

COMPORSYS Connector for IBM IMS

Version 1.5

Installation and Reference Guide

Contents

Preface	4
Installation	5
Requirements	5
Contents of IMS_CONNECTOR.ZIP	6
J2EE 1.3 compliant Application Server.....	7
Installation steps.....	7
Deployment and Configuration	8
J2EE 1.3 compliant Application Server.....	8
Configuration Parameters	8
ConnectionFactory	8
Connection	10
COMPORSYS CodeGen	13
Processing COBOL Copybooks	13
Naming Conventions	14
Supported COBOL keywords.....	14
Data type mapping	14
Using HIGH-VALUE and LOW-VALUE.....	15
Generating access classes for IMS transactions.....	15
Design of access classes.....	17
Programming Pattern.....	18
Transaction Management	19
Programming with the UCI	20
Establishing a connection	20
Managed Environment.....	20
Unmanaged Environment	21
Accessing the UCI	22
Synchronous transaction call.....	22
Properties of an IMS request.....	23
Send-only transaction call	24
Asynchronous transaction call	24
Programming with the CCI	26
COMPORSYS Connector for IBM IMS	2

Contents

- Setting up the connector for CCI 26
- Establishing a connection 26
 - Managed Environment..... 26
 - Unmanaged Environment 27
- Call a transaction using CCI 28
- Send-only transaction call 29
- Asynchronous transaction call 29
- Properties of the InteractionSpec bean 30
- Retrieves internal UCI properties through CCI 31
- Further information..... 33**

List Of Tables

- Table 1: Directory Structure 6
- Table 2: Required Java Archives..... 7
- Table 3: ImportCobol Parameters..... 13
- Table 4: Data type mapping between COBOL and Java 14
- Table 5: GenIMS Parameters 16
- Table 6: Influence of AutoCommit on Transaction Management 19
- Table 7: Properties of an IMS request 24
- Table 8: Properties of an IMS InteractionSpec 31

Preface

The COMPORSYS Connector for IBM IMS enables fast and easy integration of IMS transactions and programs in Java applications. It can be used in a Client/Server Environment, for example in a Java application (*Unmanaged Environment*), as well as in an Application Server of the Java 2 Enterprise Edition (J2EE)[™] (*Managed Environment*).

The COMPORSYS Connector includes

- a resource adapter for IBM IMS according to the J2EE[™] Connector Architecture Specification 1.0.
- the code generator COMPORSYS CodeGen
 - to create Java records from COBOL Copybooks for data exchange with IBM IMS. These *records* allow an easy access to the Communication Area and carry-through the necessary mapping of data types.
 - to create access records for IMS programs, often making your own programming superfluous.

The COMPORSYS Connector supports the standard Common Client Interface (CCI) of the Java Connector Architecture as well as COMPORSYS User Call Interface (UCI), that allows an effortless access to all features of the Connector.

Installation

Requirements

- IBM IMS Connect installed and configured.
(<http://www.ibm.com/software/data/ims>)
- Application Server with support for Java Connector Architecture.
We recommend BEA WebLogic Server.
(<http://www.bea.com>)
- ANT from Apache to configure the resource adapter and to build the examples.
(<http://jakarta.apache.org/ant/index.html>)

Contents of IMS_CONNECTOR.ZIP

The IMS_CONNECTOR.ZIP contains all files needed for the COMPORSYS Connector for IBM IMS.

Included are:

- Resource adapter IMS_CONNECTOR.RAR
- Required libraries
- Documentation and examples

ims_connector\	COMPORSYS Connector root directory
ims_connector\deploy	Resource Adapter IMS_CONNECTOR.RAR
ims_connector\lib	All required java archives
ims_connector\META-INF	Deployment Descriptors
ims_connector\doc	Documentation
ims_connector\templates	Templates for various configuration files
ims_connector\examples	Application sample files

Table 1: Directory Structure

J2EE 1.3 compliant Application Server

The COMPORSYS Connector for IBM IMS contains a JCA compliant resource adapter. Therefore it can be easily deployed into the BEA WebLogic Server or later and other Application Servers supporting the Java Connector Architecture.

Installation steps

Follow these steps to install the COMPORSYS Connector for IBM IMS.

1. Copy the COMPORSYS Connector installation files

Unzip the provided IMS_CONNECTOR.ZIP into the root directory of the BEA WebLogic Server (p.e. c:\bea\weblogic).

2. Editing the Server CLASSPATH

Add the IMS_CONNECTOR.JAR to the server CLASSPATH.

IMS_CONNECTOR.JAR	COMPORSYS Connector for IBM IMS
CONNECTOR_FOUNDATION.JAR	COMPORSYS Connector Foundation Classes

Table 2: Required Java Archives

Example:

```
startWebLogicServer.cmd:
...
set CONNECTOR=.\ims_connector\lib\ims_connector.jar;
    .\ims_connector\lib\connector_foundation.jar

set CLASSPATH=.;.\lib\weblogic_sp.jar;.\lib\weblogic.jar;%CONNECTOR%
...
```

Deployment and Configuration

J2EE 1.3 compliant Application Server

To deploy the IMS Connector either copy the IMS_CONNECTOR.RAR from IMS_CONNECTOR/DEPLOY into the APPILCATIONS directory of your server or use the management console.

The configuration parameters are stored in the deployment descriptor "ra.xml" into the IMS_CONNECTOR.RAR. To rebuild the archive use command „ANT pack" in the root directory of the connector installation.

To configure the connector use the tools that comes with your application server. Please refer to the server documentation for further information.

Configuration Parameters

The COMPORSYS Connector uses parameters to adapt to features of the target environment.

Structure: Deployment Descriptor Property (Type)

ConnectionFactory

The configuration of the ConnectionFactory of the COMPORSYS Connectors is done by means of the following parameters.

ServerName (String)

Name of the IBM IMS Connect server.

It can be either a DNS name or a IP address provided.

PortNumber (Integer)

TCP/IP port on which IMS Connect is listening.

ConnectionURL (String)

Specify the hostname and port as "host:port". Don't use this property in conjunction with the **ServerName** and **PortNumber**.

You can provide a comma-separated list to turn on the connectors failover capabilities, i.e. "host:2080,host_b:2080:host_b:2081".

The connector uses this first available address from left to right.

If a host or port goes down after the connector has a connection established, it transparently creates a new connection to the first available address.

If no host at all is reachable, an exception is thrown to the client. Otherwise the failover is documented in the trace file at WARN and EXCEPTION level.

CCISupport (Boolean)

Determines which interface the COMPORSYS Connector should offer.

Valid values are: `true`, `false`

The default value is: `false`

Remark

The interface of the `ConnectionFactory` is either type

`javax.resource.cci.ConnectionFactory`

or `de.comporsys.connector.ims.ConnectionFactory`

Multisegmentformat (Boolean)

Indicates, if the IMS Connect multisegment format should be used.

Valid values are: `true`, `false`

The default value is: `false`

Id (String)

Sets the ID of the IMS Connect UserExit.

The default value is: *SAMPLE*

Connection

The following parameters define default values for establishing connections to IBM IMS.

Datastore (String)

Sets the name of the IMS Datastore to connect to.

UserName (String)

The username under which requests to IBM IMS are executed.

Remark

This default value is only used if there is no other value given when establishing a connection.

The default value is empty.

Password (String)

The user password under which requests to IBM IMS are executed.

Remark

This default value is only used if there is no other value given when establishing a connection.

The default value is empty.

Usergroup (String)

The security group the user belongs to.

Remark

This default value is only used if there is no other value given when establishing a connection.

The default value is empty.

AutoCommitMode (Boolean)

Defines the transaction handling of the COMPORSYS Connector.

With `autoCommit = false` the a IMS conversational transaction will be executed. Refer to the IMS Connect documentation for further information.

Valid values are: `true`, `false`

Default value is: `false`

LogLevel (Integer)

The level of messages the COMPORSYS Connector is recording.

Remark

The value should only be changed for problem-shooting purposes as the result is a reduction of performance.

Valid values are:

0: LOG_LEVEL_DEBUG

1: LOG_LEVEL_TRACE

2: LOG_LEVEL_WARNING,

3: LOG_LEVEL_ERROR

Default value is: 3 (LOG_LEVEL_ERROR)

SocketSendBufferSize (Integer)

Size in bytes of the TCP/IP socket's SendBuffer in bytes.

Remark

Use with caution. Incorrect setting will reduce the performance.

Default value is: -1 (Operating System default)

SocketReceiveBufferSize (Integer)

Size in bytes of the TCP/IP socket's ReceiveBuffer in bytes.

Remark

Use with caution. Incorrect setting will reduce the performance.

Default value is: -1 (Operating System default)

BufferSize (Integer)

Size of the adapters internal buffer in bytes to send to and receive from the socket.

Remark

Use with caution. Incorrect setting will reduce the performance.

Default value is: 65536

LocalAddress (String)

Socket will be bound to this local IP address. Use this property, if your server has multiple IP addresses (multihoming)

COMPORSYS CodeGen

Use the COMPORSYS Connector code generation tool COMPORSYS CodeGen to:

- create "Java records" from COBOL Copybooks needed for data exchange with IBM IMS. A record allows access to the COMMAREA and converts data types between COBOL and Java.
- create "Access Classes" to execute IMS transactions.

Visit www.comporsys.de to get an Eclipse-Plugin for the COMPORSYS CodeGen Tool.

Processing COBOL Copybooks

Using the description of the COMMAREA modelled after a COBOL Copybook the COMPORSYS Connector allows the creation of corresponding Java classes (*records*).

A record represents the COMMAREA, that the COMPORSYS Connector exchanges with IBM IMS. A record allows a standardized access to the individual components of the COMMAREA and converts them into corresponding COBOL data types.

Command:

```
java de.comporsys.codegen.ImportCobol
    <cobol-source> <output-root> <package-name> <codepage>
```

Optional:

```
<codepage>
```

<cobol-source>	Name of the COBOL Copybook
<output-root>	Output directory of the generated class
<package-name>	Package-Name of the generated class
<codepage>	The IMS Server's code page (Default: cp273: EDCDIC (German))

Table 3: ImportCobol Parameters

Naming Conventions

1. The Copybook must start with a structured type.
2. The name of the first structured type defines the name of the class generated.
3. Every hyphen "-" in a name will be substituted by an underline "_".
4. Lowercase and uppercase letters are pertinent.

Supported COBOL keywords

The COBOL Copybooks could contain the following COBOL keywords:

- PIC, PICTURE, OCCURS, REDEFINES, A, X, 9, V, S, 0, DISPLAY, COMP, COMP-3, USAGE, IS, ARE, BINARY, PACKED-DECIMAL, DEPENDING, ON, TIMES, POINTER, THROUGH, THRU, 88 Levels

Data type mapping

The following rules for data type mapping are valid for all data types with the Usage-Clause DISPLAY, COMP and COMP-3.

PIC A	java.lang.String
PIC A(n)	java.lang.String
PIC X	java.lang.String
PIC X(n)	java.lang.String
PIC .X.	java.lang.String
PIC 9(n=0..4)	int
PIC 9(n=5..9)	int
PIC 9(n=10..18)	long
PIC S9(n=0..4)	int
PIC S9(n=5..9)	long
PIC S9(n=10..18)	long
PIC 9(n)V(m)	java.math.BigDecimal
PIC S9(n)V(m)	java.math.BigDecimal

Table 4: Data type mapping between COBOL and Java

Using HIGH-VALUE and LOW-VALUE

The COMPORSYS record and data type classes are supporting Java equivalents to the COBOL HIGH-VALUE and LOW-VALUE constructs.

Each record field can be set to and tested for those special values.

- use `set...(de.comporsys.connector.datatype.Record.HIGH_VALUE)` on the generated record class to set the field to HIGH-VALUE.
- use `set...(de.comporsys.connector.datatype.Record.LOW_VALUE)` on the generated record class to set the field to LOW-VALUE.
- use `get...Value()` to get the current value of the field as an object and use `equals(de.comporsys.connector.datatype.Record.HIGH_VALUE)` to test for HIGH-VALUE.

use `get...Value()` to get the current value of the field as an object and use `equals(de.comporsys.connector.datatype.Record.LOW_VALUE)` to test for LOW-VALUE.

Generating access classes for IMS transactions

The COMPORSYS Connector includes a tool to generate access classes for IMS transactions.

An access class

- encapsulate the call to a IMS transactions using CCI or UCI,
- contains methods to create parameter classes for IMS transactions,
- can be extended through inheritance.

```
java de.comporsys.codegen.GenIMS
    <ims-transactions> <class-name>
    <input-record-class> <output-record-class>
    <output-root> <package-name>
```

<ims-transaction>	Name of the IMS transaction
<class-name>	Name of the generated access class
<input-record-class>	Name of the input record class
<output-record-class>	Name of the output record class
<output-root>	Output directory of the generated class
<package-name>	Package-name of the generated class

Table 5: GenIMS Parameters

1. Lowercase and uppercase letters are pertinent.
2. The actual existence of the record classes specified are **not** verified at the time it is created.

The generated class contains a commented method `main(String[] args)` that can be used for testing purposes.

Design of access classes

```
public class <CLASS-NAME> {  
  
    public <CLASS-NAME>( ConnectionFactory cf )  
  
        creates an access object, that uses the ConnectionFactory. The  
        ConnectionFactory of both the CCI and UCI can be used.  
  
    public <INPUT-RECORD> createInputRecord()  
  
        creates an instance of the input record.  
        Overwrite if needed.  
  
    public <OUTPUT-RECORD> createOutputRecord()  
  
        creates an instance of the output record.  
        Overwrite if needed.  
  
    public InteractionSpec createInteractionSpec()  
  
        creates and sets up an instance of the interaction spec. (CCI only)  
        Overwrite if needed.  
  
    public Request createRequest( Statement s )  
  
        creates and sets up an instance of the request. (UCI only)  
        Overwrite if needed.  
  
    public <OUTPUT-RECORD> execute( <INPUT-RECORD> input )  
  
        executes the request with the specified input record and returns the  
        output record.  
        Do not overwrite.  
  
}
```

Programming Pattern

Using access classes simplifies the necessary steps to call a IMS program:

1. Find the ConnectionFactory using JNDI
2. Get an instance of the access class
3. Get an instance of the input record
4. Set the values of the input record
5. Call the execute() method of the access object
6. Retrieve the result in the output record

Transaction Management

The COMPORSYS Connector for IMS supports `NoTransaction` only. It is required to set the `AutoCommit` property of the `Connection` object to `true` .

If deployed as a JCA compliant resource adapter the transaction management is configured by tag `<transaction-support>` in "ra.xml".

For default all connections are created with the `AutoCommit` property value of the `ConnectionFactory` object. The property can also be set for every single connection.

- `public void setAutoCommit(boolean autocommit)`

The actual value can be retrieved by

- `public boolean getAutoCommit()`

Property	Influence on Transaction Management
<code>autoCommit=true</code>	<ul style="list-style-type: none"> ▪ Every call is executed in it's own Unit of Work. ▪ SYNC LEVEL 0 ▪ No integration into the JTS takes place in the Managed Environment. ▪ <code><transaction-support></code> must be set to <code>NoTransaction</code>
<code>autoCommit=false</code>	<ul style="list-style-type: none"> ▪ not supported

Table 6: Influence of AutoCommit on Transaction Management

Programming with the UCI

Even if using generated access classes is recommended, it may be necessary to program the COMPORSYS Connector directly.

Although the COMPORSYS Connector supports the standard Common Client Interface (CCI) you may want to use the Uses Call Interface (UCI) for special purposes.

Establishing a connection

The connection to IMS is created by the *ConnectionFactory* object.

The creation of a *ConnectionFactory* object is depending on whether the COMPORSYS Connector is being used in a *Managed Environment* or *Unmanaged Environment*.

Managed Environment

The application server is responsible for the creation of the *ConnectionFactory* object in accordance with properties for configuration and publishing in the Naming Service.

The client gets the *ConnectionFactory* object using the Java Naming and Directory Interface (JNDI) for immediate use.

```
import javax.naming.*;
import de.comporsys.connector.ims.* ;
.
.
.
InitialContext context = new InitialContext();
ConnectionFactory cf = (ConnectionFactory) context.lookup("IMS");
Connection c = cf.getConnection();
```

Unmanaged Environment

Using the COMPORSYS Connector outside of an Application Server the client must create, configure and manage the *ConnectionFactory* object.

The client uses `createConnectionFactory()` of the *ManagedConnectionFactory* to create a new *ConnectionFactory* object.

```
import de.comporsys.connector.ims.* ;
.
.
.
ManagedConnectionFactoryImpl mcf = new
    ManagedConnectionFactoryImpl();
mcf.setServerName("myhost");
mcf.setPortNumber("2004");
mcf.setDatastore("test");
ConnectionFactory cf = (ConnectionFactory)
    mcf.createConnectionFactory();
Connection con = cf.getConnection();
```

Accessing the UCI

The client uses the created connection to get the *Statement* object needed to access the UCI. All functions of the UCI can be called by using the *Statement* object.

```
import de.comporsys.connector.ims.* ;  
.  
.  
.  
Statement s = con.createStatement();
```

Synchronous transaction call

The *Request* object represents the call to a particular IMS program. The *Request* object is created with the name of the IMS program to call and the COMMAREA to be transferred. The parameterised *Request* object is executed using the *Statement* object.

```
Request request = s.createRequest( "TXNAME" );  
request.setCommArea( "Text", "cp273" );  
Reply r = s.executeQuery( request );
```

The COMPORSYS Connector offers three different methods to send a *Request* object:

- `public Reply executeQuery(Request r)`
A read-request is executed and the result is returned by the *Reply* object.
(The connector does not guaranty that the called program make any changes.
The use of `executeQuery` is for legibility only.)
- `public void executeUpdate(Request r)`
A write-request is executed that changes IMS resources. The Request object returns no data or data that is of no relevance to the client.
- `public Reply execute(Request r)`
A request is executed whose type is not further specified. The function is not different from `executeQuery(Request r)`.

Properties of an IMS request

The request class offers a set of properties. They allow you to configure the communication for your special needs.

Property	Meaning
Transaction	Provide the name IMS transaction to call
InteractionVerb	Specifies the communication model. SYNC_SEND_RECEIVE (default): Standard send-receive flow. SYNC_SEND: Send-only request. SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_MSG: RESUME TPIPE operation with IMS SINGLE option. SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_MSG_WAIT: RESUME TPIPE operation with IMS SWAIT option.
CommitMode	Specify either COMMIT_MODE_1 (default) or COMMIT_MODE_0
CommArea	Sets the data to transmit.
MultisegmentSupport	Set to <code>true</code> , if the transaction has a multisegment request or reply.
Timeout	Sets the milliseconds to wait for a reply (TCP/IP timeout).
Timer	Sets the time the IMS Connect timer value. Use it in conjunction with RECEIVE_ASYNCOUTPUT.
MultisegmentSupport	Set to <code>true</code> , if the transaction has a multisegment request or reply.
Lterm	Sets the name of the LTERM.

ClientId	The name of client. If not provided, the connector generates an unique id.
MaxSegmentSize	Comma separated list of all segment sizes. The last value is used for all following segments. The first value gives the size of the first segment, the <i>n</i> th value give the size if the <i>n</i> th segment. All segment sizes must be given including the LLZZ length (4 bytes)

Table 7: Properties of an IMS request

Send-only transaction call

The COMPORSYS Connector for IMS supports the IMS SEND ONLY protocol for IMS transactions without any reply message.

The SEND ONLY protocol requires the use of COMMIT_MODE_0.

```
Request request = s.createRequest( "TXNAME" );
request.setCommitMode( Request.COMMIT_MODE_0 );
request.setInteractionVerb( Request.SYNC_SEND );
request.setCommArea( "Text", "cp273" );
s.execute ( request );
```

Asynchronous transaction call

The COMPORSYS Connector for IMS supports the IMS RESUME TPIPE operation to receive asynchronous output from IMS.

The Connector supports the RESUME TPIPE options SINGLE and SINGLE_WAIT. Specify the desired option as InteractionVerb on the request object

```
Request request = s.createRequest( "TXNAME" );
request.setTimer(Request.TIMER_DEFAULT );
request.setClientId( "TPIPE" );
request.setInteractionVerb(
    Request.SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_MSG_WAIT);
request.setTransaction( "" );
request.setCommArea(new byte[0]) ;
Reply r = s.executeQuery( request );
```

The timer value specifies how long IMS waits for a message to arrive in the TPIPE.

Use the SINGLE option to return immediately if no message is available in the TPIPE.

```
Request request = s.createRequest( "TXNAME" );
request.setClientId( "TPIPE" );
request.setInteractionVerb(
    Request.SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_MSG);
request.setTransaction( "" );
request.setCommArea(new byte[0]) ;
Reply r = s.executeQuery( request );
```

Programming with the CCI

Setting up the connector for CCI

Set the **CCISupport** property of the `ManagedConnectionFactoryImpl` to **true** to enable the CCI.

Note: The CCI is implemented on top of the UCI, so refer to the corresponding topics their for further details if needed.

Establishing a connection

The connection to IMS is created by the *ConnectionFactory* object.

The creation of a *ConnectionFactory* object is depending on whether the COMPORSYS Connector is being used in a *Managed Environment* or *Unmanaged Environment*.

Managed Environment

The application server is responsible for the creation of the *ConnectionFactory* object in accordance with properties for configuration and publishing in the Naming Service.

The client gets the *ConnectionFactory* object using the Java Naming and Directory Interface (JNDI) for immediate use.

```
import javax.naming.*;
import javax.resource.cci.* ;
.
.
.
InitialContext context = new InitialContext();
ConnectionFactory cf = (ConnectionFactory) context.lookup("IMS");
Connection c = cf.getConnection();
```

Unmanaged Environment

Using the COMPORSYS Connector outside of an Application Server the client must create, configure and manage the `ConnectionFactory` object.

The client uses `createConnectionFactory()` of the `ManagedConnectionFactoryImpl` to create a new `ConnectionFactory` object.

```
import de.comporsys.connector.ims.* ;
import javax.resource.cci.* ;
.
.
.
ManagedConnectionFactoryImpl mcf = new
    ManagedConnectionFactoryImpl();
mcf.setServerName( "mainframe" ) ;
mcf.setPortNumber( 7000 ) ;
mcf.setDatastore( "IMSTEST" ) ;
mcf.setCodepage( "cp273" ) ;
mcf.setCCISupport( true ) ;
ConnectionFactory cf = (ConnectionFactory)
    mcf.createConnectionFactory();
Connection con = cf.getConnection();
```

Call a transaction using CCI

The *InteractionSpec* object is set with the name of the IMS transaction and the data to be transferred.

The parameterised *InteractionSpec* object is executed using the *Interaction* object.

```
import de.comporsys.connector.ims.cci.InteractionSpecImpl ;
import javax.resource.cci.* ;
.
.
.
Connection con = cf.getConnection();
// create and setup the specification
InteractionSpecImpl spec = new InteractionSpecImpl() ;
spec.setFunctionName( "TESTTX" ) ;
// set up the input message as a record
MyRecord in = new MyRecord() ;
in.setText( "MyData" );
// create the output record to hold the reply
MyRecord out = new MyRecord() ;
// create and execute the interaction
Interaction ix = con.createInteraction() ;
ix.execute ( spec, in, out );
// extract the reply from the output record
System.out.println( "received: " + out.getText() ) ;
```

Send-only transaction call

The COMPORSYS Connector for IMS supports the IMS SEND ONLY protocol for IMS transactions without any reply message.

The SEND ONLY protocol requires the use of COMMIT_MODE_0.

```
InteractionSpecImpl spec = new InteractionSpecImpl() ;
spec.setFunctionName( "TESTTX" ) ;
spec.setInteractionVerb(InteractionSpecImpl.SYNC_SEND);
spec.setCommitMode(InteractionSpecImpl.COMMIT_MODE_0);
// create and execute the interaction
Interaction ix = con.createInteraction() ;
ix.execute ( spec, in, out );
```

Asynchronous transaction call

The COMPORSYS Connector for IMS supports the IMS RESUME TPIPE operation to receive asynchronous output from IMS.

The Connector supports the RESUME TPIPE options SINGLE and SINGLE_WAIT. Specify the desired option as InteractionVerb on the request object

```
InteractionSpecImpl specAsync = new InteractionSpecImpl() ;
specAsync.setTimer(InteractionSpecImpl.TIMER_DEFAULT );
specAsync.setFunctionName( "" ) ;
specAsync.setClientId( "TPIPE" );
specAsync.setInteractionVerb(InteractionSpecImpl.
    SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_MSG_WAIT);
TextRecord in = new TextRecord("cp273");
// use an empty InputRecord
in.setText("");
TextRecord out = new TextRecord("cp273");
ix.execute(specAsync, in, out);
```

The timer value specifies how long IMS waits for an message to arrive in the TPIPE.

Use the SINGLE option to return immediately if no message is available in the TPIPE.

```
InteractionSpecImpl specAsync = new InteractionSpecImpl() ;
specAsync.setFunctionName( "" ) ;
specAsync.setClientId( "TPIPE" );
specAsync.setInteractionVerb(InteractionSpecImpl.
    SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_MSG;
.
.
.
ix.execute(specAsync, in, out);
```

Properties of the InteractionSpec bean

The InteractionSpec bean offers a set of properties. They allow you to configure the communication for your special needs.

Property	Meaning
FunctionName	Provide the name IMS transaction to call
InteractionVerb	Specifies the communication model. SYNC_SEND_RECEIVE (default): Standard send-receive flow. SYNC_SEND: Send-only request. SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_MSG: RESUME TPIPE operation with IMS SINGLE option. SYNC_RECEIVE_ASYNCOUTPUT_SINGLE_MSG_WAIT: RESUME TPIPE operation with IMS SWAIT option.
CommitMode	Specify either COMMIT_MODE_1 (default) or COMMIT_MODE_0

ExecutionTimeout	Sets the milliseconds to wait for a reply (TCP/IP timeout).
Timer	Sets the time the IMS Connect timer value. Use it in conjunction with RECEIVE_ASYNCOUTPUT.
MultisegmentSupport	Set to <code>true</code> , if the transaction has a multisegment request or reply.
Lterm	Sets the name of the LTERM.
ClientId	The name of client. If not provided, the connector generates an unique id.
MaxSegmentSize	Comma separated list of all segment sizes. The last value is used for all following segments. The first value gives the size of the first segment, the <i>n</i> th value give the size if the <i>n</i> th segment. All segment sizes must be given including the LLZZ length (4 bytes)

Table 8: Properties of an IMS InteractionSpec

Retrieves internal UCI properties through CCI

The reply record instance offers some useful properties to internal information usually only available through the UCI.

```
Connection con = cf.getConnection();
// create and setup the specification
InteractionSpecImpl spec = new InteractionSpecImpl() ;
MyRecord out = new MyRecord() ;
// create and execute the interaction
Interaction ix = con.createInteraction() ;
ix.execute ( spec, in, out );
// get the clientid of the request.
String clientId =
    out.getProperty(InteractionSpecImpl.PROP_CLIENT_ID) ;
// get the internal UCI reply object.
Reply reply = out.getProperty(InteractionSpecImpl.PROP_IMSREPLY) ;
```

Further information

- JAVADOC of the COMPORSYS Connector for IBM IMS
[%IMS_CONNECTOR%/doc/javadoc/index.html](#)
- J2EE™ Java Connector Architecture
<http://www.javasoft.com/j2ee/connector>
- IBM IMS Connect
<http://www.ibm.com>
- BEA WebLogic Server
<http://edocs.bea.com>
- Support for installation and deployment of the COMPORSYS Connector
<mailto://support@comporsys.de>