IMS Version 10 Connectivity Enhancements and the IMS Connect API

Kevin Flanigan
kf@us.ibm.com

Dave Cameron
daveac@ca.ibm.com

Jack Yuan
jackyuan@us.ibm.com

Information Management software

Agenda

- V10 IMS Connect API
- V10 IMS Connect Updates
- V10 Synchronous Callout via IMS Connect


IMS Version 10
V10 IMS Connect API
Kevin Flanigan
kf@us.ibm.com
Information Management software
© 2009 IBM Corporation

Agenda

- IMS Connect API Overview
  - Business Value
  - Description and Positioning
  - Prerequisites
- Design Assumptions
- Restrictions
- Externals
- Interfaces
- API Environment
- Properties files
- Tracing
- Usage
- IMS Connect Functions Supported
- IMS Connect API Functions

4

Business Value

- Target Market
  - The IMS Connect API is intended for all customers who write client applications that interact directly with IMS Connect (i.e., IMS Connect clients that do not use IMS TM Resource Adapter or IMS SOAP Gateway)
- Challenge Addressed
  - Simplifies interactions with IMS Connect by handling:
    - IMS Connect message header
    - IMS Connect interaction protocols
    - TCP/IP socket connections
- Solution Statement
  - Provides a set of reusable profiles that define connections and types of interactions to be performed and set of high- and low-level methods for performing interactions with IMS Connect
- Business Value Statement
  - Simplifies design, development, test and maintenance of IMS Connect client applications written in Java and C

5

This line item addresses the need for a simplified way to interact with IMS Connect. Currently, IMS Connect client application developers must understand the complexities of both the IMS Connect IRM header and IMS Connect protocols for all of the types of interactions supported by IMS Connect as well as the complexities of TCP/IP socket programming. The goal of the IMS Connect API is to shield IMS Connect client application developers from these complexities by providing an easy-to-use API which facilitates interacting with IMS Connect. This will be accomplished by providing simple ways of describing the connections that they need and the interactions that they want performed along with easy ways provide the data that they want sent to IMS for those interactions. The will provide re-usable profiles to configure connections and interactions and simple methods to perform those interactions. In addition, if IMS Connect client application developers want more direct control over certain interactions, they will still be able to use the more granular, lower-level calls to interact with IMS Connect and IMS as they do now with IMS Connect RYO (Roll-Your-Own) clients.

The IMS Connect API will establish socket connections between the client application using the API and IMS Connect based on a connection profile supplied either through customizable default connection property values or connection property values set at runtime using the Connection setter methods. The IMS Connect API can then perform interactions with IMS Connect on behalf of the client application after the TmInteraction has been configured based on the interaction properties specified by the client application in the same manner as the Connection properties are specified.

Versions of the API will initially be provided for both Java and C. Please note that, although the C implementation is mentioned occasionally in this presentation, the focus of the presentation is the Java implementation which is planned to be made available to customers prior to the C implementation.

Description and Positioning

- Facilitates remote access to IMS transactions and commands
- Shields users from:
  - IMS Connect protocol
  - Socket communications
- Targets users who need to write and maintain RYO applications
  - Accomplishes same task as IMS TM Resource Adapter and IMS SOAP Gateway
  - Leaner implementation
    - No J2EE/JCA
    - No SOAP server
  - Allows different RYO applications to use the same code for accessing IMS from Java applications
- Utilizes JCA-like paradigm for accessing IMS Connect
  - ConnectionFactory, Connection and TmInteraction interfaces

6

The IMS Connect API is a set of Java classes whose methods can be used to perform interactions with IMS through IMS Connect. The user “tells” the API what to do by setting values for the API properties which describe the connection to be made and the interaction to be performed.

The API can be used to execute both IMS transactions as well as IMS Connect and OTMA-supported IMS commands, while shielding the clients from having to manage the IMS Connect protocol or deal with the TCP/IP socket connections. The API targets users who need the same access to IMS that they can get through the IMS TM Resource Adapter or IMS SOAP Gateway, but who don’t need the overhead of a J2EE/JCA implementation or SOAP server and SOAP messaging environment – those who are looking for a “leaner” implementation. It allows all of your Java RYO client applications to share the same code for accessing IMS transactions and commands.

Although it is not a JCA connector, it uses a JCA-like paradigm for interactions. Your application instantiates a ConnectionFactory from which you obtain a Connection interface which the API communicates with IMS Connect, and from that Connection interface, you create a TmInteraction interface through which the interaction is performed.

Prerequisites

- Software requirements
  - IMS V10 license required
  - IMS V10 Integrated IMS Connect function
  - JDK/JRE 1.4.2 or higher
- Hardware requirements
  - Same as existing IMS Connect client application environment
- Tooling
  - None required

7

The IMS Connect API is being delivered as an IMS V10 SPE and, as such, requires a valid IMS V10 license. It will initially be supported only with IMS V10 and the Integrated IMS Connect V10 function. A follow-on version will be released in conjunction with IMS V11 which supports the new IMS V11 functions as well as additional IMS V10 SPE functions. JDK/JRE 1.4.2 or higher is required for developing and running IMS Connect API client applications. Tooling support is not required in the initial release but may be considered for a follow-on release.

The IMS Connect API does not impose any additional hardware requirements beyond those that would be needed to execute the same interactions without the use of the API. In this release there is no requirement for tooling to develop client applications that use the API.

Design Assumptions

- IMS Connect API
  - API will support existing IMS Connect function for RYO client applications
    - TCP/IP only
  - Client application responsible for:
    - Preparing input data as expected by IMS application
    - Interpreting output data as returned by IMS application
    - Management of memory for required data structures used in C implementation
  - Java implementation initially supported on Windows and System z
  - C implementation initially supported on Windows only

8

In order to make the IMS Connect API as flexible as possible, only the above-listed design assumptions were made. These design assumptions are in keeping with the way RYO applications are typically programmed today. No design assumptions were made which would restrict or otherwise impact its use. The TCP/IP-only nature of the API reflects the functionality currently available in IMS Connect to Roll-Your-Own client applications. Also, preparing the input request data in byte array form and interpreting the output response data are currently responsibilities of the RYO application and would continue to be so with the API. However, with the API you will be able to provide the input request data to the API and retrieve the output response data from the API in byte[], byte[][], String or String[] form.

The Java version of the API will initially be supported for use on Windows and System z only. This is a reflection of where the API is being tested. Being Java, it should run on any platform where there is a supported JRE/JVM that has TCP/IP connectivity to the target IMS Connect. The C version will be supported on Windows only.

Restrictions

- IMS Connect API intended to support existing IMS Connect function
  - New features added after start of development not included in initial GA
- Two-Phase Commit (2PC)
  - Not supported in initial release
  - May be restrictions which would prevent 2PC from ever being feasible with API
- SSL
  - May not be supported in initial release
  - May not be concern if running API on System z behind firewall
- Cancel client ID
  - May not supported in initial release
- XML
  - Not supported in initial release

9

The IMS Connect API is intended to support all existing IMS Connect function at the time of release of the API. However, in the first release, the 2PC functionality currently available in IMS Connect will not be supported. In fact, it could be that there are limitations to what the API can do which prevent 2PC from ever being feasible with the API. In addition, it is possible that SSL and cancel client ID support may also not be available in the initial Java release. XML support will not be available in the first release.

It should be noted that Local Option is IMS TM Resource Adapter-specific and not supported for IMS Connect RYO clients. As a result, it is also not supported by the IMS Connect API.

Externals

- Major Functionality from Users' Point-of-View
  - IMS Connect API executes interactions with IMS Connect
    - Shields IMS Connect client application developers from dealing with complexities of:
      - IMS Connect headers and protocols,
      - TCP/IP sockets programming, and
      - IMS Connect user exits (defaults to HWSSMPL1)
  - The IMS Connect client application developer responsible for configuring connection and interaction property files used by API
  - Optional features
    - SSL connections
    - API runtime tracing

10

The IMS Connect API is intended to be used to execute interactions with IMS Connect. In order to shield client applications (and their developers) from the complexities of interacting with TCP/IP sockets and IMS Connect, the IMS Connect API will internally both generate the IMS Connect input message header and manage the interaction with IMS Connect according to the IMS Connect message protocols. In addition, the IMS Connect API will deal with socket connections to IMS Connect made on behalf of the client applications.

The API will support all existing IMS Connect user message exits. The default user exit is HWSSMPL1. Customer written user exits will also be supported provided they use the same message and header structures used by the HWSSMPL0 or HWSSMPL1 user exits. HWSSMPL0 and HWSSMPL1 are identical except that response messages processed by HWSSMPL1 begin with a 4-byte length field, LLLL in front of the IRM, whereas messages processed HWSSMPL0 begin with the IRM fields without an LLLL prefix.

IMS Connect client application developers are responsible for configuring the connection and interaction properties files used by the API during execution or for coding the client applications to set appropriate values for these properties during execution.

Client applications will be able to elect to use SSL connections for secure, encrypted TCP/IP communications between the IMS Connect API and IMS Connect. The API will also allow the user to specify several different levels of tracing to be used at runtime. With tracing enabled and configured, runtime execution tracing is provided through the java.util.logging classes which generates tracing output similar to what is provided by WebSphere Application Server on distributed platforms.

Interfaces

- Customers may provide pathnames to customized files
  - Specify default connection- and interaction-related properties
- OO approach used for Java API
  - Similar to design used for J2EE and JCA
  - ConnectionFactory getConnection call returns Connection interface on which createInteraction is called to return TmInteraction interface
- Low-level calls for Java Connection interface
  - **Connect ( )**
    - Input parameters: Java – none
      C – pointer to ConnectionAttributes
      – pointer to call completion/error status structure
    - Return value: void
    - Opens TCP/IP socket connection to IMS Connect identified in properties of Connection instance
    - Invoked explicitly by client application or called internally by API during high-level execute call if Connection is not connected

11

A client application may provide pathnames to customized files which specify default values for connection- and interaction-related properties to be used Connection and TmInteraction instances used by an application. Different ConnectionFactory, Connection and TmInteraction instances may use separate customized files which specify the connection- and interaction-related property values to be used by that instance.

The IMS Connect API will allow client applications to interact with IMS Connect at either a high level, in which an interaction is configured and executed under complete control of the API, or at a lower level through which the client is able to exert more control over the interaction such as invoking individual calls to open a connection to the target IMS Connect, send a request message and receive a response message over that connection and finally disconnect that connection. However, although the user can exert this lower level control over the interaction that gets performed, the preferred means of performing interactions with IMS Connect will be to use the high-level execute interaction. The actual implementations of these calls will usually differ slightly between the Java and C implementations due to the different natures of the two languages, Object Oriented vs. procedural.

While the low-level calls give the application greater control over the interaction, they also require a deeper understanding of the IMS Connect protocols. The connect() call is used to open a physical TCP/IP socket connection to the target IMS Connect and associate it with the Connection object on which the Connect call is made.

The Java low-level connect() call takes no input parameters and has a return value of void. After the connect call completes successfully, the Connection object can then be used to perform interactions with the IMS Connect at the other end of the underlying socket. For C, the connect() call takes a parameter containing a pointer to the connection properties of the desired connection (for values not set by the application, default values for the connection properties will be used,) and a parameter containing a pointer to a status structure which contains the call completion code and any error message returned from the API or IMS Connect. Since the output is returned in the status structure, the C connect() call does not have a return value.

Interfaces (continued)

- Low-level calls for Connection interface (cont.)
  - **disconnect( )**
    - Input parameters: Java – none
      C – pointer to ConnectionAttributes structure
      – pointer to call completion/error structure
    - Return value: void
    - Closes underlying TCP/IP socket connection to IMS Connect for that Connection instance
    - Invoked explicitly by client application or called internally by API during high-level execute call if Connection is not connected
  - **send( )**
    - Input parameters: Java – input byte array
      C – pointer to ConnectionAttributes structure
      – pointer to input byte array
      – pointer to call completion/error structure
    - Return value: void
    - Sends input byte array to IMS Connect using underlying socket for that Connection instance
    - Invoked explicitly by client application or called internally by API during high-level execute call

12

The low-level disconnect() call can be invoked by the client application when an application has finished using a particular socket connection. The Java disconnect method has no input parameters and no return value. The C disconnect() call takes a pointer to a ConnectionAttributes structure along with a pointer to a call completion/error structure as input parameters and has no return value.

The low-level send() call is invoked by the API when an application wants to send a request message to IMS Connect. In Java, the send() call takes one input parameter: a one-dimensional input data byte array containing LLLL, IRM, LLZZ, trancode (if applicable), data and the End-Of-Message indicator, an empty segment consisting of just LLZZ. Multi-segment messages are constructed in a similar manner with as many segments as needed placed between the first data segment and the EOM segment and each segment delineated by an LLZZ prefix. The Java low-level send() call does not have a return value. In C, the send() call takes three parameters, one containing a pointer to a ConnectionAttributes structure, another containing a pointer to the input data byte array and a third containing a pointer to the call completion codes and any error message returned from the API, IMS or IMS Connect. The C send() call also does not have a return value. Since the low-level send call only sends a message to IMS Connect, any response must be retrieved using a separate receive call or resumeTpipe interaction if the response was treated as asynchronous output by IMS Connect and routed to an asynchronous hold queue.

Interfaces (continued)

- Low-level calls for Connection interface (cont.)
  - **receive( )**
    - Input parameters: Java – void
      C – pointer to ConnectionAttributes structure
      – pointer to output buffer
      – pointer to call completion/error structure
    - Return value: Java – output byte array
      C – void
    - Receives output byte array from IMS Connect using underlying socket for that Connection instance
    - Invoked explicitly by client application or called internally by API during high-level execute call

13

The low-level receive() call is invoked by the API when an application wants to receive a response message from the IMS application (or in some cases, the response from the IMS control region, IMS Connect or an IMS Connect user exit.) The receive() call is typically used after a send() call when the send and receive are invoked separately (as opposed to invoking a sendReceive interaction through the execute call described later in this presentation) or following a resumeTpipe request sent in using a low-level send() call. In Java, the receive() call takes no parameters and returns a one--dimensional output byte array containing either a single-segment response message or a multi-segment response message, provided the segments are delineated by LLZZ prefixes. In C, the receive() call takes three parameters, one containing a pointer to a ConnectionAttributes structure, another containing a pointer to the output data byte array and a third containing a pointer to the status structure which holds the call completion codes and any error message returned from the API, IMS or IMS Connect. The C receive() call does not have a return value since the response message is returned in the output buffer pointed to in the parameter list of the receive() call.

Interfaces (continued)

- High-level calls for Connection interface (cont.)
  - **loadConnectionPropertiesFromFile( )**
    - Input parameters: Java – qualified or unqualified name of file containing Connection property values
      C – qualified or unqualified name of file containing Connection property values
      – pointer to ConnectionAttributes structure
      – pointer to call completion/error structure
    - Return value: void
    - Sets the values for all Connection properties identified by name-value pairs in the specified Connection properties file
    - Invoked explicitly by client application

14

There is also a higher level loadConnectionPropertiesFromFile() call in the Connection interface. This call allows the user to set the values for some or all of the Connection properties from a text file containing name-value pairs of different Connection properties. The loadConnectionPropertiesFromFile() call would typically be invoked by the API when an application wants to change the values for a number of different Connection properties from their default values to the values specified in the name-value pairs in the In Java, the loadConnectionPropertiesFromFile() call takes a String parameter containing the qualified or unqualified name of file containing Connection property name-value pairs and has no return value. In C, the loadConnectionPropertiesFromFile() call takes three parameters: a string parameter containing the qualified or unqualified name of file containing Connection property name-value pairs, a pointer to a ConnectionAttributes structure and a third parameter containing a pointer to the status structure which holds the call completion codes and any error message returned from the API, IMS or IMS Connect. The C loadConnectionPropertiesFromFile() call has no return value.

Interfaces (continued)

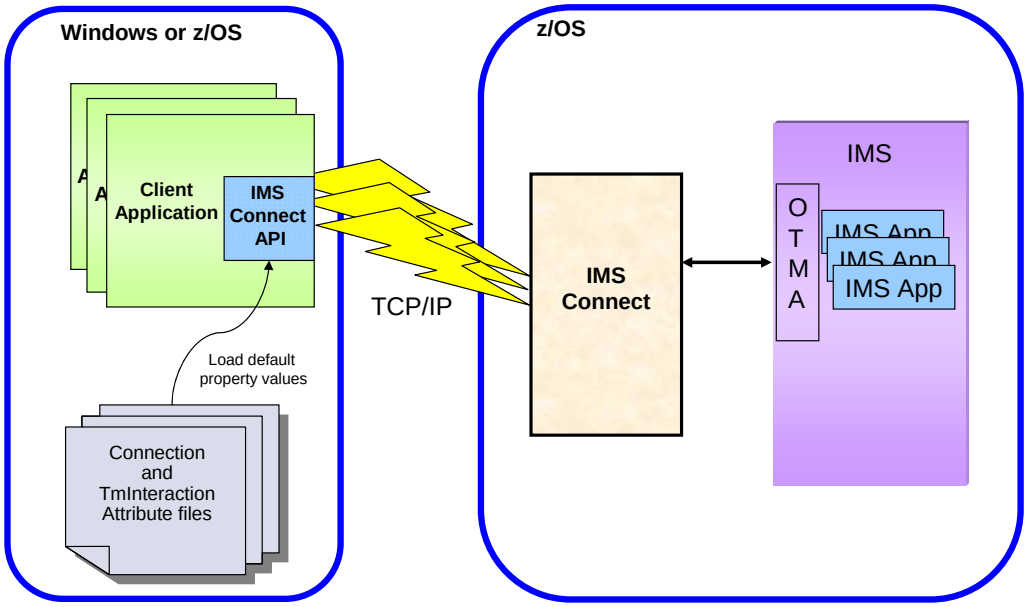
- High-level calls for TmInteraction interface
  - **execute( )**
    - Input parameters: Java – none
      C – pointer to ConnectionAttributes structure
      – pointer to input buffer
      – pointer to output buffer
      – pointer to InteractionAttributes structure
      – pointer to call completion/error structure
    - Return value: void
    -
    - Invoked explicitly by client application
      - Establishes socket connection with IMS Connect if not already connected
      - Builds input request message to be sent to IMS Connect including the IRM headers and with or without trancode and data as necessary
      - Enforces rules of IMS Connect protocol, throwing exceptions if interaction would violate protocol
      - Sends request message to IMS Connect
      - Receives output from IMS Connect and IMS
      - Processes output and returns either output or exception to calling client application

15

The most method for most users will be the execute() call in the TmInteraction interface. This call allows the user to perform an interaction with IMS Connect and IMS utilizing the connection and interaction configuration that has previously been set up. The execute() call is provides users with a single method call that performs the entire interaction, relieving the application of the burden of managing most of the communications protocol with IMS Connect. In Java, the execute() call takes no parameters and return value. The input has already been set by the client application in the InputMessage interface and the output response can be retrieved from the OutputMessage interface in a variety of ways after the execute() call has completed and control has been returned to the client application. In C, the execute() call takes five pointer parameters: a pointer to the ConnectionAttributes structure, another to the input buffer, the third to the output buffer, the fourth an InteractionAttributes structure and the fifth pointer to the status structure which holds the call completion codes and any error message returned from the API, IMS or IMS Connect. Because the response message and other properties of the interaction are returned in structures in the method parameters, the C execute() call has no return value.

The execute method performs all of the setup for an interaction, completes the interaction, then interprets the response message to determine if the interaction successfully returned a response or returned an error condition and finally returns either the response or an appropriate exception as needed. The steps required to accomplish this begin with determining if underlying socket connection of the Connection instance to be used is connected to the target IMS Connect. If the socket is not connected, the execute method first executes the low-level connect() method on the Connection to open a connection to IMS Connect. Once the connection has been opened, the execute() method continues by analyzing the TmInteraction property values to determine if the interaction that it is being asked to perform is valid and then performing the requested interaction if it conforms to the IMS Connect protocol. If the interaction is not valid, the execute() method builds an appropriate exception and then returns that exception to the calling client application. The client application then gets control and is able to process the output message or exception and proceed accordingly.

API Environment



16

This figure shows the environment in which the IMS Connect API can be used. It also depicts the fact that multiple client applications that use the IMS Connect API can be invoked simultaneously. The API will communicate with IMS Connect. Upon request by the client application, for example, in an execute() or connect() call, the API will create a connection object for use by that application only, which the client must keep track of, calling disconnect() on the connection object before exiting. Of coarse in Java, orphaned connections would eventually be cleaned whenever there are no longer any references to those connection objects, but this still requires that the application developer does not keep references to connection objects that it no longer needs.

In addition to IMS transactions, the initial release of the IMS Connect API will support the IMS Connect user message exit-supported PING and RACF password change commands along with all IMS commands supported by OTMA.

Properties Files

- Default values
  - Properties files used to set default values for Connection and TmInteraction
  - Default values can be over-written by Connection and TmInteraction properties setter methods
- Two types of properties files supported
  - Connection properties
    - Identify target IMS Connect, type of connection and SSL-related properties of Connections
    - Same connection properties used by ConnectionFactory class and Connection Interface
      - ConnectionFactory can be loaded with connection property values which are then used as default values for Connections it creates
      - Connections can also be loaded with connection property values which overwrite values set by ConnectionFactory
    - Changing of connection properties allowed only while Connection is not connected
  - TmInteraction properties
    - Describe type and characteristics of interaction to be performed
    - Can be changed any time – changes affect subsequent interactions

17

Properties files, plain text files containing name-value pairs, are supported by the IMS Connect API for the ConnectionFactory class and the Connection and TmInteraction interfaces. The intent of these text files is to allow users to customize these files for use with either individual applications or groups of applications. All of the properties are included in these files, but the values set in the files as shipped are the API's default values for these properties. Because they are read in at runtime, it is recommended that you remove the settings for any properties in these files where application(s) will use the API's default value. To support the use of these properties files by multiple applications, you can configure the files to set only those properties whose values are the same in multiple applications and are not the API's default values. Values which differ from application to application or are changed during the running of an application can be set using the properties setter methods in order to make the applications faster and more efficient.

Although there are three different Java classes and interfaces that use properties files, there are only two different types of properties files, Connection and TmInteraction. This is because the same properties can be specified for the ConnectionFactory class and Connection interface. The two types of properties files differ only in the properties that can be specified in them. They can be given any names that OS can use to locate the files, but they must be plain text files. Java-style (//) or C-style comments can be used in the files, but it is recommended that they be removed from the copies used at runtime for better performance. If an invalid property name or value is encountered while reading in one of these files, loading will terminate immediately and an exception will be returned to the user.

The ConnectionFactory and Connection properties are the same to allow users to configure connections at either the higher-level ConnectionFactory or in the Connection itself. This allows you to configure a base set of properties once at the ConnectionFactory level, and then configure other properties at the Connection level on a Connection-by-Connection basis. The Connection properties are used to identify the target IMS Connect, hostname and portnumber, the type of connection, persistent or transaction and SSL-related properties such as the whether you want to use SSL for that connection, the type of encryption you want to use, and the locations and passwords for a keystore and truststore and the passwords. Note that these properties can only be set while there is not an open physical TCP/IP socket associated with the Connection. Otherwise, an exception will be returned to the client application without the API having changed the property's value or setting any of the remaining values in the properties file.

TmInteraction properties are used to describe the type of interaction that you want performed and its characteristics. When you are using the high-level execute method to perform your interactions, in general, the TmInteraction property values can be changed at any time in your application. This is because their values are only used while the execute and certain other setter methods are running. The API has control while these methods are running and therefore the application is not able to make any property value changes after it has passed control to the API until the API returns control to the calling application. The one caveat to this statement is that you would not want to set the data in your InputMessage object and then change a property such as the imsConnectCodePage which affects how the data is saved in the InputMessage object. The correct approach would be to set the values of all of the environment-related properties first, before you invoke methods which utilize those values in there processing.

Properties Files (continued)

Connection Properties

Target IMS Connect
- Host name
- Port number

SSL configuration
- Use SSL flag
- SSL encryption level
- Keystore and truststore

Other properties
- Client ID
- Socket type

TmInteraction Properties

Interaction
- Datastore name (IMS)
- Commit mode, sync level
- Interaction type

Message routing
- Purge undeliverable output
- Reroute undeliverable output
- Reroute name

Security
- RACF settings

Timeouts
- IMS Connect timeout
- Interaction timeout

Message encoding
- IMS Connect codepage

Response
- responseIncludesLLLL
- returnMfsModName

18

This slide lists the categories of properties and some examples of the Connection properties and the TmInteraction properties.

As mentioned previously, the Connection properties identify the TCP/IP endpoint, which is the listening port of the target IMS Connect, by hostName and portNumber properties. The SSL configuration includes properties which tell the API whether or not to use SSL for that Connection, what level of encryption you want used for data passed back and forth on the underlying SSL connection along with the name of the keystore and truststore and their passwords. In addition, there are other properties which allow you to set the clientId and socket type for a Connection. The clientId when assigned a non-null, non-blank value, will be used by IMS Connect to identify that connection. If the clientId is set to blanks, the IMS Connect user message exit will create a unique clientId to be used for that connection. The socket type determines how IMS Connect will treat a connection when an interaction has completed. Persistent sockets remain open until they are explicitly closed by the client application or the API or until a fatal error has occurred on the connection and IMS Connect needs to close the connection because it can no longer guarantee that its state is valid. Transaction sockets on the other hand are automatically closed by IMS Connect at the conclusion of an interaction. An interaction does not conclude until any required ACK or NAK has been received or, if it is a conversational interaction, until all of the iterations of the conversation have completed and the conversation has been ended by IMS. Conversations can be ended by IMS either in response to an end conversation request sent by the client through the API or by the IMS application that was processing the conversation.

The TmInteraction properties can be divided into several different categories. The above charts list these categories and some example properties in these categories First there are interaction-related categories which contain properties that control various facets of an interaction with IMS Connect. These would include the datastore name which tells IMS Connect what IMS you want to interact with, the commitMode and syncLevel which determine the process that IMS goes through to complete transaction and commit any database changes along with the interactionType which determines the type of interaction to be performed such as sendrecv, the default, sendOnly, resumeTpipe, Ack or Nak. Message routing properties such as the rerouteUndeliverableOutput and purgeUndeliverableOutput flags and rerouteName determine what OTMA and IMS Connect do with output that cannot be delivered to the client application. Security-related properties sucn as racfUserId, racfPassword, racfGroupName and racfApNm provide the security information that IMS Connect uses for authentication of the client and IMS uses to verify client authorization to use RACF protected resources in IMS. Timeout properties used by the IMS Connect API include the imsConnectTimeout property which determines the amount of time that IMS Connect should wait for response to be returned by OTMA, and the interactionTimeout property which determines the amount of time that the API should spend waiting for a response to be received from IMS Connect before throwing a timeout exception back to the originating client application. MessageEncoding properties include the imsConnectCodePage which determines what codepage the API is to use when creating the input request message byte array and interpreting the output response message. Response-related properties include properties such as responseIncludesLLLL which tells the API whether the output response messages received from IMS Connect are prefixed with LLLL values as is the case with the default user message exit, HWSSMPL1, and the returnMfsModName property, which tells IMS whether or not to return an MFS MOD name in a separate segment in the output response message.

More information about these and other properties will be provided in a later slide in this presentation.

Tracing

- Uses java.util.logging package
- Provides four levels of tracing from no tracing to detailed tracing:
  - None
  - Exception – exceptions only

```
Mon, Jan 19 2009, 14.05.15.971 PST [SEVERE] Exception caught in client application
Mon, Jan 19 2009, 14.05.15.986 PST [SEVERE] Exception was:
[com.ibm.ims.connect.ImsConnectCommunicationException: HWS0006E: IMS OTMA returned an
error. IMS Connect return code: [16], OTMA sense code: [26], OTMA reason code: [25] (SMB
transaction or LTERM stopped (see DFS065))]
```

  - EntryExit – exceptions plus significant method entries and exits

```
Fri, Jan 16 2009, 17.18.05.111 PST [FINER] --> ConnectionImpl.connect()...
Fri, Jan 16 2009, 17.18.05.283 PST [FINER] <-- ConnectionImpl.connect()...
```

  - Internal – trace exceptions plus entries and exits plus useful information such as the input and output buffer contents formatted and interpreted

```
InputMessage.getBytes() - writing trancode [IVTNO     ] to "dout" DataOutStream
InputMessage.getBytes() - copied bout.toByteArray() to "message" byte array:
00000098 00700200 5ce2c1d4 d7d3f15c  00000000 10001000 c3d3c9c5 d5e3f0f1  |...q.ø..*SAMPL1*........CLIENT01| : 32
00400140 c9e5e3d5 d6404040 c9d4e2f1  40404040 40404040 40404040 d9c1c3c6  |. . IVTNO   IMS1            RACF| : 64
e4c9c440 d9c1c3c6 c7d9e4d7 d9c1c3c6  d7e2e6c4 d9c1c3c6 c1d7d5d4 40404040  |UID RACFGRUPRACFPSWDRACFAPNM    | : 96
40404040 c8e6e2e7 d4d3c1f0 c8e6e2e7  c3d5e5f0 00200000 c9e5e3d5 d6404040  |    HWSXMLA0HWSXCNV0....IVTNO   | : 128
4040c4c9 e2d7d3c1 e840d3c1 e2e3f140  40404040 00040000                    |  DISPLAY LAST1     ....        | : 152
```

19

The IMS Connect API uses the java.util.logging package to provide tracing services for the API. Four levels of tracing are provide, None in which tracing is done, Exception where only exception messages are traced, EntryExit in which tracing statements are logged on entry and exit of all methods where any significant processing occurs and Internal where additional information is traced such as the contents of the send and receive buffers.

Tracing

- Tracing output at Internal level also includes interpreted IRM fields

```
Thu, Jan 15 2009, 00.01.15.877 PST [FINEST]            llll = 0x98 (decimal 152)
Thu, Jan 15 2009, 00.01.15.877 PST [FINEST]          irm_ll = 0x70 (decimal 112)
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]         archLvl = 0x02
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]              f0 = 0x00 (decimal 0] (no XML transformation)
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]           irmId = [*SAMPL1*]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]          resWrd = 0x00 (decimal 0]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]              f5 = 0x10 (decimal 16]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]                    EBCDIC translation not done by client
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]                    resumeTpipe option is SINGLE_WAIT
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]        irmTimer = 0x00 (decimal 0]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]        sockType = 0x10 (decimal 16]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]                    Persistent socket
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]       encSchema = 0x00 (decimal 0]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]        clientId = [CLIENT01]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]              f1 = 0x00 (decimal 0)
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]                    Return MFS modname not requested
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]              f2 = 0x40 (decimal 64)
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]                    Commit mode 0
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]              f3 = 0x01 (decimal 1)
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]                    Sync Level is CONFIRM (1),
                                                               Purge undeliverable output is false,
                                                               Reroute undeliverable output is false
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]              f4 = 0x40 (decimal 64, chararcter [ ])
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]                    SENDRECV interaction
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]     irmTrancode = [IVTNO   ]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]     datastoreId = [IMS1    ]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]       ltermOvrd = [        ]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]      racfuserid = [RACFUID ]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]       racfGroup = [RACFGRUP]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]    racfPassword = [********]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]      racfApplNm = [RACFAPNM]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]     rerouteName = [        ]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]    xmlAdapterNm = [HWSXMLA0]
Thu, Jan 15 2009, 00.01.15.892 PST [FINEST]    xmlConvrtrNm = [HWSXCNV0]
```

20

In addition to the contents of the buffer sent to IMS Connect, the trace output at the Internal level includes a listing of the contents of each field in the IRM header along with the meanings of those fields.

Usage

- API uses JEE/JCA-like paradigm
  - Instantiate ConnectionFactory
    - Configure using loadConnectionFactoryPropertiesFromFile() method or ConnectionFactory property setter methods
  - Create Connection from ConnectionFactory
    - Configure using loadConnectionPropertiesFromFile() method or Connection property setter methods
  - Create TmInteraction from Connection
    - Configure interaction using loadTmInteractionPropertiesFromFile() method or TmInteraction property setter methods
  - Create input data as one- or two-dimensional byte array or String
  - Obtain InputMessage object using TmInteraction.getInputMessage()
    - Set input data in InputMessage
  - Invoke TmInteraction.execute() to perform interaction
  - Obtain OutputMessage using TmInteraction.getOutputMessage()
    - Get response data and properties from OutputMessage

21

From an external standpoint, the structure of the API bears a resemblance to the structure of the J2EE Connector Architecture. The basic way that you use the API is to instantiate a ConnectionFactory and from that, create Connections which are the interfaces through which the client applications accesses either directly or indirectly the underlying physical connection to IMS Connect. Both the ConnectionFactory as well as the Connection interfaces that are created by that ConnectionFactory can be configured, that is, the connection properties which define the partner endpoint of a Connection and some of the characteristics of that endpoint, can be set. This setting of properties can be accomplished either using setter properties on individual properties or by loading a properties file containing a name-value pair for each property that you want to set. Allowing properties to be set both at the connection factory level, allows you to set customized values at the ConnectionFactory level

for a base set of properties that will usually or always be the same in all of the Connection instances that are created from that ConnectionFactory. Then the individual Connection objects created can be further customized as required, again using the Connection setter methods or by loading a properties file containing a name-value pair for each property that you want to customize.

From the Connection, you can create a TmInteraction interface which is the interface through which you set properties that describe the interaction that you want to perform. The TmInteraction object can be also be configured by loading a properties file containing a name-value pair for each property that you want to set or by setting the property values you want to configure individually. As with the Connection interface, you can use a combination of these two approaches to set the majority of the properties, those that are common in most TmInteraction interfaces used by your application, with the properties file loader approach and use the TmInteraction property setters to customize the TmInteraction interfaces for the particular interaction they are to perform.

The TmInteraction object contains references to an InputMessage interface to which you provide the input data for the interaction in a variety of ways, as well as an OutputMessage interface which the application will use after the interaction completes to retrieve the interaction's response. The IMS Connect return code as well as the OTMA sense code and reason code are available through getter methods in both the TmInteraction and OutputMessage interfaces. The output response data must be retrieved from the OutputMessage, but can be retrieved in a variety of byte array and String formats.

Usage *(continued)*

- Simplified example

```
 1 Logger logger = apiLoggingConfig.configureApiLogging(“trace.out”, ApiProperties.TRACE_LEVEL_INTERNAL);
 2
 3 ConnectionFactory myCF = new ConnectionFactory();
 4 myCF.loadConnectionFactoryAttributesFromFile(“c:\\myPropAttributesFiles\\ConnAttr.txt");
 5 myCF.setHostName(“myhost.mycompany.com");
 6 myCF.setPortNumber(9999);
 7
 8 Connection myConn = myCF.getConnection();
 9 myConn.setClientId(“client01");
10
11 TmInteraction myTmInteraction = myConn.createInteraction();
12 myTmInteraction.setImsDatastoreName("IMS1");
12 String myImsConnectCodepage = myTmInteraction.getImsConnectCodepage();
13 byte[] indata1 = (new String("/STA TRAN IVTNO")).getBytes(myImsConnectCodepage);
14 InputMessage inMsg = myTmInteraction.getInputMessage();
15 inMsg.setInputMessageData(indata1);
16
17 myTmInteraction.execute();
18
19 OutputMessage outMsg = myTmInteraction.getOutputMessage();
20 System.out.println(“\nResponse was: [” + outMsg.getDataAsString + “]\n”);
21
22 myTmInteraction.setTrancode(“IVTNO     “);
23
24 byte[] indata2 = (new String(“DISPLAY LAST1     ")).getBytes(imsConnectCodepage);
25 inMsg.setInputMessageData(indata2);
26
27 myTmInteraction.execute();
28
29 System.out.println(“\nResponse was: [” + outMsg.getDataAsString + “]\n”);
```

22

Here is a very simplified example. It describes a very simple scenario in which an IMS /START TRAN command is sent in followed by request to run an IVTNO DISPLAY LAST1 command. The IVTNO transaction is part of the IMS INSTALL/IVP installation verification program which is shipped with the IMS product and therefore be available in customer shops.

Some things that you should observe about this small sample program are:

- Connections are configured at both the ConnectionFactory and Connection level using both properties file loaders as well as individual property setter methods
- TmInteraction uses all default values except for the imsDatastoreName and the trancode so the property file loader is never called
- The trancode is right-padded with 5 blanks which makes the trancode longer than 8 bytes. This is done because the message format of IVTNO in the INSTALL/IVP requires a total of 10 characters between the start of the trancode, IVTNO and the start of the command field in the message, DISPLAY. Since the API has no knowledge of this requirement, the client application must specify the actual trancode field to be set in the message including any required trailing blanks.
- Note that this is a simplified sample program for demonstration purposes so important considerations such as error handling have not been included in this example.

IMS Connect Functions Supported

- IMS transactions
- OTMA-supported IMS commands
- IMS Connect PING command
- IMS Connect RACF password change command
- **Persistent** and transaction sockets
- IMS Connect-generated client IDs (application-generated – **CLIENTID**)
- SSL connections to IMS Connect (**non-SSL**)
- HWSSMPL0 and **HWSSMPL1** user message exits (responses with and without LLLL)
- IMS Connect DATASTORE definitions
- ResumeTpipe processing options **single wait** and nowait, auto and noauto
- ResumeTpipe alternate client ID
- LTERM override name (**8 blanks**)
- Return MFS MOD name with response (**false**)
- Commit mode's **commit-then-send** (CM0) and send-then-commit (CM1)
- Sync level's none and **confirm**
- Interaction types ack, nak, cancel timer, end conversation, receive, resume tpipe, send only, send only w/ ack, send only w/ XCF ordered delivery and **send receive**
- Purge undeliverable output (**false**)
- Reroute undeliverable output (**false**)
- Reroute name (**8 blanks**)
- RACF user ID (**RACFUID**)
- RACF group name (**RACFGRUP**)
- RACF password (**RACFPSWD**)
- RACF appl name (**RACAPNM**)
- IMS Connect timeout (**0** – use IMS Connect timeout value)
- Interaction timeout (**0** – wait forever)
- IMS Connect codepage (**cp037** – EBCDIC)
- ASCII or EBCDIC messages

- **Bold = default value**

This slide lists the IMS Connect functions supported by the IMS Connect API which generally correspond to the Connection properties and the TmInteraction properties.

The IMS Connect API supports both IMS TM transactions and those IMS Commands supported by OTMA. The IMS Information Center contains a section which lists the OTMA modified commands, IMS commands which were modified to support invocation through OTMA. The API also supports the IMS Command PING command which allows a client to submit a PING request and get a PING RESPONSE to verify the health of the target IMS Connect as well as the IMS Connect RACF password change command which allows a client to submit a request to IMS Connect to change the RACF password of the RACF user ID specified in the TmInteraction properties. A user simply submits an interaction with the data set to HWSPWCH oldpaswd/newpaswd/newpaswd. The EOM, that is, the End Of Message indicator is an "empty" segment, which is a segment that consists solely of LLZZ (00 04 00 00). The LLLL prefix, the LLZZ for the data and the EOM indicator are all added to the message as needed during processing in the execute() method.

The API supports two socket types. The socket type determines how IMS Connect will disconnect a connection when an interaction has completed. Persistent sockets remain open until they are explicitly closed by the client application or the API or until a fatal error has occurred on the connection and IMS Connect needs to close the connection because it can no longer guarantee that its state is valid. Transaction sockets on the other hand are automatically closed by IMS Connect at the conclusion of an interaction where an interaction does not conclude until the any required ACK or NAK has been received or, if it is a conversational interaction, until all of the iterations of the conversation have completed and the conversation has been ended by IMS. Conversations can be ended by IMS either in response to an end conversation request sent by the client through the API or by the IMS application that was processing the conversation.

Client IDs are used by IMS Connect to identify a connection. The clientId in the IMS Connect API can either be assigned a non-null, non-blank value, in which case that clientId value will be used by IMS Connect. If the clientId is set to blanks, the IMS Connect user message exit will create a unique clientId to be used for that connection referred to as an IMS Connect-generated client ID. When generated clientId's are used, the client application will never know what the clientId of that connection is since the clientId is not passed back to the client by IMS Connect. The use of generated clientId's is a way to ensure that a Connection has a unique clientId within a given IMS Connect since the clientId is generated by IMS Connect which can guarantee that the clientId is not a duplicate of another, already-existing clientId in that IMS Connect.

The SSL configuration includes properties which tell the API whether or not to use SSL for that Connection, what level of encryption you want used for data passed back and forth on the underlying SSL connection along with the name of the keystore and truststore and their passwords. The API handles the keystore and truststore in such a way that they can either be used to hold public and private certificates separately or together in the same keystore or truststore.

The API supports the HWSSMPL0 and HWSSMPL1 IMS Connect user message exits. The API will work with any message exit that uses the same IRM as is used by the HWSSMPL0 and HWSSMPL1 exits. You specify which exit to use by specifying the imsConnectUserMessageExitIdentifier, which must match the identifier of one of the exit you want to use in the target IMS Connect. The TmInteraction responseIncludesLLLL property can be used to tell the API whether the output response messages received from IMS Connect are prefixed with LLLL values as is the case with the default user message exit, HWSSMPL1. If the imsConnectUserMessageExitIdentifier is "*SAMPLE*", the API will process response messages as if they do not have an LLLL prefix. If the imsConnectUserMessageExitIdentifier is "*SAMPL1*", the API will process response messages as if they are prefixed with an LLLL value. If the imsConnectUserMessageExitIdentifier is anything other than "*SAMPLE*" or "*SAMPL1*", the API will use the responseIncludesLLLL property to determine whether to process response messages as if they are or are not prefixed with an LLLL value.

The datastoreName property in the TmInteraction interface tells the API and IMS Connect what IMS you want to interact with. The datastoreName property value must match the ID value on the appropriate DATASTORE statement of the target IMS Connect's configuration member.

The TmInteraction resumeTpipeProcessing property tells the API which type of resumeTpipe you want to use. The API supports SINGLE_WAIT, SINGLE_NOWAIT, AUTO and NOAUTO. The difference between SINGLE_WAIT and SINGLE_NOWAIT is that, if there is no output available to be returned when the resumeTpipe call is made, IMS Connect will return always return timeout for NOWAIT even if output is received by IMS Connect receives output before the IMS Connect timeout expires. On the other hand, if IMS Connect is processing a resumeTpipe WAIT, and IMS Connect receives output before the IMS Connect timeout expires, IMS Connect will return that new output response rather than timing out. AUTO and NOAUTO are processed in a similar fashion. The difference between SINGLE and AUTO is that resumeTpipe SINGLE's will return at most, one output response message. Auto on the other hand, will return as many messages as are available to be returned with the limitation that a resumeTpipe NOAUTO will only return those messages that are available at the time the resumeTpipe interaction request is received by IMS Connect for processing. The resumeTpipeAlternateClientId property of TmInteraction allows a client application to retreive the asynchronous output for another client ID. This function is typically used in conjunction with output that has been rerouted to a different clientID defined by a reroute name, described below.

The lTermOverrideName of the TmInteraction interface allows the client application to specify an name that OTMA places in the IOPCB LTERM field. The returnMfsModName property of the TmInteraction interface allows a client to request that IMS return an MFS MOD name in a separate segment in the output response message.

Other interaction-related properties include commitMode and syncLevel which determine the process that IMS goes through to complete transaction and commit any database changes. The commit mode specifies that either commit-then-send (frequently referred to as CM0) or send-then-commit (CM1) is to be used. The sync level determines whether response messages must be acknowledged (syncLevel CONFIRM) or not (syncLevel NONE.)

The TmInteraction interactionType property determines the type of interaction to be performed such as sendrecv, the default in which the API attempts to send an input request message to IMS Connect and receive an output response from IMS Connect, sendOnly in which the API attempts only to send an input request message to IMS Connect, resumeTpipe in which the API attempts retrieve asynchronous output, and Ack or Nak to acknowledge a previous output response message received from IMS Connect.

The TmInteraction message routing properties purgeUndeliverableOutput, rerouteUndeliverableOutput and rerouteName determine what OTMA and IMS Connect will do with output that cannot be delivered to the client application.

Security-related properties of the TmInteraction interface include racfUserId, racfPassword, racfGroupName and racfApplName which provide the security information that IMS Connect uses for authentication of the client and IMS uses for authorization to use RACF protected resources in IMS by the client.

TmInteraction timeout-related properties include the imsConnectTimeout property value which is used by IMS Connect to override its TIMEOUT value which in turn, determines the amount of time that IMS Connect will wait for a response to be returned by OTMA, and the interactionTimeout value which determines the amount of time that the IMS Connect will wait for an interaction response to be returned from IMS Connect.

Message encoding properties of the TmInteraction interface include the imsConnectCodePage property which determines what codepage the API will use when creating the input request message byte array and interpreting the output response message. Client applications can use the imsConnectCodePage to tell the API whether to use ASCII or EBCDIC for the input message encoding.

IMS Connect API Functions

- IMS Connect API-specific Functions
  - Client ACK or NAK vs. API internal ACK
- LLZZ and trancode can be provided by client application or API
- Input and output message data types
  - Byte array
    - Single-segment
    - Multi-segment if segment in data provided are delineated with LLZZ's
  - Array of byte arrays
    - Single-segment if only one byte array element is populated
    - Multi-segment – LLZZ not required to be specified
  - String - LLZZ cannot be specified
    - Single-segment only since there is no way to delineate segments within String
  - Array of Strings - LLZZ cannot be specified
    - Single-segment if only one String element is populated
    - Multi-segment

24

The IMS Connect API has some functions of its own that are worthy of mention. First, the API allows a client application to reply with an ACK or NAK when an acknowledgment to a response message is required or to defer that responsibility to the API. In this latter case, that is when the ackNakProvider property value has been set to API_INTERNAL_ACK, the API will send an ACK interaction back to IMS Connect under the covers during the execute() method processing before returning control back to the client application. Note that when the ackNakProvider property is set to API_INTERNAL_ACK, the API will only send in an ACK. The API does not have any criteria to decide whether the response is valid or invalid so there would never be a reason to NAK a message. On the other hand, if the ackNakProvider property is set to CLIENT_ACK_NAK, the client application can send in an ACK in which case the output message will be dequeued in IMS or a NAK in which case the output response message will remain queued in IMS and can be retrieved and dealt with at a later time as long as the API has not been configured to purge undeliverable output.

Input request messages sent to IMS Connect must include LLZZ and an optional trancode depending on the interaction. The LLZZ and trancode can either be provided by the client application within the data that it sets in the InputMessage interface or it can be left out of that data and the API will automatically add an LLZZ along with a trancode if appropriate to the input request message byte array that it sends to IMS Connect.

The input data can be provided to the InputMessage and retrieved from the OutputMessage interface in any of several different data types: as a one- or two- dimensional byte array or as a one- or two-dimensional String.

Setting input data in an InputMessage using a one-dimensional byte array:
- Use of both single- and multi-segment data is supported
- Single-segment message data can be provided with or without LLZZ.
- For multi-segment data, each segment must be delineated by an LLZZ and the trancode must be included after the LLZZ in the first segment if appropriate.

Setting input data in an InputMessage using two-dimensional array of byte arrays :
- Use of both single- and multi-segment data is supported
- Both single- and multi-segment message data can be provided with or without LLZZ and trancode if appropriate
- To set single-segment data in an array of byte arrays, only populate the first byte array element
- For multi-segment data, each segment is provided in a separate byte array element in the array of byte arrays

When providing input data as a String or a String array, the LLZZ, and therefore the trancode as well, cannot be specified by the client application since there is no way to delineate segments with binary LLZZ values within the String. As a result LLZZ and trancode must be added by the API during the execute method.

Setting input data in an InputMessage using a String:
- Use of only single-segment data is supported

Setting input data in an InputMessage using a String array:
- Use of both single- and multi-segment data is supported
- To set single-segment data in a String array, only populate the first element of the String array
- For multi-segment data, each segment is provided in a separate String element in the String array

In a similar fashion, a number of different methods are provided to retrieve the response data from an OutputMessage. Response data can be retrieved from the OutputMessage using the following getter methods:
- getDataAsByteArray()
- getDataAsArrayOfByteArrays()
- getDataAsString()
- getDataAsArrayOfStrings()

In each of these methods, the data from the response message will be returned without the LLLL if applicable and without the LLZZ(s).


IMS Version 10
V10 IMS Connect Updates
Dave Cameron
daveac@ca.ibm.com
Information Management software
© 2009 IBM Corporation

IMS Connect V10 Updates

- OTMA Degraded System Monitoring
- Transaction Expiration
- Cancel Clientid Option
- Performance APAR
- TCP/IP KeepAlive
- MAXSIZE
- Mixed Case Passwords
- BPEINI00
- IMS TM Resource Adapter Socket Reconnect

26

These are the topics that will be covered for the IMS Connect V10 updates.

OTMA Degraded System Monitoring

- Provide protocol for OTMA to inform its clients about the system health
  - Initially deals with TIB flood protection
- OTMA sends x'3C' protocol messages
  - At client bid with initial status
  - Immediately when status changes
  - Heartbeat once every 60 seconds
- x'3C' protocol message provides:
  - Overall OTMA status: available, warning, severe
  - Warning indicators; both global and local (this TMEMBER)
  - Severe indicators; both global and local (this TMEMBER)
- IMS Connect externalizes the OTMA status:
  - Expanded XIBDS for User Message Exits
  - Event 45 issued when XIBDS status updated

27

IMS V10 OTMA introduces new protocol messages to inform its clients (like IMS Connect and MQSeries IMS Bridge) of the IMS system health.

Initially OTMA will monitor and report on the TIB usage, also known as flood protection.

IMS Connect provides the OTMA status in an expanded XIBDS for User Message Exits. These exits can use this information to aid message routing decisions.

There is new Event 45 issued for the Event Recorder Exit (HWSTECL0) that signifies a change in status.

This function was introduced to IMS V10 through the service process as PK70458 (OTMA) and PK70960 (IMS Connect).

Transaction Expiration – part 1 IMS

- Provide capability to expire unprocessed transactions
  - Reduce cost of processing transactions when clients no longer care
- IMS checks for timeout at:
  - Input receiving phase – NAK x'34' (OTMA only)
  - Enqueuing phase – NAK x'34' (OTMA only)
  - GU phase – U0243 & DFS555I / DFS2224I (OTMA and IMS V11 non-OTMA)
- Specify transaction expiration at system level or message level
  - System Level (in IMS V11 only):
    - TRANSACT macro EXPRTIME
    - Output Creation Exit (DFSINSX0)
    - DRD CREATE/UPDATE TRAN/TRANDESC
  - Message Level:
    - STCK value (IMS Connect)
    - elapsed time in seconds, similar to EXPRTIME
    - overrides System Level

28

IMS V10 introduces the capability for customers to request a time value to expire unprocessed transactions. Once expired, these transactions will be deleted before being scheduled thus reducing processing costs.

IMS V10 provides transaction expiration for only OTMA transactions and at the Message Level.
IMS V11 extends this support to non-OTMA transactions and also implements System Level transaction expiration.

Transaction Expiration – part 2 IMS Connect

- Client requests transaction expiration on input transaction
  - IRM_F1_TRNEXP (x'01') in the IRM_F1 byte
  - Can set in User Message Exit
- IMS Connect uses message execution timeout value
  - IRM_TIMER or TIMEOUT from HWS configuration file
  - When client times out, unprocessed transaction will expire
- IMS Connect calculates STCK value for expiration and passes to OTMA
  - More closely match client time out than expiration elapsed time

29

IMS Connect clients request transaction expiration by setting IRM_F1_TRNEXP on the input transaction or modifying their User Message Exit.

IMS Connect uses the execution timeout value from the IRM_TIMER (or the TIMEOUT value from the HWS configuration file if no IRM_TIMER value provided). IMS Connect calculates the expiration time based on the current time plus the execution timeout value. This way the IMS Connect client timeout and OTMA transaction expiration will more closely match.

This function was introduced to IMS V10 through the service process as PK78195 (IMS), PK74017 (OTMA) and PK74024 (IMS Connect).

Cancel Clientid Option

- Addresses the issue of “Duplicate Clientid” when a client becomes disconnected from IMS Connect and tries to re-connect
- Old session can be held up:
  - In CONN State waiting for OTMA message or timer
  - In RECV State waiting for TCP/IP Keepalive
- Cancel Timer didn’t work well, particularly with Sysplex Distributor
- Cancel Clientid is an option on the first request from a client
  - Client session initially starts off as DELDUMMY
  - Client set IRM_F3 to IRM_F3_CANCID (x’80’) or

    User Message Exit can set OMUSR_FLAG1 to OMUSR_CANCID (x’20’)
- If duplicate clientid found and Cancel Clientid requested, old session cleaned up and this session proceeds

30

IMS Connect clients may encounter “Duplicate Clientid” errors when attempting to re-connect after a network failure. It can take IMS Connect an unacceptable length of time to be informed that the client has failed.

The client can use Cancel Timer but it may not be effective when used with Sysplex Distributor or when the prior session is in certain states.

The new Cancel Clientid option is specified on the first request from a client when the session clientid is being established.
IMS Connect checks for the existence of the same clientid on this Port and will clean it up if found before proceeding with the new session.

This function was introduced to IMS Connect through the service process as IMS V9 PK70327 and IMS V10 PK73829.

Performance APAR PK57574

- IMS Connect searches clients by socket to match clientid for messages from OTMA
- Search can become CPU intensive when many clients (sockets) active
- Added clientid hash table per Port to access client socket directly by clientid passed from OTMA
- Internal benchmark showed better than 20% reduction in IMS Connect CPU
  - large number of clients with high transaction rates
  - reduction dependent on number of clients and workload type

31

IMS OTMA sends response messages to IMS Connect specifying the target clientid.
IMS Connect must search through all of the clients chained by socket addresses to match the clientid.
This becomes more CPU intensive as the number of active clients (sockets) increases.

IMS Connect now implements a clientid hash table for each Port to access the clients directly by clientid when matching the clientid sent by OTMA.
This can greatly reduce the CPU required to determine the correct target socket, however the savings is dependent upon the number of active clients (sockets) and transaction rates.

TCP/IP KeepAlive

- TCP/IP will send KeepAlive packets after period of inactivity on a socket to determine if partner is still healthy
  - can determine when partner terminates abruptly
- To minimize network overhead of KeepAlive packets, TCP/IP stack KeepAlive interval is usually fairly long
- IMS Connect provides SO_KEEPALIVE to override the stack interval
- New PORT= definition in HWS configuration file

  PORT=(ID=name,KEEPAV=nnnn)

  Where:

  PORT= is the statement to define the Port.

  ID= Defines the Port name

  KEEPAV= nnnn is the KeepAlive override value in seconds.

  Default of 0 uses stack value.
- A Port can only be defined once in PORTID, SSLPORT or PORT

32

Many customers specify a large TCP/IP KeepAlive interval to reduce the network overhead of KeepAlive packets. This may prevent IMS Connect from being informed of client problems in a timely manner.

IMS Connect introduces a new PORT definition in the HWS configuration file that allows customers to override the TCP/IP KeepAlive interval for socket on specific Ports.

Each Port can only be defined once as illustrated in the following example:

```
TCPIP=(HOSTNAME=TCPIP,PORTID=(9999,7777,LOCAL),RACFID=GOFISHIN,
SSLPORT=(8888),SSLENVAR=HWSCFSSL,TIMEOUT=3000,MAXSOC=65535,
PORT=(ID=9998,KEEPAV=600),PORT=(ID=9997,KEEPAV=120),
PORT=(ID=9996),PORT=(ID=9995,KEEPAV=300), ...)
```

This function was introduced to IMS V10 through the service process as PK72652.

MAXSIZE

- IMS Connect reads input messages from clients for a length specified by the first four bytes (LLLL)
  - internal maximum message size limit of 10MB
  - HWSP1440E INVALID LENGTH SPECIFIED IN MESSAGE PREFIX
- New MAXSIZE parameter on HWS statement in HWSCFG
  - overrides default 10MB, either smaller or larger

MAXSIZE=

A decimal field set to the maximum message size that will be allowed in the four byte total length field that proceeds the IRM.
Use the MAXSIZE= parm to override the internal default of 10,000,000.

33

IMS Connect uses the first four bytes of input messages from clients to determine the total size of the message. There is an internal size limit of 10MB for input messages.

IMS Connect introduces a new MAXSIZE parameter on the HWS statement in the HWS configuration file to override this internal default value.
Customers may specify either a larger or smaller value to limit the size of input messages.

This function was introduced to IMS Connect through the service process as IMS V9 PK57769 and IMS V10 PK57770.

Mixed Case Password

- IMS Connect provides support for RACF mixed-case passwords
- PSWDMC option on HWS statement in the HWS configuration file
  - 'R' – determine mixed-case password option from RACF (default)
  - 'Y' – always enable mixed-case passwords
  - 'N' – always disable mixed-case passwords
- dynamically change option using

  UPDATE MEMBER TYPE(IMSCON) SET(PSWDMC(xxx))

  where 'xxx' is
    - RCF – determine mixed-case password option from RACF
    - ON – enable mixed-case passwords
    - OFF – disable mixed-case passwords

34

IMS Connect provides support for RACF mixed-case passwords with three options.

The default is to determine the mixed-case password option from RACF.

This allows customers to implement RACF mixed-case passwords with IMS Connect active and the change will automatically be picked up.

The mixed-case password option can be specified in the HWS configuration file or changed dynamically with the UPDATE MEMBER command.

This function was enhanced in IMS Connect through the service process by PK80037.

BPEINI00

- IMS Connect runs in Key 7 so has required an update to the Program Properties Table (PPT) for HWSHWS00
- BPEINI00 is pre-defined in the PPT as supplied by z/OS
- IMS Connect now allows either BPEINI00 or HWSHWS00 to be used
  - if using BPEINI00, no longer require update to PPT
- change required to IMS Connect JCL for BPEINI00

```
//STEP1  EXEC  PGM=BPEINI00,REGION=&RGN,TIME=1440,
//     PARM='BPECFG=&BPECFG,BPEINIT=HWSINI00,HWSCFG=&HWSCFG'
```

  - Note that must add “BPEINIT=HWSINI00”

35

IMS Connect has required customers to update the Program Properties Table (PPT) to add HWSHWS00. The PPT supplied by z/OS already includes an entry for BPEINI00.

IMS Connect is enhanced to allow customers to use BPEINI00 thus removing the requirement to update the PPT. The IMS Connect JCL must be updated to use BPEINI00.

This function was introduced to IMS Connect through the service process as IMS V10 PK41284.

TMRA Socket Reconnect

- IMS TM Resource Adapter (TMRA) Version 10 introduces socket reconnect
- will try to reestablish a stale connection in a connection pool when one of the connections encounters a communication problem in the process of sending a request to or receiving a response from IMS Connect
- when TMRA detects a stale connection, the adapter will throw an exception, but will set up an internal flag about this issue
- subsequent connections will be compared with the time the initial stale connection was discovered and if necessary TMRA will reconnect before submitting the interaction request
- the aged timeout property in WebSphere Application Server connection pool settings can be used to redistribute connections in an IMS Connect Sysplex Distributor environment

36

IMS TM Resource Adapter (TMRA) connections to IMS Connect are managed by the WebSphere Application Server (WAS) connection pool.

When network problems are encountered or IMS Connect is brought down, WAS may not be aware that the connections are no longer good.

TMRA may encounter many stale connections from the connection pool.

To minimize the impact, TMRA will throw an exception on the first stale connection encountered and sets an internal flag.

Subsequent connections are compared with the time the initial stale connection was discovered and reconnected if necessary.

If an instance of IMS Connect is not available with Sysplex Distributor while the connections are being established, the sessions will not be evenly distributed when it again becomes available. Using the aged timeout property in the WAS connection pool settings can help to redistribute the sessions.

IMS Version 10

V10 Synchronous Callout via IMS Connect

Jack Yuan
jackyuan@us.ibm.com

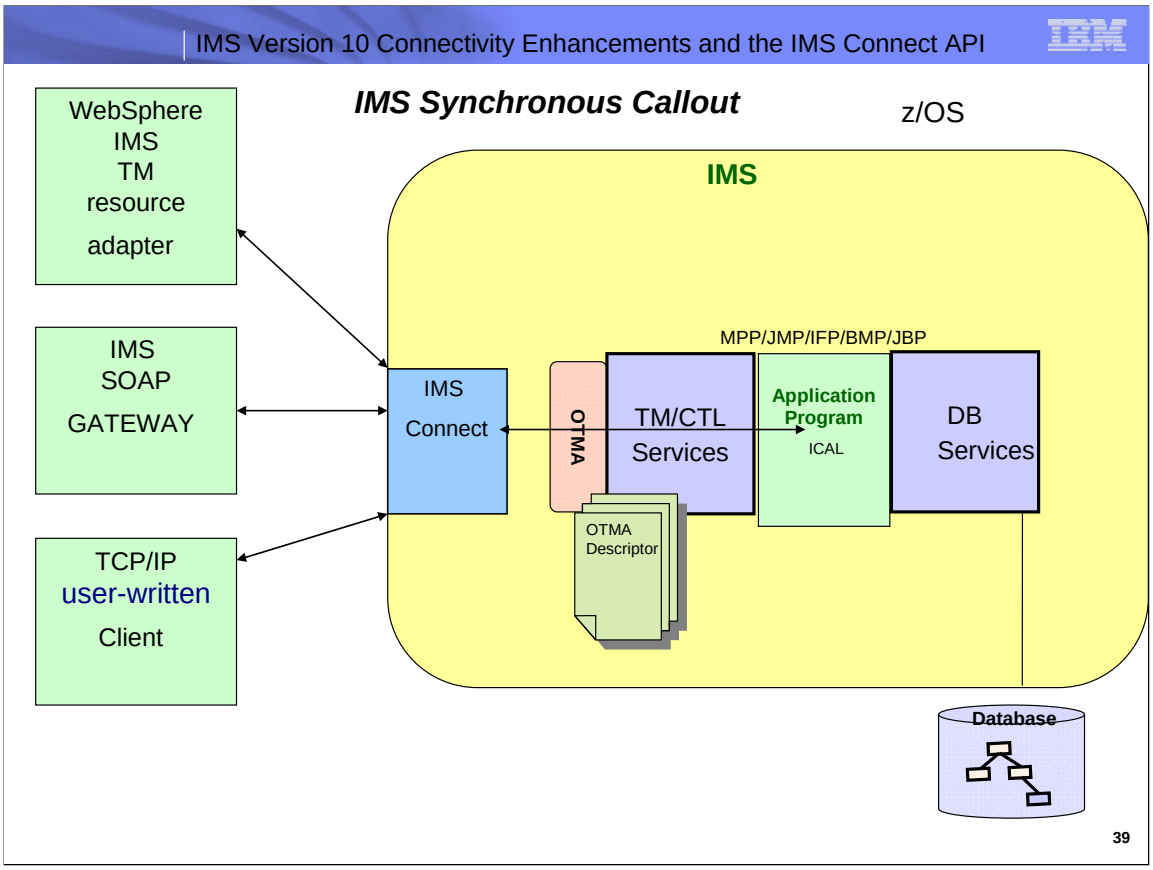
Information Management software

Synchronous Callout Request

- Capability
  - IMS applications can invoke external applications and synchronously receive a response in the same IMS transaction instance
  - ICAL new DLI call
    - AIBTDLI call provides synchronous callout request
  - Timeout capability
    - Control the time for synchronous callout processing
  - Relieve 32K segmentation limitation
    - IMS Connect and OTMA handle buffer and segmentation internally

38

IMS V10 Synchronous Callout function enhances the current asynchronous IMS Callout support to allow the IMS application to callout synchronously and wait for the response to come back. Using this programming style, the IMS application program can initiate direct communication with other external application programs and receive the response in the same IMS transaction instance



This diagram shows that with the IMS callout support IMS applications can be a client and server.
IMS provides bi-directional access between IMS applications and external application and servers.
The IMS Application Program can callout to:

- Callout to user-written IMS Connect client
- Callout to WebSphere EJB/MDB using IMS TM Resource Adapter
- Callout to Web Service Provider using IMS SOAP Gateway

ICAL AIBTDLI Interface

Call AIBTDLI USING ICAL,aib,REQ_area,RESP_area

- ICAL is new DL/I call verb
  - SENDRECV is the new subfunction
- REQ_area is the Request data area for sending data
- RESP_area is the Response data area for returned data
  - REQ and RESP messages are not recoverable
  - req-area and resp-area do not specify LLZZ, data can be > 32K

40

The ICAL synchronous call function is only supported using the AIBTDLI interface in IMS TM runtime environments for IFP, MPP, BMP, JMP and JBP regions
AIBTDLI USING ICAL,aib,REQ_area,RESP_area
ICAL is the new function for call processing
SENDRECV is the new sub function for sync callout
REQ_area is the request data area for sync callout. Note do not specify LLZZ
RESP_area is the response data area for returned data. Note do not specify LLZZ
Req-area for data sent to application . Can be greater than 32K
Resp-area data returned from called application can be greater that 32K

ICAL AIBTDLI Interface . . .

- AIB
  - AIBID = DFSAIBbb
  - AIBLEN = AIB length
  - AIBSFUNC = SENDRECV
  - AIBRSNM1 = 8 byte OTMA Descriptor name
  - AIBRSFLD = Timeout value
    - 4 byte field for time value 100th seconds. System default is 10 sec.
  - AIBOALEN = **REQ_area** length
    - As an input parameter: 4 byte field contains the length of the request area
    - As an output parameter: Actual length of the response message
  - AIBOAUSE = **RESP_area** length
    - As an input parameter: 4 byte field contains the length of the response area
    - As an output parameter: Length of the response message.
  - AIBRETRN = AIB Return code
  - AIBREASN = AIB Reason code.
  - AIBERRXT = 2 byte sense code from external application

41

When partial data is returned because the response area is not large enough, AIBOAUSE contains the length of data returned in the response area and AIBOALEN contains the actual length of the response message.

OTMA ALTPCB Descriptors – Maximum number = 256 Synchronous Callout

- DFSYDTx member of IMS.PROCLIB
  - TYPE: Destination type
  - TMEMBER: OTMA Target Client
  - TPIPE: Destination Name
  - SMEM: YES|NO
  - ADAPTER: Type of IMS Connect Adapter
  - CONVERTR: Routine called by Adapter
  - **SYNTIMER=timeout**
    - **If both ICAL & Descriptor specify timeout, the lower value is used**

IMS 11 Type -2 Command

```
UPDATE OTMADESC NAME(OTMASYN) SET(SYNTIMER(5000))
```

42

IMS 10 introduces the type D of OTMA Destination Routing Descriptor which externalizes the routing definitions and specifications for callout messages without IMS user exits. It is read and initialized at IMS startup.

The IMS 11 provides the Type-2 UPDATE commands to dynamically create, update, or query the type D Destination Routing Descriptors.

The command can be used to dynamically change the SYNTIMER value in the Descriptor

UPDATE OTMADESC NAME(OTMASYN) SET(SYNTIMER(5000))

Example

D SOAPGW1 TYPE=IMSCON TMEMBER=HSW2 TPIPE=HWS2SOAP

D SOAPGW1 ADAPTER=XMLADPTR CONVERTR=XMLCNVTR SYNTIMER=2000

DFSYDTx

TYPE=Specifies output is for IMS Connect (IMSCON)

TMEMBER=IMS Connect OTMA TMEMBER name

TPIPE= TPIPE name the client specifies for Resume Tpipe call

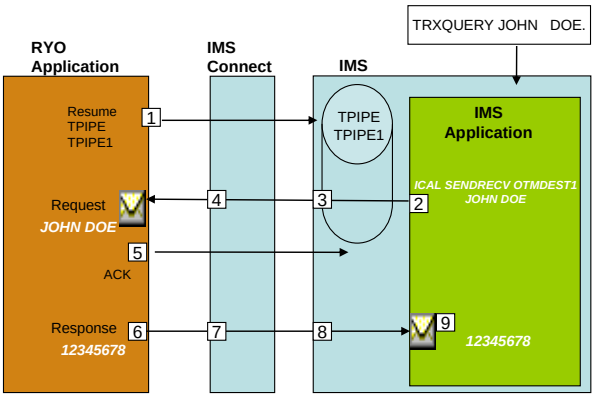
SMEM= Specifies whether (YES) or not (NO) this destination is a supermember

ADAPTER= Identifies the IMS Connect adapter to be used on these messages.

CONVERTR= Name of the converter to be used by the adapter specified on ADAPTER=

SYNTIMER Sets timeout value for ICAL synchronous callout. The AIBRSFLD parameter in the ICAL SENDRECV call can override this value. If no timer (or value of 0) is specified IMS will use 10 seconds as the default to timeout the synchronous call process in the dependent region.

V10 Synchronous Callout – RYO Client Message Flow



1. Resume TPIPE Archlvl=3, sync-only or sync-and-async
2. ICAL with request area and AIBOALEN for LLLL
3. OTMA breaks message into 32K segments
   OMPFX | LLZZ Data | {OMPFX | LLZZ Data …}
4. ICON combines data and sends to client
   LLLL | LL CORTKN | {LL RMM} | LLLL Data | LL CSM
5. Client sends ACK
6. Client sends Sync Callout response
   LLLL | LL IRM | LLLL Data | 00040000
   Note: response IRM contains CORTKN from request
7. ICON breaks message into 32K segments
   OMPFX | LLZZ Data | {OMPFX | LLZZ Data …}
8. OTMA combines data and returns to ICAL
9. ICAL gets data in response buffer and AIBOAUSE is LLLL

Client Programming – NAK Synch Callout

- NAK-STOP
  - Reject message and terminate Resume TPIPE
  - AIBRETRN=X'100' AIBREASN=X'108' return to ICAL
    - Client can provide extended error code for AIBERRXT
- NAK-Continue (Reroute)
  - Reject message, continue retrieving messages for Resume TPIPE
  - AIBRETRN =X'100' AIBREASN=X'108' returned to ICAL
    - Client can provide extended error code for AIBERRXT
- NAK-Pause (Hold Message)
  - Terminate this Resume TPIPE and keep message
    - OTMA will hold message for another Resume TPIPE request
    - Used if message is OK but don't want to receive more messages
    - Note ICAL is waiting for response message

44

The Synch Callout message retrieved by Resume TPIPE is CM0 so requires ACK or NAK

There are 3 types of NAK message for a sync callout message: NAK-STOP, NAK-Continue, and NAK-Pause.

Enhanced Commands

- /DISPLAY ACTIVE REGION
  - New status for region waiting on synchronous callout response
    - WAIT-CALLOUT
  - Continuation line for region waiting on synchronous callout response
    - TMEMBER name TPIPE: tpipename

```
REGID  JOBNAME    TYPE   TRAN/STEP  PROGRAM   STATUS             CLASS
    1  MPP1A      TP     APOL11     APOL1     WAITCALLOUT      1
                                     TMEMBER HWS TPIPE:   TPIPE1
       JMPRGN     JMP    NONE
       JBPRGN     JBP    NONE
       BATCHREG   BMP    NONE
       FPRGN      FP     NONE
       DBTRGN     DBT    NONE
       DBRZCSAJ   DBRC
       DLIZCSAJ   DLS
*08213/165100*
```

45

The /DIS ACTIVE REGION command response now includes TMEMBER as well as TPIPE. You can use the /DIS TMEM xx TPIPE yy SYNC to find status.

Enhanced IMS commands . . .

- /DIS TMEMBER TPIPE
  - WAIT-S (WT-S)
    - The transaction pipe is waiting for an ACK or NAK for a synchronous callout message
- /DIS TMEMBER TPIPE SYNC
  - Display the detailed sync callout message count and status
- /PSTOP
  - Clear the wait in the region
  - Dequeues synchronous callout messages from the TPIPE

1. /STOP REG ABDUMP
   - Clear the wait in the region and terminate the application program

- /STOP TMEMBER TPIPE
  - Clear state of all messages for the TPIPE
- /STOP OTMA
  - Clear all the ICAL messages for all the TPIPEs.
  - New ICAL synchronous callout requests rejected
- These stop commands return to ICAL
  - AIBRETRN = 100
  - AIBREASN = 10C

46

When the /STOP OTMA command is issued, it will clear or reject all the ICAL messages for all the transaction pipes.
/PSTOP, /STOP OTMA, or /STOP TMEMBER TPIPE, may be used to end the ICAL,
In this case, the IMS application will get return code 100 and reason code 10C.

Prerequisites

- Software requirements
  - IMS and IMS Connect Version 10
    - PK70078, PK70330 ,PK73224 - Precondition
    - PK71135 ,PK74168 – Activation
  - For synchronous callout to J2EE application/Web Service in WebSphere Application Server (WAS):
    - IMS TM Resource Adapter Version 10.3*
    - WAS 6.0* for callout to EJB
    - WAS 6.1* for callout to MDB
  - For synchronous callout to Web Service using IMS SOAP Gateway:
    - IMS SOAP Gateway Version 10.1.1
- Tooling
  - For synchronous callout to J2EE application/Web Service in WAS
    - Rational Application Developer (RAD) V7.007 or later
      - V7.5 is recommended
      - IMS TM RA ifix 10.3
    - WebSphere Integration Developer (WID) V6.1* or later
    - MDB generation requirement to RAD/WID
  - For synchronous callout to Web Service using IMS SOAP Gateway
    - Rational Developer for System z (RDz) 7.1.1* or later

47

The synchronous callout support requires IMS and IMS Connect 10 with the SPE.

APAR PK70078 and PK73224 contains all the IMS changes (e.g. IMS Systems, OTMA) and APAR PK70330 contains all IMS Connect changes.

Activation APARS are PK71135 and PK74168

For callout to WebSphere application using IMS TM Resource Adapter, and updated IMS TM Resource Adapter is required

WAS 6.0 or later is required for the Client Managed programming model

WAS 6.1 or later is required for the MDB programming model.

RAD V7.5 or WID 6.1 or later tooling is required to develop your Java Application code.

IMS SOAP Gateway support is provided in 10.1.1

RDz Version 7.007 or later tooling is required. V7.5 is recommended

Thank You for Joining Us today!

Go to **www.ibm.com/software/systemz** to:

- Replay this teleconference
- Replay previously broadcast teleconferences
- Register for upcoming events

48