FUJITSU

WHITE PAPER

# BeanConnect™ V3.0
# Technischer Überblick

**Ausgabe** September 2013

**Seiten** 13

## Zusammenfassung

Fujitsu bietet mit dem Produkt BeanConnect™ 3.0 eine der JCA 1.6 Spezifikation genügende Software an.
Java EE Connector Architecture (JCA) 1.6 ist Teil der Java Platform, Enterprise Edition 6 (Java EE 6) und beschreibt eine Standardarchitektur für den Zugriff auf Ressourcen eines Enterprise Information Systems (EIS). Die Connector Architektur vereinfacht die Integration von Java Anwendungen mit heterogenen EIS Systemen.
Das Produkt BeanConnect™ 3.0 verbindet Anwendungen auf Basis eines Java EE-Application Servers mit Anwendungen der Transaktionssysteme openUTM (Fujitsu) und CICS (IBM).
Dieses Papier beschreibt die BeanConnect™ 3.0 Funktionalität und ihre Nutzung.

**Inhalt**

**Zusammenfassung** **1**
**Einführung** **2**
**JCA 1.6 Überblick** **2**
Herausforderungen bei der EIS Integration 2
Die Java EE Plattform und die Connector Architektur 2
JCA 1.6 Kontrakte 2
**BeanConnect Überblick** **3**
Motivation 3
Komponenten 3
Datenaustausch 4
Transaktionen 4
Sicherheit 4
Encoding (Codekonvertierung) 4
Paralleles Request/Response Modell 5
**Die Neuheiten von BeanConnect V3.0 gegenüber V2.1** **5**
**Fazit** **5**
**Weiterführende Quellen im Internet** **6**
**Anhang 1: Fachbegriffe** **6**
EIS (Enterprise Information System) 6
BeanConnect Interface (BCI) 6
ProxyURL 6
Outbound Service 7
Communication Endpoint 7
Connection Factory 7
ConnectionURL 7
OLTP Message-Driven Bean 7
Inbound Message Endpoint 8
Inbound Service 8
**Anhang 2: Code Beispiele** **9**
BeanConnect-spezifische Interfaces für Outbound Kommunikation 9
Beispiel 1: Einschritt-Service 9
Beispiel 2: Paralleles Request/Response Modell 11
Beispiele zur Outbound Security 11
Beispiel zur Inbound Kommunikation 11

## Einführung

Die meisten Firmen haben über die Jahre große Summen in Enterprise Information Systeme (EISe), wie z.B. ERP Systeme, Legacy Anwendungen, Mainframe basierte Datenbanken und Transaktionssysteme investiert. Damit diese Investitionen geschützt werden, müssen die EIS Systeme in die heutige Landschaft von webbasierten Anwendungen eingebunden werden, was eine große Herausforderung darstellt.
Die Hersteller der EIS Systeme bieten proprietäre Schnittstellen zu ihren Systemen an, die jedoch nicht unbedingt für die Unterstützung einer Enterprise Application Integration (EAI) geeignet sind. Die Application Server Anbieter müssten deshalb für jedes EIS System eigene Schnittstellen zur Verfügung stellen. Die Anwendungsentwickler müssten dabei Eigenschaften der Systemebene, wie Sicherheit, Transaktionen und Verbindungspooling selbst in der Anwendung implementieren und verwalten.
Hier setzt die Java EE Connector Architecture an. Mit Hilfe dieser Spezifikation werden Schnittstellen festgelegt, die die Einbindung eines EIS Systems in einen Application Server vereinfachen.

## JCA 1.6 Überblick

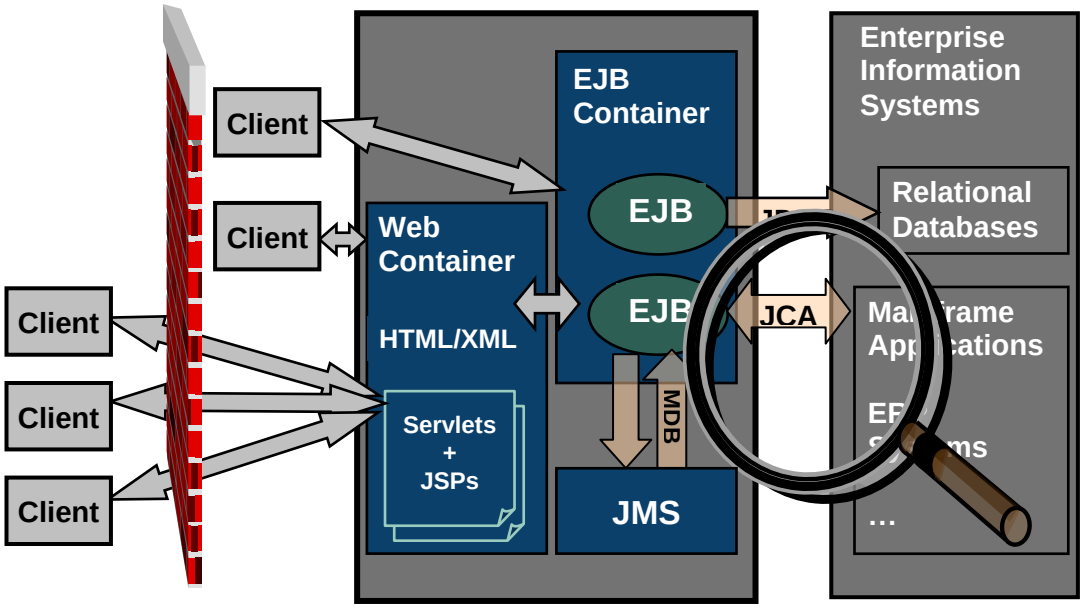
### Herausforderungen bei der EIS Integration

Die Integration von EIS Systemen enthält viele Herausforderungen:

- Die Backend EIS Systeme sind sehr komplex und heterogen. Die Anwendungsprogrammierung hängt deshalb stark vom jeweils genutzten EIS System ab. Dadurch steigen die Komplexität und der Aufwand bei einer Anwendungsintegration. Tools zur Vereinfachung der Integration von EIS Systemen sind deshalb besonders wichtig.
- Transaktions- und Sicherheitsverwaltung erhöhen darüber hinaus die Komplexität der Integration von Backend EIS Systemen.
- Die webbasierte Frontend Architektur erfordert eine hohe Skalierbarkeit, damit eine hohe Anzahl von Clients parallel auf die EIS Systeme zugreifen kann.

### Die Java EE Plattform und die Connector Architektur

Die Connector Architektur geht diese Herausforderungen direkt an. Die Java EE Plattform bietet hierfür mit ihrem Modell der wieder verwendbaren Komponenten, in dem Enterprise JavaBeans (EJB) und Java Server Pages (JSP) genutzt werden, eine ideale Grundlage, um eine multitier Anwendung zu erzeugen und zu konfigurieren, die plattform- und anbieterunabhängig ist. Die Java EE Plattform basiert auf dem von Java her bekannten Anspruch: "Write Once, Run Anywhere" und hat eine große Verbreitung in der Industrie.
Die Connector Architektur fügt eine vereinfachte EIS Integration zur Java EE Plattform hinzu. Das Ziel ist es, die Stärken der Java EE Plattform (Komponentenmodell, Transaktionen, Sicherheit, …) zu nutzen, um die Herausforderung einer EIS Integration zu meistern.
Die Connector Architektur definiert eine allgemeine Schnittstelle zwischen Application Servern und EIS Systemen. Das bedeutet konkret: Implementiert ein EIS Hersteller für sein System einen JCA konformen Resource Adapter, so kann das EIS System in jedem Java EE konformen Application Server eingebunden werden. Das Ergebnis dieses Ansatzes ist eine vereinfachte Anwendungsintegration, bei der die Vorteile der skalierbaren Standardarchitektur der Java EE Plattform genutzt werden können.




### JCA 1.6 Kontrakte

Die Schnittstellen der Connector Architektur müssen im Application Server und im EIS spezifischen Resource Adapter implementiert werden. Ein Resource Adapter ist eine EIS spezifische Systembibliothek, die die Verbindung zum EIS System zur Verfügung stellt. Resource Adapter haben eine Ähnlichkeit mit JDBC-Treibern: Die Schnittstelle zwischen einem Resource Adapter und einem EIS System ist EIS spezifisch. Die Schnittstelle zwischen Resource Adapter und Application Server ist jedoch standardisiert.

Die Connector Architektur definiert sieben Typen von so genannten System Contracts und ein Common Client Interface (CCI):

- Connection Management Contract
  Dieser Kontrakt erlaubt dem Application Server Verbindungen zu einem EIS System aufzubauen. Hierzu gehört auch ein Pooling der durch den Resource Adapter zur Verfügung gestellten Verbindungen im Application Server.
- Transaction Management Contract
  Transaktionen im Application Server können hierüber in das EIS System propagiert werden.
- Security Management Contract
  Im Application Server können die Zugangsdaten für das EIS System konfiguriert werden.
- Lifecycle Management Contract
  Verwaltung der Lebensdauer eines Resource Adapter (Startup/Shutdown).
- Work Management Contract
  Typischerweise ist ein Resource Adapter eine multi-threaded Software. Die benötigten Threads werden dabei vom Application Server über diesen Kontrakt zur Verfügung gestellt und in einem Thread Pool verwaltet.
- Message Inflow Contract
  Dieser Kontrakt erlaubt, dass der Application Server auch vom EIS System angesprochen werden kann.
- Transaction Inflow Contract
  Eine EIS Transaktion kann hierüber in den Application Server importiert werden.
- Common Client Interface (CCI)
  Ein allgemeines Client Interface.
  Das CCI wird typischerweise von Tools für die Business Integration verwendet.
- Generic Work Context Contract (neu in JCA 1.6)
  Dieser Kontrakt erlaubt dem Resource Adapter Kontextinformation für Inbound Kommunikation, die er vom EIS erhalten hat, an den Application Server zu übergeben.
- Security Work Context (neu in JCA 1.6)
  Der Security Work Context erlaubt es dem Resource Adapter mittels des Generic Work Context Contract Security-Information für die Inbound Kommunikation, die er vom EIS erhalten hat, an den Application Server zu übergeben.

## BeanConnect Überblick

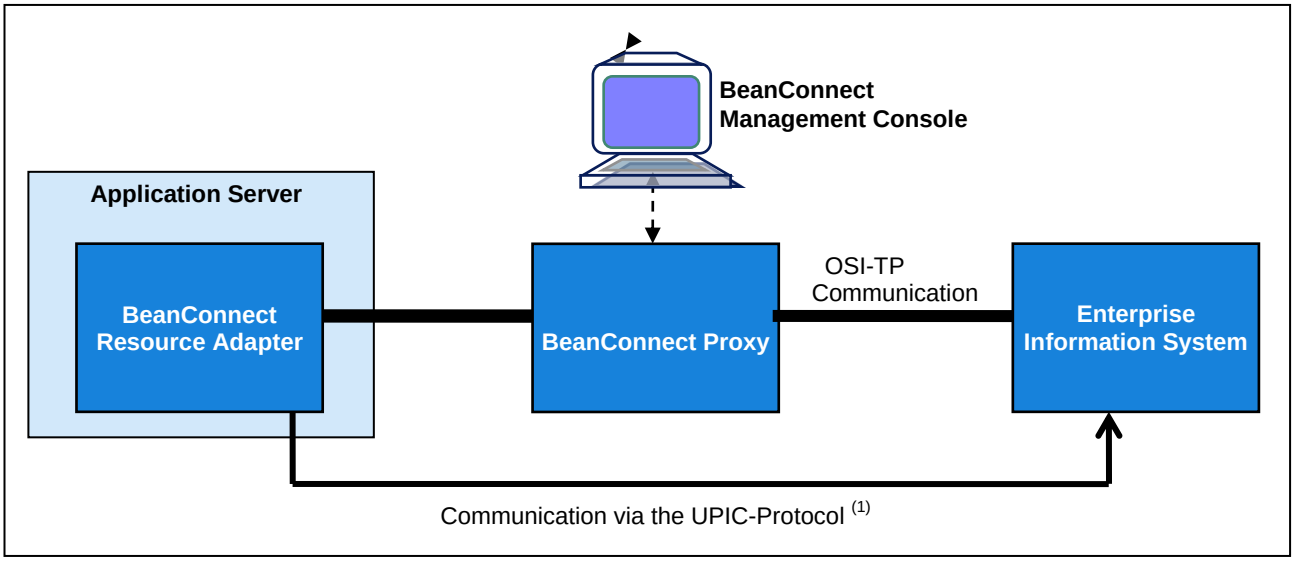
### Motivation

Mit openUTM hat Fujitsu ein hervorragendes Produkt, das in den Punkten Transaktionen, Skalierbarkeit und Stabilität sehr gute Leistungsmerkmale hat. Darüber hinaus ist openUTM durch die Unterstützung von OSI-TP bestens für den Einsatz in heterogenen, transaktionalen Umgebungen geeignet.
Basierend auf dieser technologischen Erfahrung bietet BeanConnect eine JCA konforme Resource Adapter Implementierung, die den Zugriff auf openUTM bzw. CICS basierte Anwendungen in der Welt der Java EE Application Server ermöglicht.

### Komponenten

Die BeanConnect Software besteht aus den folgenden Komponenten:

- **BeanConnect Resource Adapter**
  Der BeanConnect Resource Adapter implementiert die JCA 1.6 -Schnittstellen.
  Als konformer JCA-Adapter wird er in den Application Server deployt und läuft daher innerhalb des Application Servers ab.
  Für Application Server, die nur JCA1.5 unterstützen, wird ein separater BeanConnect Resource Adapter ausgeliefert, der nur die JCA 1.5 Schnittstellen implementiert.
- **BeanConnect Proxy**
  Der BeanConnect Proxy stellt die Funktionalität einer Protokollmaschine sowie Funktionen zur Transaktionssteuerung bereit. Er kann als intelligenter Gateway betrachtet werden. Er kommuniziert mit dem Resource Adapter, der im Application Server läuft, auf der einen Seite und mit dem EIS-Partner auf der anderen Seite.
- **BeanConnect Management Console**
  Die BeanConnect Management Console (MC) ist eine Java-basierte grafische Benutzeroberfläche für die Konfiguration und Administration von BeanConnect Proxies, die auf dem gleichen oder auf entfernten Rechnern ablaufen.
  Zusätzlich zur graphischen Administration ist eine Administration per Skripts möglich, womit die Automatisierung von Administrationsaktivitäten unterstützt wird. Als Skriptsprache wird Jython verwendet.
  Bei der Konfiguration werden mit der Management Console alle für die Kommunikation zwischen dem BeanConnect Resource Adapter und dem EIS notwendigen Daten hinterlegt, auf die zur Laufzeit zugegriffen wird.
  Administrativ kann mit der Management Console der BeanConnect Proxy gestartet/beendet und überwacht sowie zur Diagnoseunterstützung eingesetzt werden.




(1): Kommunikation via UPIC Protokoll d.h. ohne den Proxy ist nur bei EIS openUTM möglich.

Details zur Konfiguration:

- Auf dem EIS System muss keine Software zur Nutzung von BeanConnect installiert werden (d.h. BeanConnect ermöglicht eine **nicht-invasive Integration**).
- Der Application Server mit dem BeanConnect Resource Adapter, der BeanConnect Proxy sowie die Management Console können sowohl gemeinsam auf einem Rechner als auch auf getrennten Rechnern ablaufen.

**Datenaustausch**

Als JCA 1.6 Resource Adapter unterstützt BeanConnect sowohl die Outbound Kommunikation (d.h. vom Application Server zum EIS System) als auch die Inbound Kommunikation (d.h. vom EIS System zum Application Server). Beide Kommunikationsarten können sowohl synchron, d.h. mit Warten auf eine Antwort, als auch asynchron, d.h. ohne Antwortmöglichkeit genutzt werden.
Für Inbound und Outbound Kommunikation werden jeweils zwei Interfaces angeboten:

- Das BeanConnect spezifische API (**B**ean**C**onnect **I**nterface, kurz BCI)
- Das von der JCA spezifizierte **C**ommon **C**lient **I**nterface, kurz CCI.

**Transaktionen**

Wird für den Datenaustausch das OSI-TP Protokoll verwendet, können Transaktionen transparent für den EJB Programmierer sowohl vom Application Server ins EIS System propagiert (= Outbound Kommunikation) als auch vom EIS System in den Application Server importiert werden (= Inbound Kommunikation).

**Sicherheit**

Es werden je nach Kommunikationsszenario unterschiedliche Zugriffskontrollen unterstützt.

**Outbound Kommunikation:**
Zur Berechtigungsprüfung können Benutzerkennungen und Passwörter zur Prüfung an das EIS System weitergeleitet werden.
Zwei unterschiedliche Optionen werden angeboten:

- **container based security**:
  Per Konfiguration können Benutzer und Passwort für die Prüfung durch das EIS festgelegt werden.
- **application based security:**
  Am API werden vom EJB Programmierer mit der Klasse net.fsc.jca.communication.PasswordCredential Benutzer und Passwort festgelegt und beim Aufruf der Methode getConnection() als Parameter übergeben.

**Inbound Kommunikation:**

Wenn der EIS Partner bei der Inbound Kommunikation Benutzerkennungen und Passwörter übergibt, so werden diese über den Security Work Context (JCA 1.6) in den Application Server propagiert, sofern die Prüfung der Daten durch den BeanConnect Proxy erfolgreich war.

**Encoding (Codekonvertierung)**

Zur Umsetzung der ausgetauschten Daten, die in der Java-Umgebung und im EIS verschieden dargestellt werden müssen, gibt es je nach Einsatzszenario unterschiedliche Verfahren zur Codeumsetzung.

- Für **abdruckbare Zeichen** (Strings) kann sowohl ein automatisches Encoding auf Basis von **mitgelieferten Code-Tabellen** aktiviert als auch für Spezialfälle **eigene Codeumsetzungstabellen** vom EJB Programmierer festgelegt werden.

- Für in **Cobol** spezifizierte **Datenstrukturen**, die sowohl druckbare Zeichen als auch binäre Daten enthalten, bietet BeanConnect **leistungsfähige Werkzeuge** an, die sowohl **automatische Umsetzungsroutinen** als auch **Java Methoden** für den komfortablen Zugriff aus dem Java-Programm auf die konvertierten Datenfelder **generieren** können.
- Für weitere **Spezialfälle**, z.B. für in Assembler oder C definierte Datenstrukturen, kann mittels des Interfaces net.fsc.jca.communication.**ByteContainer** ein Satz von passenden **Umsetzungsmethoden selbst implementiert** werden.

  In einer Implementierung dieses Interfaces kann der Programmierer selbst flexibel entscheiden, wie die Daten umzusetzen sind.
  Die Methode getBytes() des ByteContainers wird von BeanConnect vor dem Senden an das EIS System aufgerufen und die Methode setBytes() wird entsprechend nach dem Empfang von Daten aufgerufen.

**Paralleles Request/Response Modell**

Im BeanConnect Resource Adapter kann ein paralleles Request/Response Modell verwendet werden.
Dieses Vorgehen ist wesentlich performanter als die sequentielle Abfrage der EIS Systeme.
In BeanConnect wird für die parallele Verarbeitung der Begriff "Connection Groups" verwendet. Man fasst hierbei mehrere Verbindungen (EISConnections) zu einer Gruppe zusammen. Nachdem alle für das Senden benötigten Parameter auf den einzelnen Verbindungen gesetzt sind, ruft man die Methode execute() auf der EISConnectionGroup auf. Es werden dann alle Requests an die beteiligten EIS Systeme abgesetzt. Der BeanConnect Proxy sendet diese Requests parallel an alle Partnersysteme und wartet auf die Antworten. Ist die letzte Antwort verfügbar, dann kehrt der execute() Aufruf zurück.
Im Anhang 2 (Code Beispiele) wird die Verwendung von Connection Groups dargestellt.

## Die Neuheiten von BeanConnect V3.0 gegenüber V2.1

### Resource Adapter

- Neuer Standard Application Server

  Der Standard Application Server von BeanConnect ab V3.0 ist Oracle™ WebLogic Server (bis V2.1: Oracle Internet Application Server).
- JCA 1.6

  BeanConnect V3.0 kann mit Application Servern zusammenarbeiten, die die Spezifikation

  JCA 1.5 oder JCA 1.6 unterstützen. Mit BeanConnect V3.0 wird der Resource Adapter in zwei Varianten ausgeliefert. Standardmäßig kann der BeanConnect Resource Adapter für Application Server verwendet werden, die JCA 1.6 unterstützen (z.B. Oracle™ WebLogic Server).

  Für Application Server, die nur JCA1.5 unterstützen, wird ein separater BeanConnect Resource Adapter ausgeliefert.
- Lastverteiler Funktionalität

  Nutzung der JConnect Lastverteiler Funktionalität bei der Outbound-Kommunikation mit einer openUTM-Cluster-Anwendung über das UPIC-Protokoll.

  Für Outbound Kommunikation über OSI TP wird die UTM Lastverteiler Funktionalität (MASTER-OSI-LPAP) des BeanConnect Proxy verwendet.

### Management Console

- Command Line Interface (CLI)
  Mit BeanConnect V3.0 ist, zusätzlich zur graphischen Administration, eine Administration per Skripts möglich. Als Skriptsprache wird Jython verwendet
- UTM Cluster als EIS Partner
  Es ist möglich eine UTM-Cluster-Anwendung als EIS-Partner zu konfigurieren.
- Generierung EIS Partner
  Proxies können mit unterschiedlichen APT konfiguriert werden, damit bei einer Kopplung einer UTM-Anwendung mit mehreren Proxies jeder Proxy einen eigenen Application Process Title (APT) erhalten kann.
- EIS Partner im Eigenschaftsdialog eines Outbound Communication Endpoints
  Es ist nun möglich im Eigenschaftsdialog des Outbound Communication Endpoints den EIS-Partner direkt zu ändern.

### MCCmdHandler

- Windows, Freischaltung des MCCmdHandler Server-Ports
  Da es beim Betrieb des BeanConnect MCCmdHandler zu Problemen kommen kann, wenn die Windows Firewall aktiviert ist, ist es nun möglich, während der Installation den MCCmdHandler Server-Port freizuschalten.
- Windows, Dienstüberwachung
  Beim Überwachen des Dienstes auf unerlaubte Meldungen wird eine Meldungsbox ausgegeben.

## Fazit

Die Connector Architektur erweitert die bewährte Java EE Plattform zur Integrationsplattform für EIS Systeme. Sie erlaubt eine skalierbare Nutzung der Enterprise Ressourcen in einem modernen Umfeld, ohne dabei die Datenintegrität und die Sicherheit der EIS Systeme zu gefährden.
BeanConnect bietet die geeignete Unterstützung, um transaktionale Systeme wie openUTM und CICS mit Hilfe der Connector Architektur in eine Java EE basierte Plattform zu integrieren.

Detailliertere Informationen findet man
- In den Anhängen „Fachbegriffe“ und „Code Beispiele“ dieses Dokumentes.
- Auf der BeanConnect Webseite (siehe unten.).
- Anderen Quellen im Internet (Auswahl siehe unten).

## Weiterführende Quellen im Internet

- BeanConnect
  http://www.de.ts.fujitsu.com/beanconnect
- openUTM
  http://www.de.ts.fujitsu.com/openUTM
- openSEAS
  http://www.de.ts.fujitsu.com/openSEAS
- Oracle WebLogic
  http://www.oracle.com/de/products/middleware/cloud-app-foundation/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/

## Anhang 1: Fachbegriffe

In diesem Anhang werden die wichtigsten Fachbegriffe erklärt, die nicht grundsätzlich als bekannt vorausgesetzt werden, aber für das Verständnis der nachfolgenden Beispiele hilfreich sind.

Die Reihenfolge ist so gewählt, dass bei weiteren Begriffserklärungen vorausgesetzte Begriffe zuerst erläutert werden.

**EIS (Enterprise Information System)**

Ein Enterprise Information System (EIS) auch EIS-Partneranwendung genannt, ist das Partner-System, mit dem über BeanConnect kommuniziert wird.
BeanConnect V3.0 unterstützt zwei Ausprägungen von EIS:
- Anwendungen auf Basis von openUTM von Fujitsu.
- Anwendungen auf Basis von CICS von IBM.

Bei Inbound Kommunikation werden zwei weitere Ausprägungen von EIS unterstützt:
- UPIC-Anwendung
- openUTM-Socket- oder RFC1006-Anwendung

**BeanConnect Interface (BCI)**

Das BeanConnect Interface, kurz BCI, basiert auf dem Connection Contract der JCA und ermöglicht die Kommunikation mit den von BeanConnect unterstützten EISen auf Basis von openUTM oder CICS über einfach zu handhabende Schnittstellen.

Die Beispiele in diesem Dokument sind alle BCI basiert, da dieses API die von BeanConnect bevorzugte Zugriffsmethode auf ein EIS darstellt.

**ProxyURL**

Die ProxyURL legt die Zuordnung des deployten Resource Adapters zum BeanConnect Proxy fest.

Die ProxyURL wird beim Deployment des Ressource Adapters festgelegt.

Die Definition erfolgt nach dem Muster `oltp://<host>:<port>/<name>`
Die Bestandteile haben folgende Bedeutung:
- `<host>` Rechner, auf dem der Proxy-Container installiert ist
- `<port>` Portnummer des Proxy-Containers +4
- `<name>` Anwendungsname des Proxy-Containers (BCU<port>)

Standardwert für die ProxyURL ist:

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

d.h. der BeanConnect Proxy befindet sich auf demselben Rechner wie der Resource Adapter und ist über Port 31004 zu erreichen.

**Outbound Service**
Ein Outbound-Service stellt einen vom EIS-Partner zur Verfügung gestellten Service (Transaktionscode (TAC)) dar.

Outbound Services werden mit der BeanConnect Management Console definiert.

**Communication Endpoint**

Ein Communication Endpoint wird für die Outbound Kommunikation benötigt.
In der Definition des Outbound Communication Endpoint wird der symbolische Service-Name auf einen realen Service-Namen einer EIS-Partneranwendung abgebildet.

Outbound Communication Endpoints werden mit Hilfe der Management Console definiert.

Ein Outbound **Communication Endpoint** hat folgende Eigenschaften:
**Name:** Gibt den symbolischen Namen des Communication Endpoints an.
**EIS Partner:** EIS-Partner, zu dem der Communication Endpoint gehört.
Hinweis: Der EIS-Partner muss zuvor eingerichtet worden sein.
**Partner Service:** Tatsächlicher Name des Services innerhalb des EIS-Systems. (d.h. Transactioncode von openUTM bzw. CICS)
Hinweis:
Der Service muss zuvor als Outbound-Service definiert werden.

**Connection Factory**

Mit der Methode getConnection() einer Connection Factory können Sie ein Objekt für eine Verbindung zum EIS erhalten.

BeanConnect bietet die folgenden Connection Factories:
- net.fsc.jca.communication.EISOltpConnectionFactory
  Kommunikation über OSI-TP mit openUTM oder über LU6.2 mit CICS.
- net.fsc.jca.communication.EISUpicConnectionFactory
  Kommunikation über das openUTM UPIC Protokoll mit einer openUTM Anwendung.
- Beide Connection Factories sind Erweiterungen der Basisklasse
  net.fsc.jca.communication.EISConnectionFactory

Sofern nur die Basisklasse EISConnectionFactory verwendet wird, weil keine Connection Factory spezifischen Funktionen benötigt werden, kann das verwendete Protokoll durch Anpassung der Konfigurationdateien für den Resource Adapter (Standard und WebLogic-spezifischer Deployment Descriptor ra.xml und weblogic-ra.xml) geändert werden, ohne dass der Code der Enterprise Java Bean (EJB) geändert werden muss.

**ConnectionURL**

Für die Outbound Kommunikation über OSI-TP gibt die Konfigurations-Property ConnectionURL den Namen des Outbound Communication Endpoints an, der die Verbindung zu einem EIS-Partner repräsentiert.

Die ConnectionURL wird beim Deployment einer Connection Factory festgelegt.

Die Definition erfolgt nach dem Muster `<type>://<name>`
Die Bestandteile haben folgende Bedeutung:
- `<type>` Typ des EIS Partners, mögliche Werte:
  - `utm` der EIS Partner ist vom Typ openUTM.
  - `cics` der EIS Partner ist vom Typ CICS.
  - `xatmi.rr` der EIS Partner ist vom Typ XATMI und die Kommunikation wird nach dem Request/Reply-Paradigma durchgeführt.
  - `xatmi.cv` der EIS Partner ist vom Typ XATMI und die Kommunikation wird nach dem Conversational-Paradigma durchgeführt.
- `<name>` Name eines Outbound Communication Endpoints, wie er in der Management Console definiert wurde

Standardwert ist: utm://outboundCommunicationEndpoint

**OLTP Message-Driven Bean**

OLTP Message-Driven Beans sind JCA-konforme Message-Endpoint-Anwendungen, die von BeanConnect unterstützt werden.
Eine EIS-Anwendung kann per Inbound Communication EJBs aufrufen, die in einem Application Server deployt sind und das OLTP Message-Driven Bean-Interface implementieren.

**Inbound Message Endpoint**

Ein Inbound Message Endpoint ist der Endpunkt der Inbound Kommunikation im Java EE Application Server.
Für jeden Inbound Message Endpoint im Application Server muss mit Hilfe der BeanConnect Management Console ein gleichnamiger Inbound Message Endpoint im BeanConnect Proxy konfiguriert werden. Jedem Inbound Message Endpoint muss mindestens ein Inbound Service zugeordnet werden.

**Inbound Service**

Ein Inbound Service ist ein Service, den ein EIS Partner bei Inbound Kommunikation adressiert.

## Anhang 2: Code Beispiele

Im folgenden Abschnitt wird die Nutzung der BeanConnect Schnittstellen an einfachen ausgewählten Szenarien veranschaulicht.

**BeanConnect-spezifische Interfaces für Outbound Kommunikation**

**Beispiel 1: Einschritt-Service**

Hier ein Codefragment aus einer EJB, die eine EISConnectionFactory nutzt, um den Einschritt Service "KDCINF" mit dem Parameter "STAT" in einer openUTM Anwendung auf einem BS2000 Rechner aufzurufen:

```
Context ctx = new InitialContext();

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

eisConnection = eisConnectionFactory.getConnection();

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

Der zugehörige Eintrag in der Deploymentdatei `ejb-jar.xml` der Enterprise JavaBean sieht folgendermaßen aus:

**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>
```

Wird **transaktional** mit openUTM auf BS2000 **über den BeanConnect Proxy** kommuniziert, so sieht die Weblogic Server spezifische Deploymentinformation für den BeanConnect Resource Adapter folgendermaßen aus:

**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>
```

Wird **nicht-transaktional** über das openUTM UPIC Protokoll **direkt mit openUTM auf BS2000** kommuniziert, so sieht die WebLogic Server spezifische Deploymentinformation für den BeanConnect Resource Adapter für die Properties `connectionfactory-interface` und `ConnectionURL` etwas anders aus:

**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>
```

**Beispiel 2: Paralleles Request/Response Modell**

Hier zur Veranschaulichung ein Codefragment, bei dem zwei Services parallel aufgerufen werden und mit der execute Methode auf die Antworten gewartet wird:

```
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();
```

**Beispiele zur Outbound Security**

In den folgenden Beispielen zur Nutzung der Security-Funktionen bei Outbound Kommunikation werden die zwei möglichen Optionen aufgezeigt.

1. **container based security:**

   Bei WebLogic werden die Credential Mappings für Container Managed Secutiry für Outbound Communication über die WebLogic Administrations Console festgelegt (siehe BeanConnect Handbuch Seite 120).

2. **application based security:**

   Auszug aus Java-Code:

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

**Beispiel zur Inbound Kommunikation**

Grundsätzlich wird für die Inbound Kommunikation BeanConnect intern eine Socket für die Kommunikation zwischen dem BeanConnect Proxy und dem BeanConnect Resource Adapter verwendet.
Dieser Port wird mit als **inboundListenerPort** in der zugehörigen Deploymentinformation des Ressource Adapters konfiguriert:

**InboundListenerPort Definition in der Deskriptordatei ra.xml**

```
<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>
```

Um mit einer OLTP Message-Driven Bean kommunizieren zu können, sendet eine EIS-Anwendung eine Nachricht an einen Inbound Service, der einem mit Hilfe der Management Console konfiguriertem Inbound Message Endpoint zugeordnet ist. Die Property messageEndpoint der OLTP Message-Driven Bean, die in der Datei ejb-jar.xml definiert ist, muss mit dem Namen des Inbound Message Endpoints übereinstimmen, der im Proxy definiert ist.

Zur Laufzeit wird die Nachricht vom BeanConnect Proxy an die OLTP Message-Driven Bean weitergeleitet, dessen Message-Endpoint-Namen per Konfiguration mit dem verwendeten Service-Namen verknüpft ist.

Die MessageDrivenBean, die die eingehenden Nachrichten verarbeiten soll, muss eines der folgenden BeanConnect spezifischen Interfaces implementieren:

**Listener Interfaces für die MessageDrivenBean**

```
net.fsc.jca.communication.OltpMessageListener       (synchron)
net.fsc.jca.communication.AsyncOltpMessageListener  (asynchron)
```

In der onMessage Methode der MessageDrivenBean Implementierung können dann die eingehenden Nachrichten gelesen und verarbeitet werden. Anbei ein Beispiel für einen einfachen synchronen Service:
**MessageDrivenBean Implementierung**

```
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<OltpMessagePart> 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);
}
}
```

Zur obigen Implementierung gehören die folgenden Deploymentdaten:

**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>
```

Man beachte: die Property `encodingActive` wurde hier auf `true` gesetzt und es wurde ein Encoding angegeben ("OSD_EBCDIC_DF04_15"). D.h.: werden zwischen EIS System und Application Server Strings ausgetauscht, so werden die Zeichen automatisch konvertiert.

Hier noch die WebLogic Server spezifische Deployentinformation für das obige Beispiel:

**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>
```



**Herausgegeben durch:**

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