB programmer.
- For **data structures** specified in **Cobol** which contain both printable characters and binary data, BeanConnect offers **powerful tools** which can **generate** both **automatic conversion routines** and **Java methods** to permit convenient access to the converted data fields from the Java program.

- A set of appropriate **conversion methods can be implemented by the user** for other **special cases**, e.g. for data structures defined in Assembler or C, using the net.fsc.jca.communication.**ByteContainer** interface.

  In one implementation of this interface programmers themselves can decide flexibly how the data is to be converted. BeanConnect calls the getBytes() method of the ByteContainer before the data is transmitted to the EIS system, and the setBytes() method after it has been received.

Parallel Request/Response Model

A parallel Request/Response Model can be used in the BeanConnect Resource Adapter.

This process is considerably more powerful than the sequential inquiry of the EIS systems.

In BeanConnect the term "connection grouping" is used for parallel processing. Here multiple connections (EISConnections) are included in a group. After all the parameters required for transmission have been set on the various connections, the execute() method is called on the EISConnectionGroup. All requests are then sent to the EIS systems involved. The BeanConnect Proxy sends these requests simultaneously to all partner systems and waits for the responses. Once the last response is available the execute() call is returned.

The use of connection groups is presented in appendix 2 (Sample codes).

Innovations of BeanConnect V3.0 compared to V2.1

The innovations of BeanConnect V3.0 compared to V2.1:

Resource Adapter

- New standard application server
  The standard application server for BeanConnect as of V3.0 is Oracle™ WebLogic Server (up to V2.1: Oracle Internet Application Server).
- JCA 1.6
  BeanConnect V3.0 can work together with application servers, which provide support for the specification JCA 1.5 or JCA 1.6. Two versions of the resource adapter are supplied with BeanConnect V3.0. As standard, the BeanConnect Resource Adapter can be used for application servers that provide support for JCA 1.6 (e.g. Oracle™ WebLogic Server).
  A separate BeanConnect Resource Adapter is supplied for application servers that only provide support for JCA1.5.
- Load balancer
  Use of the JConnect load balancing functionality for outbound communication with an UTM-Cluster application using the UPIC protocol.
  For outbound communication via OSI TP the UTM load balancing feature (MASTER-OSI-LPAP) of the BeanConnect Proxy is used.

Management Console

- Command Line Interface (CLI)
  In addition to graphical administration, administration via scripts is possible with BeanConnect V3.0. Jython is used as the script language.
- UTM Cluster as an EIS Partner
  It is possible to configure a UTM-Cluster application as an EIS partner,
- Generating EIS Partners
  Proxies can be configured with various APTs, thus enabling each proxy to receive an own application process title (APT) when coupling a UTM application with several proxies.
- EIS Partners in the Properties Dialog of an Outbound Communication Endpoint
  It is now possible to make direct changes to the EIS partner in the properties dialog of the outbound communication endpoint.

MCCmdHandler

- Windows, Activation of the MCCmdHandler Server Port
  Since problems can occur when running the BeanConnect MCCmdHandler if the Windows firewall is enabled, it is now possible to activate the MCCmdHandler server port during installation.
- Windows, Service Monitoring
  A message box is output when monitoring the service for impermissible messages.

Conclusion

The Connector Architecture expands the tried-and-tested Java EE platform into the integration platform for EIS systems. It permits scalable use of the enterprise resources in a state-of-the-art environment without endangering the data integrity and security of the EIS systems.

BeanConnect offers suitable support to allow transaction-oriented systems such as openUTM and CICS to be integrated into a Java EE-based platform with the aid of the Connector Architecture.

Detailed information is provided

- in the appendixes “Terms” and “Sample codes” in this document.
- on the BeanConnect website (see below).
- from other sources on the Internet (see below for a selection).

Further sources on the Internet

- BeanConnect
  http://www.ts.fujitsu.com/beanconnect
- openUTM
  http://www.ts.fujitsu.com/openUTM
- openSEAS
  http://www.ts.fujitsu.com/openSEAS
- Oracle WebLogic
  http://www.oracle.com/technetwork/middleware/weblogic/overview/index.html
- Java EE Connector Architecture:
  http://www.jcp.org/en/jsr/detail?id=322
- Java EE Connector Architecture Specification:
  http://jcp.org/aboutJava/communityprocess/final/jsr322/index.html
- White Paper:
  http://www.oracle.com/technetwork/java/javaee/overview/connector-jsp-135375.html
- The Java EE 6 Tutorial
  http://docs.oracle.com/javaee/6/tutorial/doc/

Appendix 1: Technical terms

This appendix contains the principal technical terms which cannot be assumed to be generally known but which are helpful in understanding the examples below.

The order has been established to ensure that terms needed to explain other terms are explained first.

EIS (Enterprise Information System)

An Enterprise Information System (EIS), also referred to as EIS partner application, is the partner system which is accessed via BeanConnect.

BeanConnect V3.0 supports two versions of EIS:

- **BeanConnect for openUTM** permits communication with applications based on openUTM from Fujitsu.
- **BeanConnect for CICS** permits communication with applications based on CICS from IBM.

For inbound communication two additional versions of EIS are supported:

- UPIC application
- openUTM Socket or RFC1006 application.

BeanConnect Interface (BCI)

The BeanConnect Interface, BCI for short, is based on the JCA connection contract and, via interfaces that are easy to handle, permits communication with the EISs based on openUTM or CICS which are supported by BeanConnect.

The examples in this document are all based on BCI since this API is BeanConnect’s preferred method for accessing an EIS.

ProxyURL

The ProxyURL defines the assignment of the deployed Resource Adapter to the BeanConnect Proxy.

The ProxyURL is defined when the Resource Adapter is deployed.

The definition has the format oltp://<host>:<port>/<name>

The components here have the following meaning:

- `<host>` System on which the Proxy Container is installed
- `<port>` Port number of the Proxy Container +4
- `<name>` Application name of the Proxy Container (BCU<port>)

The default value for the ProxyURL is:

```
oltp://localhost:31004/BCU31004
```

i.e. the proxy is located on the same system as the Resource Adapter and can be reached via port 31004.

Outbound Service

An Outbound Service is a service (transaction code) which is provided by the EIS partner.

Outbound Services are defined using the BeanConnect Management Console.

Communication Endpoint

A Communication Endpoint is required for outbound communication.
In the definition of the Outbound Communication Endpoint the symbolic service name is mapped to an EIS partner application's real service name.

Outbound Communication Endpoints are defined with the aid of the Management Console.

An Outbound **Communication Endpoint** has the following features:

`Name:` Specifies the Communication Endpoint's symbolic name.

`EIS Partner:` EIS partner to which the Communication Endpoint belongs.

Note: The EIS partner must be configured beforehand.

`Partner Service:` Actual name of the service in the EIS system (i.e. transaction code of openUTM or CICS).

Note:
The service must be defined as an Outbound Service beforehand.

Connection Factory

You can request a connection handle for a connection to the EIS using a Connection Factory's getConnection() method.

BeanConnect offers the following Connection Factories:

- net.fsc.jca.communication.EISOltpConnectionFactory
  Communication via OSI-TP using openUTM or via LU6.2 using CICS.
- net.fsc.jca.communication.EISUpicConnectionFactory
  Communication using the openUTM UPIC protocol with an openUTM application.
- Both Connection Factories are extensions of the basic class
  net.fsc.jca.communication.EISConnectionFactory

Provided that the basic class EISConnectionFactory is used only, because no ConnectionFactory-specific functions are required, the protocol which is used may be modified by adjustment of the configuration files for the Resource Adapter (Standard- and WebLogic Server-specific Deployment Descriptor ra.xml and weblogic-ra.xml), leaving the code of the Enterprise Java Bean (EJB) unchanged.

ConnectionURL

For outbound communication via OSI-TP the configuration property ConnectionURL specifies the name of the Outbound Communication Endpoint which represents the connection to an EIS partner.

The ConnectionURL is defined when a Connection Factory is deployed.

The definition has the format `<type>://<name>`
The components here have the following meaning:

- `<type>` Type of the EIS Partner, possible values:

```
utm          the EIS partner is of type openUTM.
cics         the EIS partner is of type CICS.
xatmi.rr     the EIS partner is of type XATMI and the communikation follows
             the request/reply paradigm.
xatmi.cv     the EIS partner is of type XATMI and the communikation follows
             the conversational paradigm.
```

- `<name>` Name of an Outbound Communication Endpoint as defined on the Management Console

The default value is: utm://outboundCommunicationEndpoint

OLTP Message-Driven Bean

OLTP Message-Driven Beans are JCA-compliant Message Endpoint applications which are supported by BeanConnect.
An EIS application can be called via inbound communication EJBs which are deployed on an application server and implement the OLTP Message-Driven Bean Interface.

Inbound Message Endpoint

An Inbound Message Endpoint is the endpoint of inbound communication on the JAVA EE application server.
An Inbound Message Endpoint of the same name must be configured on the BeanConnect Proxy for each Inbound Message Endpoint on the application server using the BeanConnect Management Console. To each Inbound Message Endpoint at least one Inbound Service has to be assigned.

Inbound Service

An Inbound Service is a service addressed by an EIS partner for inbound communication.

Appendix 2: Sample codes

This section describes the use of the BeanConnect interfaces in selected simple scenarios.

BeanConnect-specific interfaces for outbound communication

Example 1: Single-step service

A code fragment from an EJB which uses an EISConnectionFactory to call the one-step service "KDCINF" in an openUTM application on a BS2000 system using the "STAT" parameter is shown below:

```
Context ctx = new InitialContext();

eisConnectionFactory = (EISConnectionFactory)
                           ctx.lookup("java:comp/env/eis/myOltp");

eisConnection = eisConnectionFactory.getConnection();

eisConnection.setServiceName("KDCINF");
String response = eisConnection.call("STAT");
```

The associated entry in the `ejb-jar.xml` deployment file of the Enterprise JavaBean is as follows:

ejb-jar.xml

```
<session>
...
  <resource-ref>
    <res-ref-name>eis/myOltp</res-ref-name>
    <res-type>net.fsc.jca.communication.EISConnectionFactory</res-type>
    <res-auth>Container</res-auth>
    <res-sharing-scope>Unshareable</res-sharing-scope>
  </resource-ref>
</session>
```

If communication with openUTM on BS2000 takes place on a **transaction-oriented basis** via the **BeanConnect Proxy**, the WebLogic Server-specific deployment information for the BeanConnect Resource Adapter is as follows:

weblogic-ra.xml

```
...
    <connection-definition-group>
      <connection-factory-
interface>net.fsc.jca.communication.EISOltpConnectionFactory</connection-factory-
interface>
      <default-connection-properties>
        <pool-params>
          <initial-capacity>2</initial-capacity>
          <max-capacity>10</max-capacity>
        </pool-params>
        <transaction-support>XATransaction</transaction-support>
      </default-connection-properties>
      <connection-instance>
        <jndi-name>eis/myOltp</jndi-name>
        <connection-properties>
          <pool-params>
            <initial-capacity>4</initial-capacity>
          </pool-params>
          <properties>
            <property>
              <name>ConnectionURL</name>
              <value>utm://UTMAPP_ON_BS2_HOST1</value>
            </property>
            <property>
              <name>encoding</name>
              <value>OSD_EBCDIC_DF04_DRV</value>
            </property>
            <property>
              <name>encodingActive</name>
              <value>true</value>
            </property>
            <property>
              <name>transactional</name>
              <value>true</value>
            </property>
            <property>
              <name>timeout</name>
              <value>30000</value>
            </property>
            <property>
              <name>bufferedIO</name>
              <value>true</value>
            </property>
            <property>
              <name>logLevel</name>
              <value>NONE</value>
            </property>
            <property>
              <name>displayName</name>
              <value>eis/myOltp</value>
            </property>
          </properties>
        </connection-properties>
      </connection-instance>
  </connection-definition-group>
```

If communication takes place **directly with openUTM on BS2000** on a **non-transaction-oriented basis** using the openUTM UPIC protocol, the WebLogic Server-specific deployment information for the BeanConnect Resource Adapter for the `connectionfactory-interface` and `ConnectionURL` properties is somewhat different:

weblogic-ra.xml

```
<connection-factory-interface>
                    net.fsc.jca.communication.EISUpicConnectionFactory
</connection-factory-interface>
            <property>
              <name>ConnectionURL</name>
              <value>upic://HOST1:102/UTMAPP/KDCHELP</value>
            </property>
```

Example 2: Parallel Request/Response Model

For exemplification a code fragment is provided below, where two services are called in parallel and the execute method is used to wait for the responses:

```
EISConnectionFactory      cf1 = (EISConnectionFactory)
                                ic.lookup("java:comp/env/eis/service1");
EISConnectionGroupFactory cgf = cf1.getEISConnectionGroupFactory();
EISConnectionGroup        cg  = cgf.getConnectionGroup();

EISConnection             c1  = cf1.getConnection(cg);

EISConnectionFactory      cf2 = (EISConnectionFactory)
                                ic.lookup("java:comp/env/eis/service2");
EISConnection             c2  = cf2.getConnection(cg);

                          C1.sndString("Message for service1");
                          C2.sndString("Message for service2");

cg.execute(); // Waits until all reply messages are available

String                r1 = c1.rcvString();
String                r2 = c2.rcvString();
```

Examples for outbound security

The two possible options are illustrated in the following examples of the use of security functions in outbound communication.

1. **Container-based security:**
   Using WebLogic, the Credential Mappings for Container Managed Security for Outbound Communication are defined with the WebLogic Administration Console (see BeanConnect manual, page 120).

2. **Application-based security:**

   Extract from the Java code:

```
PasswordCredential pc =
                   new PasswordCredential(„UTMADMIN", „UTM4EVER");
eisConnection = eisConnectionFactory.getConnection(pc);
```

Example for inbound communication

A socket for communication between the BeanConnect Proxy and the BeanConnect Resource Adapter is always used within BeanConnect for the inbound communication.

This port is configured as **inboundListenerPort** in the Resource Adapter's associated deployment information:

InboundListenerPort definition in the ra.xml descriptor file

```
<resourceadapter>
  <config-property>
   <config-property-name>inboundListenerPort</config-property-name>
   <config-property-type>java.lang.String</config-property-type>
   <config-property-value>31099</config-property-value>
...
</resourceadapter>
```

To permit communication with an OLTP Message-Driven Bean, an EIS application sends a message to an inbound service, which is assigned to an Inbound Message Endpoint configured by means of the Management Console.

The messageEndpoint property of the OLTP Message-Driven Bean defined in the ejb-jar.xml file must match the name of the Inbound Message Endpoint defined on the proxy.

At runtime the message is forwarded from the BeanConnect Proxy to the OLTP Message-Driven Bean whose Message Endpoint name is linked in the configuration to the service name used.

The Message-Driven Bean which is to process the inbound messages must implement one of the following BeanConnect-specific interfaces:

Listener interfaces for the Message-Driven Bean

```
net.fsc.jca.communication.OltpMessageListener       (synchronous)
net.fsc.jca.communication.AsyncOltpMessageListener  (asynchronous)
```

The inbound messages can then be read and processed in the onMessage method of the Message-Driven Bean. An example of a simple synchronous service is provided below:

Message-Driven Bean implementation

```
public class SimpleMessageDrivenBean
    implements MessageDrivenBean,
               OltpMessageListener
{
...

public OltpMessage onMessage(OltpMessage msg)
{
  String inMsgTxt;
  OltpMessageContext oltpMsgCtx = inMsg.getMessageContext();
...
  // read request
  if (inMsg.countMessageParts() > 0) {
    OltpMessagePart inMsgPart;
    Iterator msgParts = inMsg.getMessageParts();
    for ( ; msgParts.hasNext(); ) {
      inMsgPart = (OltpMessagePart) msgParts.next();
      inMsgTxt  = inMsgPart.getText();
      // @TODO: process message part
    }
...// @TODO: process request
  }

  // setup reply
  OltpMessage outMsg = oltpMsgCtx.createMessage();
  OltpMessagePart outMsgPart = outMsg.createMessagePart();
  outMsgPart.setText("Reply from SampleDialogOltpMdbBean");
  outMsg.addMessagePart(outMsgPart);
  return (outMsg);
}
}
```

The following deployment data is associated with the implementation above:

ejb-jar.xml

```
<enterprise-beans>
 <message-driven>
  <ejb-name>SimpleMessageDrivenBean</ejb-name>
   ...
  <messaging-type>
   net.fsc.jca.communication.OltpMessageListener
  </messaging-type>
   ...
  <activation-config>
   <activation-config-property>
     <activation-config-property-name>
     messageEndpoint
     </activation-config-property-name>
     <activation-config-property-value>
     SimpleMessageDrivenBean
     </activation-config-property-value>
   </activation-config-property>
   <activation-config-property>
     <activation-config-property-name>
     encodingActive
     </activation-config-property-name>
     <activation-config-property-value>
     true
     </activation-config-property-value>
   </activation-config-property>
   <activation-config-property>
     <activation-config-property-name>
     encoding
     </activation-config-property-name>
     <activation-config-property-value>
     OSD_EBCDIC_DF04_15
     </activation-config-property-value>
   </activation-config-property>
  </activation-config>
 </message-driven>
```

Consider the following: The `encodingActive` property was set to `true` here and an encoding was specified ("OSD_EBCDIC_DF04_15"). Which means: When strings are exchanged between the EIS system and the application server, the characters are automatically converted.

The Weblogic Server-specific deployment information for the example above is as follows:

weblogic-ejb-jar.xml

```
<weblogic-ejb-jar xmlns="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/weblogic/weblogic-ejb-jar
http://xmlns.oracle.com/weblogic/weblogic-ejb-jar/1.1/weblogic-ejb-jar.xsd">
 <weblogic-enterprise-bean>
  <ejb-name>SimpleMessageDrivenBean</ejb-name>
  <message-driven-descriptor>
   <resource-adapter-jndi-name>BeanConnect</resource-adapter-jndi-name>
  </message-driven-descriptor>
  <jndi-name>SimpleMessageDrivenBean</jndi-name>
 </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```


Published by department:

openSEAS@ts.fujitsu.com
ts.fujitsu.com