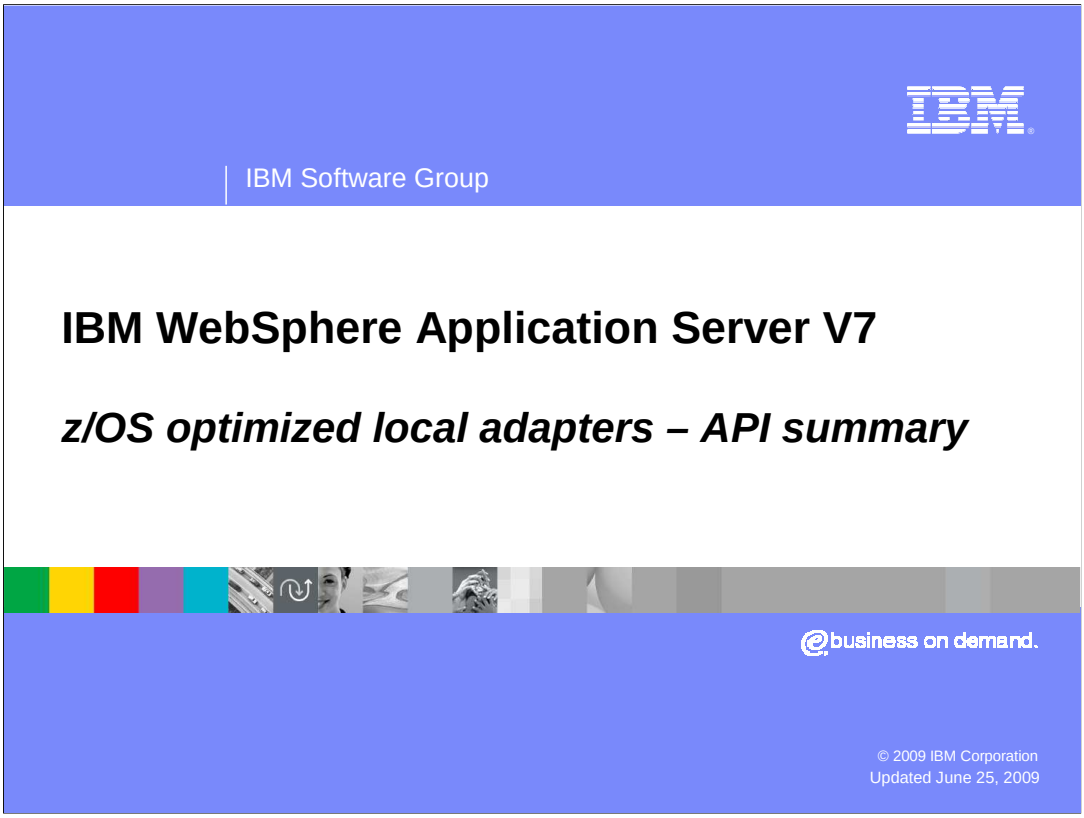


This presentation shows the API summary for optimized local adapters for WebSphere® Application Server V7.0.0.4 on z/OS®.

Overview

- Optimized local adapter API summary

This presentation briefly discusses the new APIs for the optimized local adapters function for z/OS.

Section

Basic inbound APIs

The next several slides will discuss the basic APIs available with the optimized local adapter feature that will drive work into WebSphere Application Server. These are considered inbound APIs.

Basic APIs to drive work into WebSphere

- BBOA1REG -- Register
  - Creates a connection between a native process and a WebSphere Application Server instance
  - Sets up a connection pool with min/max connections
  - Defines transactional and security characteristics of connections
  - Caller provides a 12 character ‘register name’ used on future API calls
  - Caller can register multiple times within an address space, using multiple register names

The first API is BBOA1REG. The register API creates a connection between a native process and a WebSphere Application Server instance. Those connections are maintained using control blocks that are hung off the daemon. When you call register you are setting up a connection pool with a minimum and maximum number of connections. These are the physical connections between your external address space (batch or CICS®) and a WebSphere Application Server instance. These connections are different than the ones specified when installing the RAR file. Also defined are the transactional and security behaviors of the connection, which are set using flags. You must provide a 12 character register name that is used to identify the group of connections between your program and a WebSphere Application Server instance. You will have to provide that name on future API calls. There are restrictions to the name. The name must be unique from the address space you are running in when you call from that address space into the WebSphere Application Server. The name must be globally unique when you call from the WebSphere Application Server into a CICS or batch program. You can register multiple times within an single address space, you must use different register names. The names must be unique.

Basic APIs to drive work into WebSphere

- BBOA1URG – un-register
  - Removes the connection between the native process and WebSphere Application Server instance
- BBOA1INV – invoke an EJB™ method
  - Sends a request into WebSphere, and receives the response
  - Data areas provided by caller for request data and response data
  - Target EJB JNDI name (for EJB home) is specified by caller
  - Target EJB must implement a pre-defined home and remote interface

The un-register API undoes everything that was done during register. You will provide the unique register name you created and the optimized local connectors will remove the connection between the native process and the WebSphere Application Server instance.

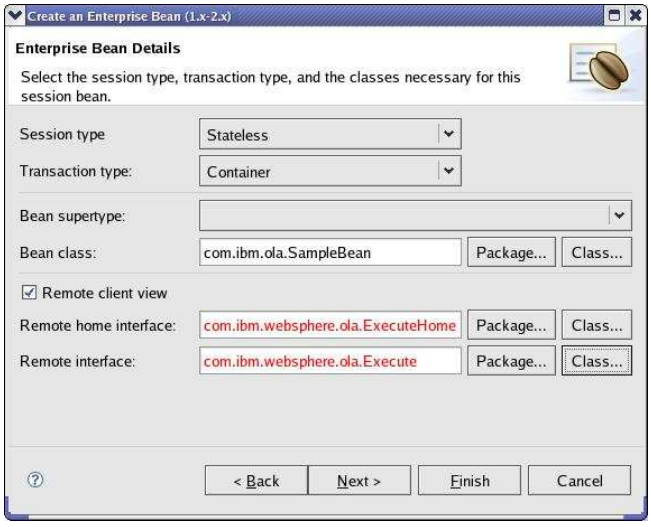
BBOA1INV is an API that is used to invoke an EJB™ in the WebSphere Application Server instance. This API will send a request into a WebSphere Application Server and receive the response in one call. Data areas are provided by the client program for the request and response data and are a fixed size. The target EJB JNDI name is specified by the caller, this is called the service name. The EJB that you want to call must implement pre-defined home and remote interfaces that were provided to you through the optimized local adapters jar file.

Basic APIs to drive work into WebSphere

- EJB definition
  - Only stateless session beans are supported
  - Remote home interface must be com.ibm.websphere.ola.ExecuteHome
  - Remote interface must be com.ibm.websphere.ola.Execute
  - Business logic is contained in the byte[] execute(byte[] input) method provided on the remote interface
  - Remote/Home interfaces in ola_apis.jar (see install slides)

Optimized local adapters support stateless session beans. The home interface must be com.ibm.websphere.ola.ExecuteHome, the remote interface must be com.ibm.websphere.ola.Execute. There is one method defined on the remote interface called execute. The method takes a byte array as input and uses a byte array as output, those two byte arrays correspond to the request and response buffers that you provided on BBOA1INV.

EJB definition in IBM Rational Application Developer



Above is a Rational® Application Developer screen capture of how you create your EJB. This is where you specify your home and remote interfaces.

Section

Advanced inbound APIs

The next several slides will discuss the advanced APIs available with the optimized local adapter feature that will drive work into a WebSphere Application Server. These APIs are used on larger client application or if you need to have more control over the requests and responses. These are considered inbound APIs.

Advanced APIs to drive work into WebSphere

- BBOA1CNG – Get Connection
  - Gets a connection from the connection pool
  - Caller provides register name from BBOA1REG
  - Caller provides a 12 byte area to copy the connection handle to
- BBOA1CNR – Release Connection
  - Puts a connection back in the connection pool
  - Caller provides the 12 byte connection handle from BBOA1CNG

The first advanced API is BBOA1CNG, or get connection. This API will get a physical connection from the connection pool. The caller provides a register name and a 12 byte area. Optimized local adapters will put a copy of the connection handle into the 12 byte area that you will need to save and supply during future calls.

The connection release API (BBOA1CNR) will release a connection associated with the connection handle that you provided. The connection handle it put back into the connection pool.

Advanced APIs to drive work into WebSphere

- BBOA1SRQ – Send Request
  - Sends a request into WebSphere
  - Caller provides a connection handle and data area for the request data
  - JNDI name conventions same as BBOA1INV
  - Response length returned if synchronous mode is used

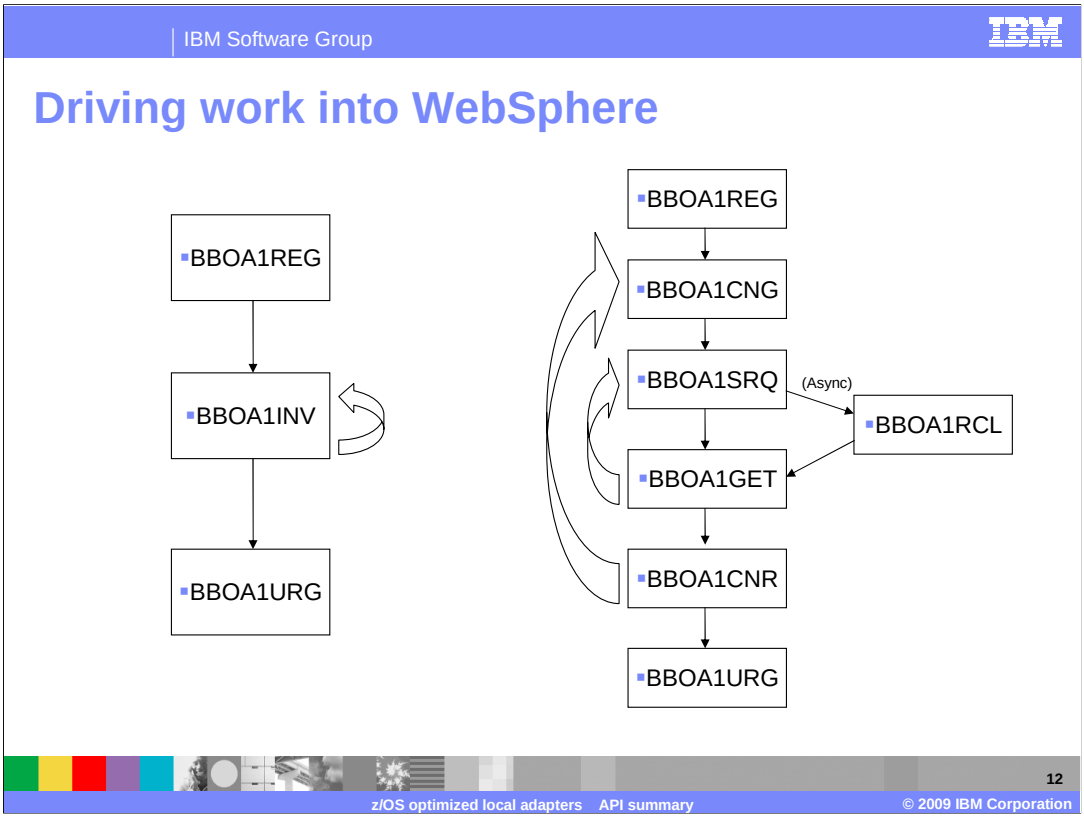
BBOA1SRQ, the send request API, is used to send a request into a WebSphere Application Server instance. The caller provides a connection handle and the data area for the request data to be returned. Optimized local adapters will return the length of the data, not the data itself, to the caller to acquire the necessary storage. The service that you are going to call is specified the same way as if you were using invoke. You will provide the JNDI name of the stateless session bean that implements the home and remote interface. This method can be done synchronously or asynchronously.

Advanced APIs to drive work into WebSphere

- BBOA1RCL – Receive response length
  - Only valid after a BBOA1SRQ in asynchronous mode
  - Caller provides the connection handle used for BBOA1SRQ
  - Response length returned, if available
- BBOA1GET – Get Data
  - Receives the response data for a request
  - Caller provides the connection handle used for BBOA1SRQ
  - Caller provides a data area to copy the response into
    - The size of the data area should be obtained from BBOA1SRQ or BBOA1RCL

If you specified the asynchronous flag on the send request API, you must call the BBOA1RCL, or receive response length API, to see if your send request had returned. The caller must provide the connection handle that was used in the send request and a response length is returned if available.

Once you have the length of the data, and have allocated enough storage to hold the data, the BBOA1GET, or get data, API is invoked. The data is be copied into the data area provided by the caller.



Shown here is a flow diagram that demonstrates the order in which you run the APIs. The left flow is a simple flow using first the register API and then the invoke API. Invoke can be called in a loop until you have completed all the services that were necessary. Once completed you can call the un-register API. It is possible to configure your application to run invoke continuously in a loop, only calling register when your program starts and un-register when it ends. Another option is, if your application has several registrations and you can register / un-register as particular tasks come and go.

The flow on the right uses the advanced APIs. You can call register, get a connection and send a request. If the request is run asynchronously you need to call receive response length. Once the length is known, get data can be called. You can loop within the get data, send request APIs to process as many requests as required or you can release the connection into the pool when the transaction has completed. Once the connection is released it is possible to loop back to get another connection from the pool and begin the flow again. The un-register API can be called when you have completely finished your processing.

Section

Basic outbound APIs

The next several slides will discuss the basic APIs available with the optimized local adapter feature that will drive work into a batch or CICS system. These are considered outbound APIs.

Basic APIs to drive work into Batch/CICS

- BBOA1SRV – host a service
  - Accepts work from a WebSphere application component
  - Caller provides the register name used on BBOA1REG call
  - Caller provides a service name (1-255 bytes) to identify the service
    - WebSphere client will use register name + service name to find the service
    - Service name can contain one or more asterisk '*' characters
  - Caller provides a data area to copy request data into
  - Caller provides a 12 byte area to copy a connection handle into
    - Connection handle is used to send the response

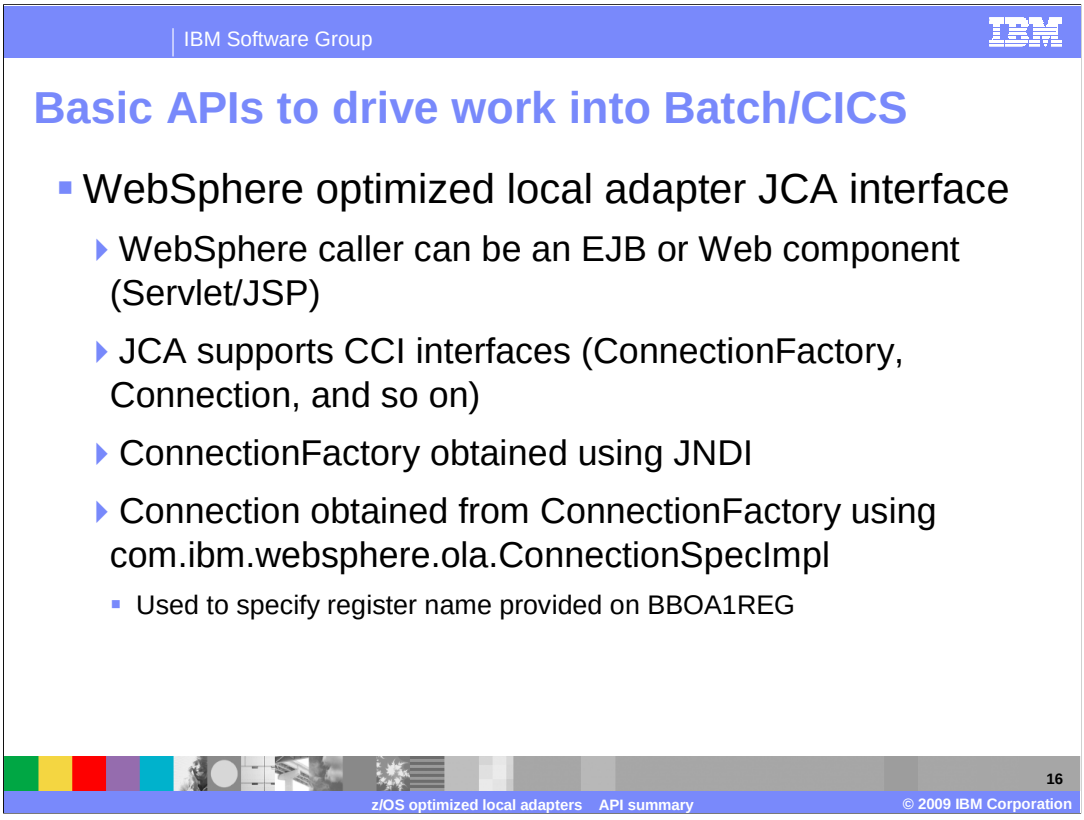
The API your client will invoke to receive work is called BBOA1SRV or host a service. This is how your batch or CICS system will accept work from a WebSphere Application Server system. Like the other APIs, the caller must provide the register name acquired when using the register API. You will also need to provide a service name when calling from CICS or batch process into a WebSphere Application Server system, which is the JNDI home of the EJB that you want to invoke. In the other direction, this is a name that you choose and must be between 1 and 255 characters. The name will describe the service that you are making available to the WebSphere Application Server caller. The name can contain the * character which will allow your system to accept more than one type of service. If you use the *, when work arrives, optimized local adapters will fill in the name of the service that the caller requested. The caller is a Java application in a WebSphere Application Server so a service name is in the request and optimized local adapters will match that service name with the service names that are waiting by callers of BBOA1SRV. For example, if your service name is service* and the Java application requests service1, optimized local adapters will match the service1 with service* allowing your batch or CICS system to be able to process the request for service.

You also will provide an area where you will receive the callers parameters into, which implies you know how much data is returned. Finally, the caller needs to provide a 12 byte area that the connection handle is copied into. The connection handle is used to send the response back to your WebSphere Application Server system.

Basic APIs to drive work into Batch/CICS

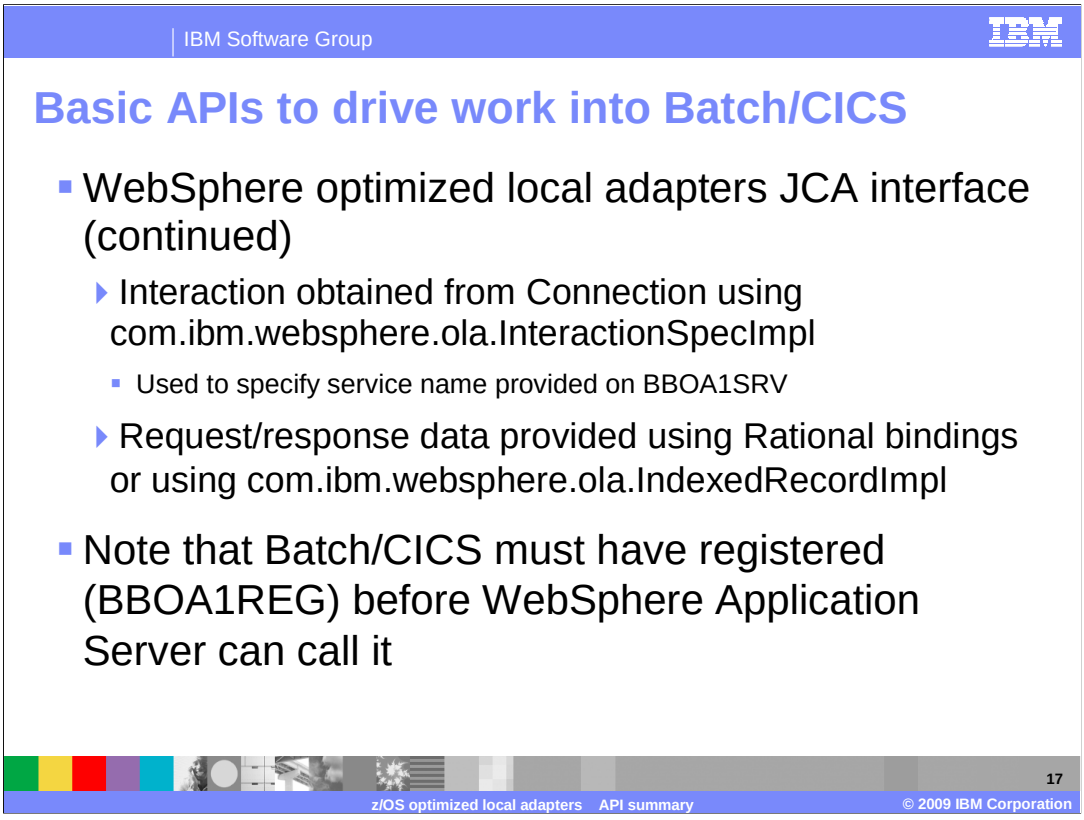
- BBOA1SRP – send a response
  - Sends the response to a BBOA1SRV call to WebSphere
  - Caller provides connection handle from BBOA1SRV, and response data
  - Caller must release connection when finished (BBOA1CNR)

Once you have your service registered, and processing has completed, you will have to send the response back to your WebSphere Application Server instance. The BBOA1SRP, or send response, API is used to complete this task. The caller provides both the connection handle and the response data area. In this situation the listener knows in advance how much data is returned. When the data has been returned the caller must release the connection by calling the BBOA1CNR API.



The Java half of an outbound application is going to use the JCA adapter that was provided with the optimized local connectors feature. The JCA interface for optimized local adapters can be an EJB or Web component, such as a servlet or JSP. Most likely your application is going to have a resource reference that will ook up the JCA adapter. The JCA supports the standard CCI interfaces that are defined in the JCA 1.5 specification, such as a ConnectionFactory, Connection and interaction object that you are going to use to talk to your resource backend.

Using the JNDI name that you provided when you installed the adapter, you are returned a reference to a connection factory and you will use that connection factory to get a connection using the standard CCI interface. You need to specify who you want to connect to so optimized local adapters provides a connectionSpecImpl object where you specify the register name that you want to connect to. This is the 12 character name that you specified on your BOA1REG call.



After you have your connection to your target register name you will create an interaction on that connection using the standard CCI get interaction method on the connection object specifying the service you want to talk to. The service name goes inside the InteractionSpecImpl object that optimized local adapters has provided. WebSphere Application Server then takes the register name and service name and finds the service that is waiting for a request. Regarding the request data, when you get your interaction object there is a method on the interaction object called execute. Execute takes the interactionSpec that was setup with your service name and an input and output parameter. The input / output can be defined either as a byte array or mapping of a copybook or C structure that you created using the Rational tools. IBM Rational Application Developer has a set of tools that you can use to create a Java mapping of a copybook or another structure that you use in a CICS or IMS™ program. The tools are called J2C CICS / IMS mapping tools in IBM Rational Application Developer. After this is complete the object you have will implement an interface called record, this interface is supported by the optimized local adapters. You can provide input as that Java object and optimized local adapters will turn that object into a binary blob that is sent over to your program that is waiting for the service. That program will then be able to use the information in the copybook form. This avoids you having to write serialization / de-serialization code. Note that ordering is important. Your Java application is going to get an error that the registration cannot be found if WebSphere Application Server calls the Java application prior to the registration occurring.

Section

Advanced outbound APIs

The next several slides will discuss the advanced APIs available with the optimized local adapter feature that will drive work into a batch or CICS system. These are considered outbound APIs.

Advanced APIs to drive work into Batch/CICS

- BBOA1RCA – Receive any request
  - Accepts work from a WebSphere application component
  - Caller provides the register name used on BBOA1REG call
  - Service name and Connection Handle specification is the same as BBOA1SRV
  - Returns the size of the request data
    - BBOA1GET is used to copy the request data into a data buffer

BBOARCA, or receive any request, accepts work from a WebSphere Application Server and provides you with the length of input data. This API can be used if you wanted to receive requests but did not know how much data you were going to be passed. Optimized local adapters will tell you how much data to be passed enabling you to acquire the necessary storage. Once you have acquired the storage you can receive the data. Optimized local adapters will provide you with a connection handle. The connection handle is used to send a response and receive the actual data that the caller provided.

Advanced APIs to drive work into Batch/CICS

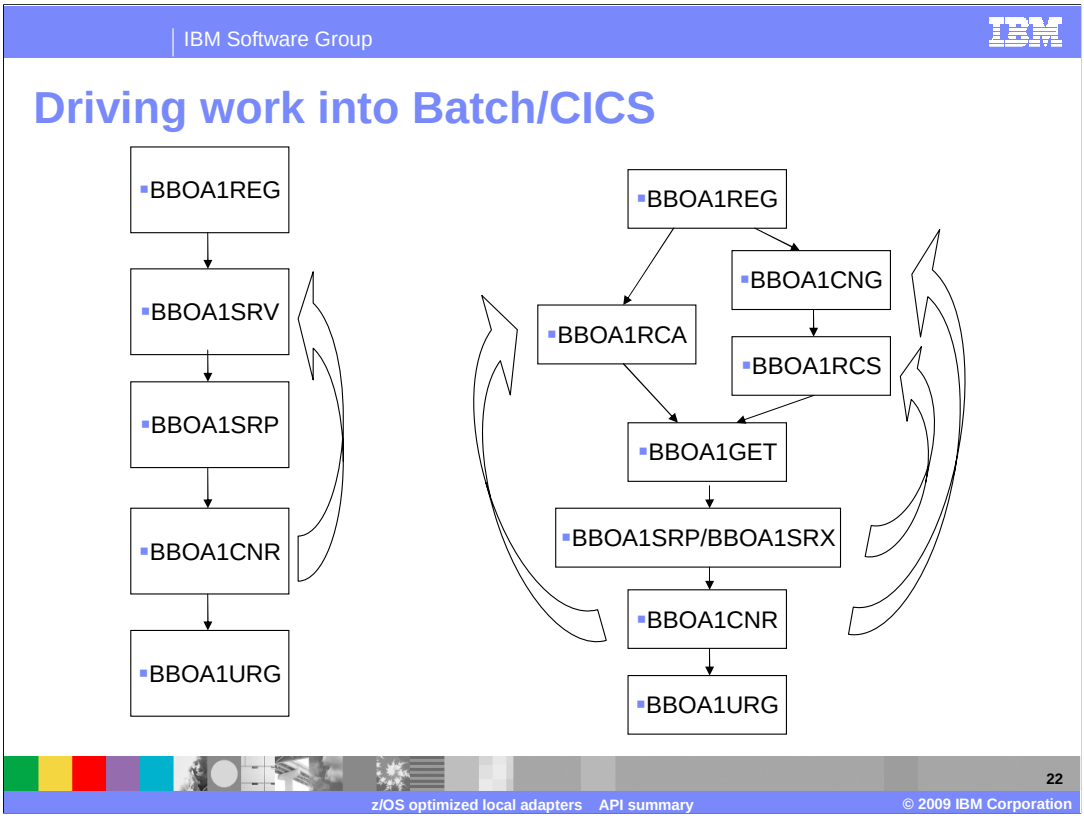
- BBOA1RCS – Receive a specific request
  - Accepts work from a WebSphere application component
  - Caller provides a connection handle from BBOA1CNG or BBOA1SRV
  - Service name specification is the same as BBOA1RCA
  - Returns the size of the request data

BBOA1RCS, or receive a specific request, accepts work from a WebSphere Application Server instance. The only difference between this API and BBOA1RCA is that BBOA1RCS does not provide a connection handle for you. You need to get a connection handle from the connection pool, using either BBOA1CNG or BBOA1SRV, and then call BBOA1RCS. The request data size is returned, allowing you to acquire the storage and then getting a copy of the data using the BBOA1GET API. Once you have the data, processing is the same as the host service API.

Advanced APIs to drive work into Batch/CICS

- BBOA1SRX – Send Response Exception
  - Sends a javax.resource.ResourceException to the WebSphere caller
  - Caller provides the connection handle and exception text

The final API is BBOA1SRX. BBOA1SRX, or send response exception, allows you to provide an exception string instead of a response buffer. Optimized local adapters will then send a javax.resource.ResourceException to the WebSphere Application Server caller. This is the standard JCA way of creating an exception. Optimized local adapters will fill in the exception with the string you provided. This allows you to throw the exception back to your application and your application can catch the exception and print it to a log, or whatever action is appropriate. This API gives your native application the ability to create a Java exception.



Shown here is a flow diagram that demonstrates the order in which you call the APIs. The left flow is a typical flow for the basic outbound APIs. This flow calls the register, hosts a service, sends a response, releases the connection and un-registers. You can choose to loop back to hosting a service after releasing the connection and not do the un-register until you are done. It is important to remember to release the connection or your system will run out of connections.

On the right side is a typical flow for the advanced APIs. After the register you can either get a connection or receive any request which gets a connection for you. If you are in a multi-treaded environment you can manage the connections yourself. If you get a connection you will have to issue the receive specific request API. Once this is complete you will get the data and then send a response or send a response exception. Finally, you will release your connection and then un-register. You can loop back from the send response or send response exception to receiving a specific request. Also after releasing the connection you can loop back to either getting a new connection or receiving any request. Having the ability to loop back prevents you from having to un-register each time.

Summary

- Basic and advanced inbound APIs
- Basic and advanced outbound APIs

This presentation has discussed the basic and advanced, inbound and outbound APIs provided with optimized local adapters.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WASv7zOS_OLA_APISummary.ppt

This module is also available in PDF format at: ../WASv7zOS_OLA_APISummary.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

CICS Rational WebSphere z/OS

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Rational is a trademark of International Business Machines Corporation and Rational Software Corporation in the United States, Other Countries, or both.

EJB, Java, JSP, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.