WebSphere Application Server z/OS Version 7

WebSphere Optimized Local Adapters

Planning Guide and Reference

Version Date: November 12, 2012

See "Document Change History" on page 22 for a description of the changes in this version of the document

IBM Advanced Technical Skills
Gaithersburg, MD

WP101490 at
ibm.com/support/techdocs

Many, *many* thanks to **Jim Mulvey**, **Tim Kaczynski** and **Dave Follis** of the WAS z/OS development team.

The WAS z/OS support team in IBM Advanced Technical Skills consists of **John Hutchinson**, **Mike Kearney**, **Louis Wilen**, **Lee-Win Tai**, **Mike Loos** and **Don Bagwell**.

We also receive wonderful support from **Ken Hain** and **Brian Pierce**.

Mike Cox, Distinguished Engineer, serves as technical advisor to all our activities.

Table of Contents

Reference

Quick Reference Facts

Minimum Level of WAS z/OS	Function first made available in 7.0.0.4
Level of WAS z/OS with Updates to WOLA	7.0.0.12
External Address Spaces Supported in Latest	CICS, IMS (MPR, IFP, BMP, DL/I batch), Batch, USS and ALCS
Programming Languages Supported	Java (in WAS), COBOL, C/C++, High Level Assembler
Transaction *(restrictions apply, see details pages)*	• 2PC inbound CICS to WAS in 7.0.0.4 • 2PC outbound WAS to CICS in 7.0.0.12 • SyncOnReturn IMS to WAS in 7.0.0.12 • CM0 or CM1 WAS into IMS in 7.0.0.12
Identity Propagation *(restrictions apply, see details pages)*	• WAS thread identity into CICS and IMS (MPP and IFP, but not BMP) • CICS region ID or application ID into WAS • IMS thread ID into WAS
InfoCenter	`publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp`
Techdocs	`ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101490`

Key InfoCenter Search Strings

WOLA in General	
Introduction	`cdat_ola`
Planning to use	`tdat_useola`
Enabling in WAS z/OS	`tdat_enableconnector`
WOLA samples	`cdat_olasamples`
WOLA variables	`cdat_olacustprop`
Custom Properties	`urun_rproperty_custproperties`
Performance	`cdat_perfconsid`
Security	`container_ola_security`
Development Related	
JCA adapter methods	`tdat_connect2wasapp`
EJB development	`tdat_useola_in` (Step 2, Develop an EJB Application)
WOLA APIs	`cdat_olaapis`
CICS Related	
Enabling in CICS	`tdat_enableconnectorcics`
BBOC Transaction	`rdat_cics`
Security	`tdat_security_out`
IMS Related	
Enabling in IMS	`tdat_enableconnectorims`
Security	`cdat_olasecurityimsconsid`

Framework of Approach to WOLA

This is to help you focus on the specifics of your particular use of WOLA. See notes below.

Programming Model Intended ⇨ / ⇩ *External Address Space Type*	**Outbound from WAS z/OS [1]**		**Inbound to WAS z/OS [2]**
CICS	***Link Server*** Note 1	***APIs*** Note 2	Note 3
IMS	***OTMA*** Note 4	***APIs*** Note 5	Note 6
Batch | USS | IMS DL/I Batch	Note 7		Note 8
ALCS	Note 9		

Note 1 ***Outbound CICS using Link Server Task***
- This provides ability to shield target CICS program from WOLA API programming
- Transaction: Provides the support for two phase commit propagation from WAS out to CICS
- Security: Provides ability to assert WAS thread identity into CICS with that ID used for DPL
- Requires the installation of WOLA support in CICS (page 8)
- Requires enabling WOLA support in WAS z/OS server node (page 7)
- See "Outbound to CICS using Link Server Task" on page 9

Note 2 ***Outbound CICS using WOLA APIs***
- You may bypass the Link Server task to achieve greater performance
- It would require at least one program in CICS be written to the outbound APIs
- Transaction: No propagation of transaction from WAS into CICS possible
- Security: No asssertion of WAS identity into CICS
- Requires the installation of the WOLA support in CICS (page 8)
- Requires enabling WOLA support in WAS z/OS server node (page 7)
- See "Outbound to CICS using WOLA APIs" on page 11

Note 3 ***Inbound CICS***
- The Link Server Task plays no role in an inbound exchange
- This requires at least one CICS program be written to the inbound APIs
- Transaction: Provides the ability to assert global transaction into WAS z/OS
- Security: Provides the ability to assert CICS region ID or the application thread ID into WAS z/OS
- Requires the installation of the WOLA support in CICS (page 8)
- Requires enabling WOLA support in WAS z/OS server node (page 7)
- See "CICS inbound to WAS z/OS" on page 12

Note 4 ***Outbound IMS using OTMA***
- This provides the ability to shield target IMS programs from WOLA API programming
- Does *not* require WOLA support installation in IMS (completely transparent to IMS)
- Transaction: Able to assert CM0 or CM1 transaction into IMS at this time
- Security: Able to assert WAS thread ID into IMS (MPP, IFP but not BMP)
- Requires enabling WOLA support in WAS z/OS server node (page 7)
- See "Outbound to IMS using OTMA" on page 16

1 The program in WAS *initiates* the exchange. It is *outbound* with respect to WAS z/OS. This occurs after a valid registration into the WAS z/OS server has been completed.
2 The program in the external address space initiates the exchange. Again, after a valid registration has been completed.

Note 5 ***Outbound IMS using WOLA APIs***
- You may call directly to programs hosting a WOLA service for greater performance
- Requires at least one IMS program to be written to the WOLA outbound APIs
- Transaction: No propagation of transaction from WAS into IMS possible
- Security: No asssertion of WAS identity into IMS
- Requires enabling WOLA support in WAS z/OS server node (page 7)
- Requires enabling WOLA support in IMS (page 15)
- See "Outbound to IMS using WOLA APIs" on page 16

Note 6 ***Inbound IMS***
- Requires at least one IMS program to write to the WOLA APIs
- Transaction: Only SyncOnReturn into WAS z/OS at this time
- Security: Assertion of IMS thread ID into WAS z/OS
- Requires enabling WOLA support in WAS z/OS server node (page 7)
- Requires enabling WOLA support in IMS (page 15)
- See "IMS inbound to WAS z/OS" on page 17

Note 7 ***Outbound batch, USS or IMS batch DL/I***
- Requires batch program to write to the outbound WOLA APIs
- Transaction: No transaction propagation
- Security: No identity assertion
- Requires enabling WOLA support in WAS z/OS server node (page 7)
- Batch process must have STEPLIB access to WOLA native module library

Note 8 ***Inbound batch, USS or IMS batch DL/I***
- Requires batch program to write to the inbound WOLA APIs
- Transaction: No transaction propagation
- Security: No identity assertion
- Requires enabling WOLA support in WAS z/OS server node (page 7)
- Batch process must have STEPLIB access to WOLA native module library

Note 9 ***ALCS***
- See "ALCS and OLA Brochure" under WP101490 Techdoc at ibm.com/support/techdocs

Enabling WOLA in WAS z/OS

Overview

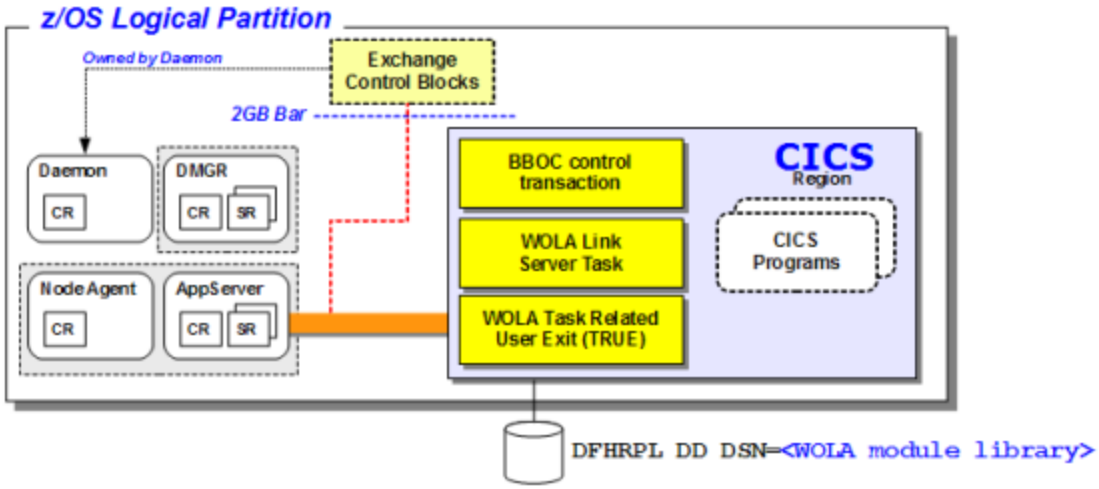
InfoCenter	`tdat_enableconnector`
Essential steps	• Create symlinks from node's configuration file system to the WOLA files in the SMP/E installation file system. `olaInstall.sh` provides this function. • Copy WOLA modules to a preallocated module library for use by external address space. `olaInstall.sh` provides this function. • Install JCA resource adapter `ola.rar` into the node with a connection factory created and assigned a JNDI name. The `olaRar.py` WSADMIN script does this, or you may do it manually. • Create environment variable `WAS_DAEMON_ONLY_enable_adapter` at the cell level with a value of `true`. The `olaRar.py` WSADMIN script does this, or you may do it manually.

Validation

InfoCenter	`cdat_olasamples`
Essential steps	• Install the sample `OLASample1.ear` into a WOLA-enabled server. Make sure it starts and you can access it with the following URL: `http://<host>:<port>/OLA_Sample1_Web/` • Review the `OLACC01` (language: C) or `OLACB06` (language: COBOL) samples. Both are the simplest examples of inbound programming. • Copy the file to a FB 80 source data set. • Edit and modify the values as directed in the comments. • Compile and invoke. It will invoke the sample EJB and receive an echo in return.

WOLA and CICS

Overview

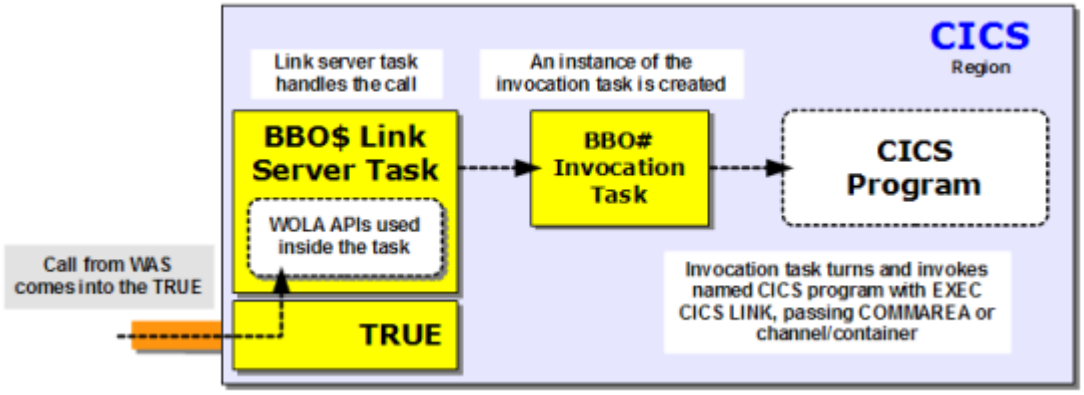


Supplied elements	• A set of WOLA definitions to be installed into the CICS region CSD • Includes a Task Related User Exit (TRUE) • Includes a Link Server Task for outbound support • Includes a BBOC 3270 control transaction • A set of native API modules
Role of the TRUE *required in all cases*	InfoCenter: The adapter is designed to run in a CICS region as a resource manager. In CICS, the Task Related User Exit (TRUE) is the primary vehicle used by resource providers. TRUE support provides the boundary between the CICS application threads and the external resource manager threads. Currently, DB2, WebSphere MQ, and TCPIP sockets execute in CICS using the TRUE support. The optimized local adapters support TRUE.
Role of the Link Server Task *use is optional*	The Link Server Task provides a way to shield your CICS programs from the specifics of WOLA programming. The Link Server Task handles the WOLA calls from WAS and invokes the named CICS program with EXEC CICS LINK. Its use is *optional*, and if used it is only applicable to *outbound* calls.
Role of the BBOC control transaction *use is optional*	The BBOC control transaction provides a convenient way to start and stop the TRUE and the Link Server Task and pass in parameters to modify the behavior of the environment. Its use is *optional.* There are other ways to achieve the same results.

Enabling support in CICS

InfoCenter	`tdat_enableconnectorcics`
Overview	• Install definitions into CSD • Place WOLA module library on DFHRPL • Start the TRUE or update PLTPI and restart CICS

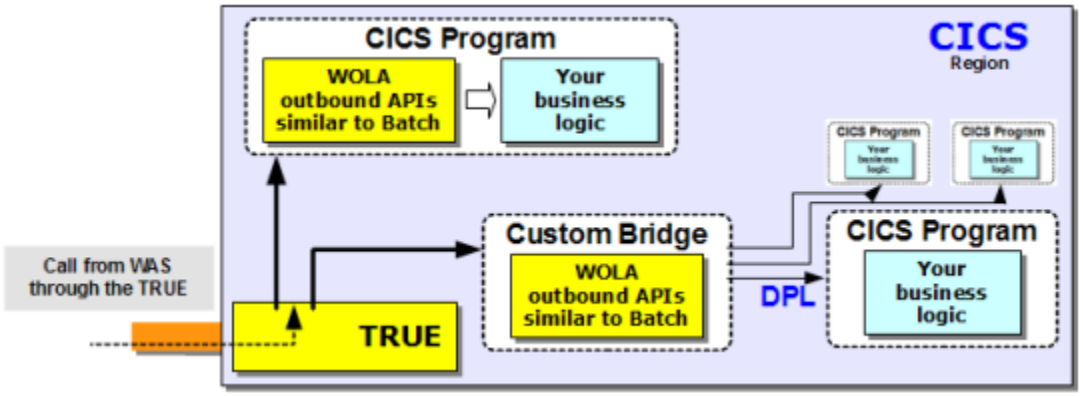
Outbound to CICS using Link Server Task



Overview	The Link Server Task is provided as part of the definitions installed into the CICS CSD. It provides a way to shield your CICS programs from the specifics of WOLA programming. The Link Server Task handles the WOLA calls from WAS and invokes the named CICS program with EXEC CICS LINK. The Link Server Task is named BBO$ by default. The Link Server Invocation Task is named BBO# by default. Both may be changed.
BBOC Commands	BBOC is the 3270 control transaction that is part of the definitions installed into the CSD. InfoCenter search `rdat_cics` provides details of syntax: Start TRUE -- `BBOC START_TRUE <parameters>` Stop TRUE -- `BBOC STOP_TRUE <parameters>` Start Link Server -- `BBOC START_SRVR <parameters>` Stop Link Server -- `BBOC STOP_SRVR <parameters>`
Infocenter *Install WOLA Trans*	`tdat_installwastranscics` **Key:** When CICS security is enabled, the user ID where the `BBOC START_TRUE` and `STOP_TRUE` parameters run must have authority to issue `EXEC CICS ENABLE PROGRAM(BBOATRUE)` and `DISABLE PROGRAM(BBOATRUE) EXITALL`. Messages issued by WebSphere Application Server modules under CICS are routed to the `BBOQ` extra partition transient data queue (TDQ). This is allocated under `DD BBOOUT` in the CICS region.
InfoCenter *WOLA security*	`tdat_security_out` **Key** Ensure that the CICS region is running with security enabled and `EXEC CICS START` checking enabled. Security is enabled at start up with the parameter `SEC=YES`. The `EXEC CICS START` checking is enabled at start up with the parameter `XUSER=YES`. Create a SAF surrogate class that grants the identity that the optimized local adapters Link server is running with the authority to issue `EXEC CICS START TRANSACTION` API and pass the USERID that was propagated to CICS from WebSphere Application Server.

Validation	• The `OLASample1.ear` sample file has a web interface that will allow you to drive an outbound request into CICS. Insure it is installed and started. See "Validation" on page 7. • Use the supplied sample `OLACB01`, which is a CICS COBOL application that will accept a `COMMAREA`. The sample WAS application is written to understand the layout expected by `OLACB01`. • Insure the Link Server Task is started • Invoke the web interface with the URL (see page 7) • Consult the InfoCenter samples page (`cdat_olasamples`) for directions on how to populate the web page to drive the `OLACB01` sample in CICS. • Insure the sample application in CICS has been successfully invoked.
Identity and Transaction Assertion	• With the Link Server Task started and SEC=Y specified, WAS will assert into CICS the identity of the execution thread from WAS. • *Before* WAS z/OS 7.0.0.12 WAS z/OS is limited to SyncOnReturn only for transactions started within the WAS container. • *With* WAS z/OS 7.0.0.12 WAS z/OS may assert its transaction into CICS 4.1 or higher with two phase commit coordination provided by RRS.
Performance Considerations	InfoCenter: `cdat_perfconsid` **BBO# Invocation Task and SEC=Y** The `BBO#` invocation task is what issues the EXEC CICS LINK against the named program in CICS. If security is enabled in CICS and SEC=Y is specified on the `BBOC START_SRVR` command, then each outbound call from WAS will result in the WAS thread ID being propagated into CICS. That results in separate instances of BBO# being invoked. Each with a SAF check for validity of the asserted ID. If you determine the ID used to start the link server task in CICS is sufficient for your security needs, you should consider using the `BBOC START_SRVR` parameters `SEC=N`, `REU=Y`. This allows re-use of the `BBO#` invocation task and less setup/teardown overhead in CICS. **Balancing Concurrent Outbound with Defined BBO# Limits** The number of servant regions for a server times the number of threads in that server determines the maximum *potential* concurrent outbound requests. The WAS environment variable `WAS_DAEMON_ONLY_adapter_max_serv` determines how many outbound services can be engaged concurrently for the WOLA registrations in the cell. The default value is 100. If your application exceeds this number there's a potential for delays and timeout issues. Evaluate the maximum potential against the default 100 and adjust accordingly. Further, the `MNC=` and `MXC=` parameters on `BBOC START_SRVR` determines how many `BBO#` invocation tasks may be active at one time. The default is `MNC=1` and `MXC=10`. If your WAS application overdrives the limit there will be delays invoking the CICS programs. However, setting the maximum unnecessarily high results in wasted CICS resources.

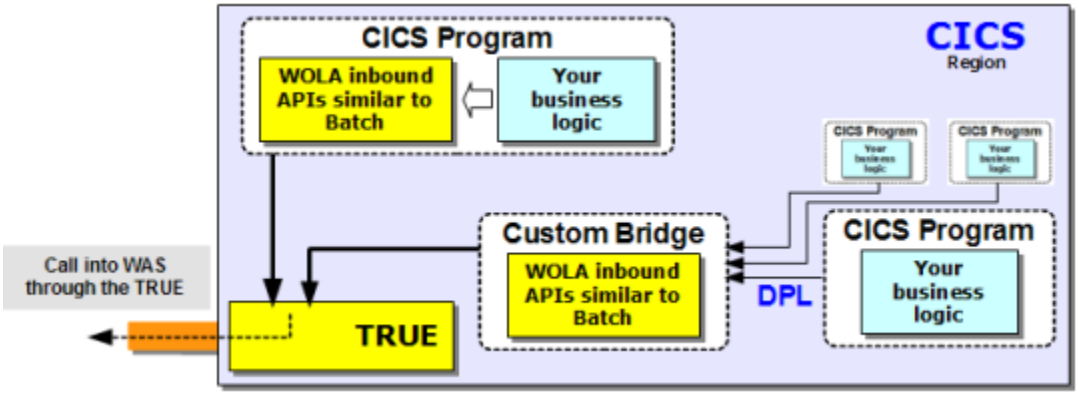
Outbound to CICS using WOLA APIs



Overview	If you have determined not to use the Link Server Task for outbound calls to CICS, the your alternative is to code to the WOLA native APIs and use the "outbound" APIs to "host a service." The Link Server Task is not required for outbound calls to CICS. It provides certain benefits (described next), but it may be bypassed if you wish to achieve maximum efficiency of operations.
TRUE	The TRUE is still required. It must be started for programs in CICS that use WOLA to work.
Registration	Is always required and the external address space always initiates. Without the Link Server task you have two ways to register: use the `BBOC REGISTER` command, or use the `BBOA1REG` API. InfoCenter search `rdat_cics` provides details of `BBOC REGISTER`. InfoCenter search `cdat_olaapis` provides details of `BBOA1REG`
Identity and Transaction Assertion	If you choose to bypass the Link Server Task you lose the ability to assert the WAS thread identity into CICS, and you lose the ability to assert the WAS transaction into CICS. Those require the Link Server Task.
Hosting a Service	This implies having the program in CICS enter a "listen state" so that it can receive and handle the outbound call from WAS. Basic APIs: `BBOA1SRV`, `BBOA1SRP`, `BBOA1CNR` Primitives: `BBOA1RCA`, `BBOA1RCS`, `BBOA1CNG`, `BBOA1CNR`, `BBOA1GET` and `BBOA1SRX`.
Samples	The `OLACB03` sample described at InfoCenter `cdat_olasamples` provides an illustration of the simplest form of "hosting a service." `OLACB04` and `05` provide illustrations of more advanced usages. The "The WOLA Native APIs ... a COBOL Primer" under the Techdoc WP101490 provides many examples of using the outbound APIs. The examples there show batch, but they may be used equally well in CICS. Be sure to install the OLASample1.ear application into the WAS server.
Performance Considerations	**Balancing Concurrent Outbound with Defined Connection Limits** The number of servant regions for a server times the number of threads in that server determines the maximum *potential* concurrent outbound requests. The WAS environment variable `WAS_DAEMON_ONLY_adapter_max_serv` determines how many outbound services can be engaged concurrently for the WOLA registrations in the cell. The default value is 100.

	If your application exceeds this number there's a potential for delays and timeout issues. Evaluate the maximum potential against the default 100 and adjust accordingly. Further, the `MNC=` and `MXC=` parameters on `BBOC REGISTER` (or the equivalent parameter on the `BBOA1REG` API) determines how many concurrent connections may be open. The default is `MNC=1` and `MXC=10`. If your WAS application overdrives the limit there will be delays invoking the CICS programs. However, setting the maximum unnecessarily high results in wasted CICS resources. **Synchronous vs. Asynchronous APIs** The "Basic" APIs provide ease of use but limit the flexibility of operations. Specifically, they assume *synchronous* control: for example, on the `BBOA1SRP` (send response) API program control is not returned to your program until WAS invokes again. The thread and the WOLA connection is tied up during that time. The "advanced" (or "primitive") APIs allow you to operate *asynchronously*. That allows your program to receive program control immediately. That allows your program to go off and do other work, or pull a request off another inbound connection. This allows for greater utilization of resources and in turn greater performance. The "The WOLA Native APIs ... a COBOL Primer" under the Techdoc WP101490 shows how outbound asynchronous operations work.

CICS inbound to WAS z/OS

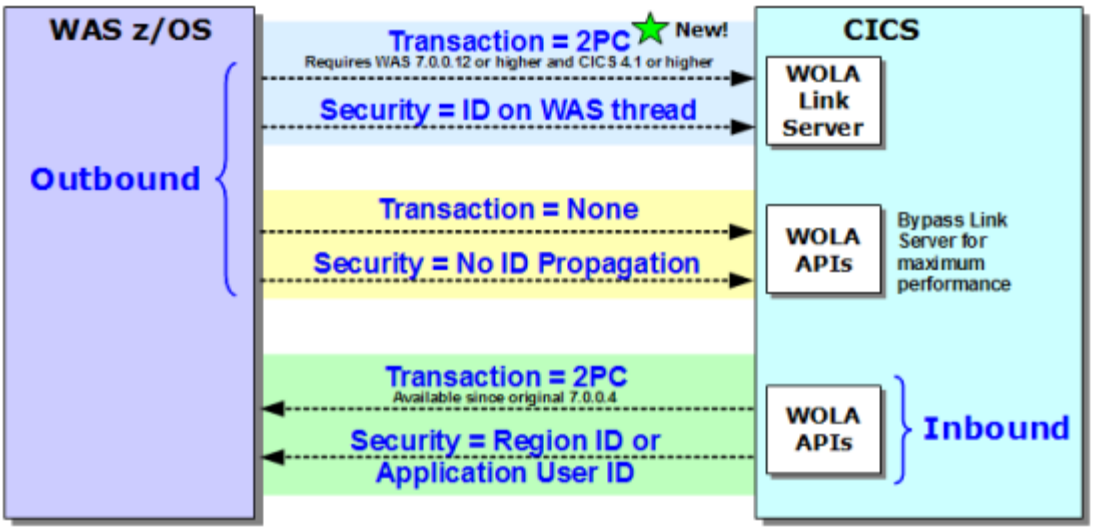


Overview	Inbound to WAS z/OS by definition implies no Link Server Task. By definition it implies at least one application written to use the WOLA native APIs. Other CICS applications may use DPL to invoke that CICS program and gain access to WOLA.
TRUE	The TRUE is still required. It must be started for programs in CICS that use WOLA to work.
Registration	Is always required and the external address space always initiates. Without the Link Server task you have two ways to register: use the `BBOC REGISTER` command, or use the `BBOA1REG` API. InfoCenter search `rdat_cics` provides details of `BBOC REGISTER`. InfoCenter search `cdat_olaapis` provides details of `BBOA1REG`

Identity and Transaction Assertion	With inbound to WAS z/OS you have the ability to assert the CICS region ID or the CICS application ID, and assert the CICS transaction into WAS z/OS with full two phase commit processing coordinated by RRS. This has been the case since the inception of WOLA with 7.0.0.4.
Enabling Identity Assertion	`tdat_security_in` • Grant the CICS ID being asserted into WAS `READ` to the WAS server's `CBIND` class profiles. There should be two profiles: `CB.<cluster_name>` and `CB.BIND.<cluster_name>`.[3] The first controls access to the CR, the second controls access to Java EE applications in the server. • Create a WAS environment variable scoped at the cell level with the following name and value: `ola_cicsuser_identity_propagate = 1` • Make sure SEC=Y is specified on the `BBOC REGISTER` command or the proper registration flag is set if `BBOA1REG` is used.
Enabling Transaction Assertion	Assertion of CICS transaction into WAS z/OS requires `TXN=Y` on registration.
Samples	The `OLACB06` sample described at InfoCenter `cdat_olasamples` provides an illustration of the simplest form of inbound invocation. `OLACB04` and `05` provide illustrations of more advanced usages. The "The WOLA Native APIs ... a COBOL Primer" under the Techdoc WP101490 provides many examples of using the inbound APIs. The examples there show batch, but they may be used equally well in CICS. Be sure to install the OLASample1.ear application into the WAS server.
Performance Considerations	**Balancing Concurrent Inbound with Defined Connection Limits** The `BBOA1REG` API specifies the minimum and maximum connections permitted across the registration. The usual default for maximum is 10. The WAS environment variable `WAS_DAEMON_ONLY_adapter_max_conn` determines the maximum permitted into WAS. The default is 100. If the BBOA1REG API specifies more than 100 there's a chance your external program will receive errors on the "get connection" activity. That may lead to retries and lost performance. **Synchronous vs. Asynchronous APIs** The "Basic" APIs provide ease of use but limit the flexibility of operations. Specifically, they assume *synchronous* control: for example, on the `BBOA1INV` (invoke) API program control is not returned to your program until WAS returns the response. The thread and the WOLA connection is tied up during that time. The "advanced" (or "primitive") APIs allow you to operate *asynchronously*. That allows your program to receive program control immediately. That allows your program to go off and do other work. This allows for greater utilization of resources and in turn greater performance. The "The WOLA Native APIs ... a COBOL Primer" under the Techdoc WP101490 shows how outbound asynchronous operations work.

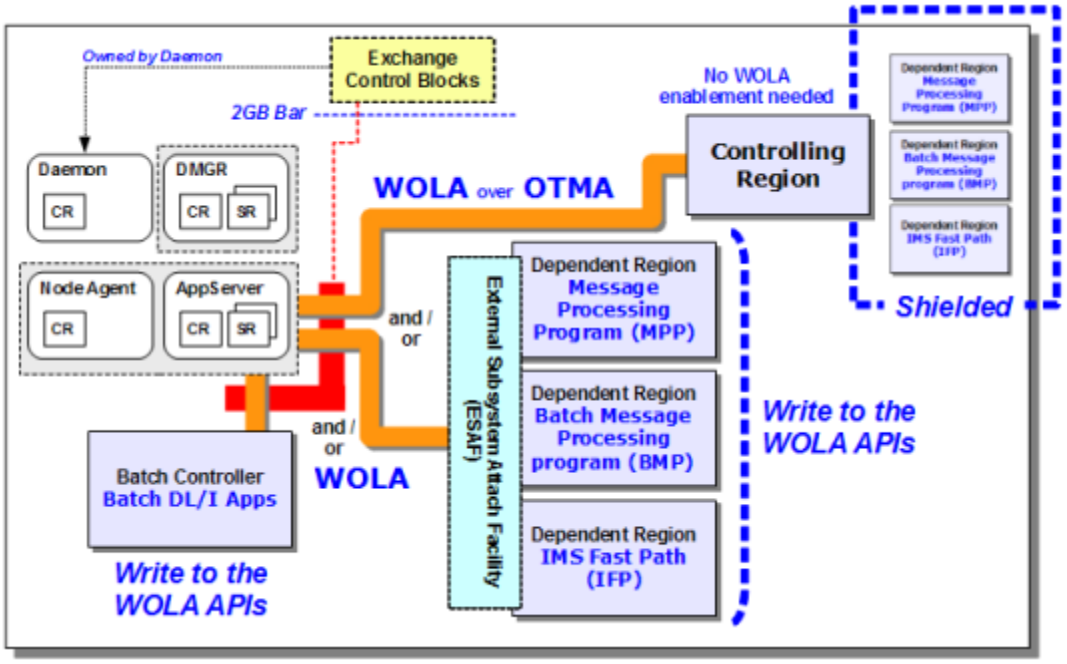
3 If the server is not clustered, then `<cluster_name>` referes to the "cluster transition name" for the server.

Summary of CICS Support



WOLA and IMS

Overview



Important Note	• WAS z/OS must be at level 7.0.0.12 or higher • The external modules you use must be from 7.0.0.12 or higher • The JCA resource adapter (ola.rar) must be from 7.0.0.12 or higher
IMS Samples	7.0.0.12 shipped without a small handful of IMS-specific sample programs. APAR PM21407 is currently scheduled to be included in 7.0.0.15. Fix pack availability information can be found at the following URL: `www.ibm.com/support/docview.wss?rs=404&uid=swg27006970`
Supplied elements	• A set WOLA native API modules • An IMS external subsystem module
Outbound from WAS	Two options exist: • OTMA, which provides complete shielding of applications and IMS. This would be through the IMS controlling region using the OTMA call interface. The WAS JCA resource adapter has been updated with OTMA-specific methods to accomodate this outbound call through OTMA. • WOLA APIs, which provides greater performance and control. This implies at least one IMS program written to the APIs.
Inbound to WAS	Inbound from IMS dependent regions is done using the supplied WOLA stub and ESAF support. Programs in the IMS dependent region use the normal WOLA APIs. They will recognize the IMS environment and invoke the ESAF interface.
Batch DL/I	See "WOLA and Batch (Including IMS DL/I)" on page 19.

Enabling support in IMS

InfoCenter	`tdat_enableconnectorims`
If OTMA	• IMS does not need to be updated to be aware of the WOLA modules. • IMS security definitions *may* need modification. See specifics under the outbound and inbound sections of IMS in this document.

If ESAF	• APF authorize the WOLA native modules • Update external subsystem proclib member and reference the WOLA ESAF support • Add the WOLA native module library to the IMS dependent region STEPLIB concatenation
If Batch	• APF authorize the WOLA native modules • Add the WOLA native module library to the batch region's STEPLIB concatenation

Outbound to IMS using OTMA

Overview	In this mode WAS z/OS invokes the OTMA call interface of the named IMS controller region. There is no specific WOLA awareness inside of IMS. To IMS it appears like any other OTMA call. To use this feature you *must* use the `ola.rar` resource adapter that comes with 7.0.0.12 or higher. That level of the resource adapter has OTMA-specific methods that facilitate the call to the OTMA call interface.
Programming in WAS	`tdat_connect2wasapp`
Identity Assertion into IMS	`cdat_olasecurityimsconsid` Note: This applys to MPP and IFP, but *not* BMP dependent regions. • Configure WAS z/OS with SyncToOS Thread enabled • Ensure that the `OTMASE` parameter for the target IMS environment is set to `F, FULL`
Transaction Assertion into IMS	`cdat_callexisttrans` Two-phase commit starting with 8.0.0.5. Prior to that Synclevel NONE or CONFIRM. Grant the thread-level user ID effected in the WebSphere Application Server application `READ` access to the OTMA resource `IMSXCF.OTMACI` in the `FACILITY` SAF class.
Samples	`cdat_olasamples` The `OTMAINIT` sample JCL is provided to show how to start the IMS OTMA callable interface SVCs on your system. The rest is done in the WAS program by calling the OTMA methods on the JCA resource adapter.

Outbound to IMS using WOLA APIs

Overview	In this mode the program in WAS z/OS interacts with the WOLA outbound APIs being hosted in the IMS dependent region. The API modules *must* be at the 7.0.0.12 level or higher. See InfoCenter `cdat_olaapis` for a reference of the APIs. See the "Primer" document in WP101490 Techdoc for examples of outbound API usage.
Registration	Is always required and the external address space always initiates. For IMS that means using the `BBOA1REG` API. InfoCenter search `cdat_olaapis` provides details of `BBOA1REG`
Identity Assertion into IMS	None
Transaction Assertion into IMS	None. Transactional control is IMS application dependent.

Performance Considerations	**Balancing Concurrent Outbound with Defined Connection Limits** The number of servant regions for a server times the number of threads in that server determines the maximum *potential* concurrent outbound requests. The WAS environment variable `WAS_DAEMON_ONLY_adapter_max_serv` determines how many outbound services can be engaged concurrently for the WOLA registrations in the cell. The default value is 100. If your application exceeds this number there's a potential for delays and timeout issues. Evaluate the maximum potential against the default 100 and adjust accordingly. Further, the `minconn` and `maxconn` parameters on the `BBOA1REG` API determines how many concurrent connections may be open. The default is `minnconn=1` and `maxconn=10`. If your WAS application overdrives the limit there will be delays invoking the IMS programs. However, setting the maximum unnecessarily high results in wasted IMS resources. **Synchronous vs. Asynchronous APIs** The "Basic" APIs provide ease of use but limit the flexibility of operations. Specifically, they assume *synchronous* control: for example, on the `BBOA1SRP` (send response) API program control is not returned to your program until WAS invokes again. The thread and the WOLA connection is tied up during that time. The "advanced" (or "primitive") APIs allow you to operate *asynchronously*. That allows your program to receive program control immediately. That allows your program to go off and do other work, or pull a request off another inbound connection. This allows for greater utilization of resources and in turn greater performance. The "The WOLA Native APIs ... a COBOL Primer" under the Techdoc WP101490 shows how outbound asynchronous operations work.

IMS inbound to WAS z/OS

Overview	In this mode the program in IMS must write to the WOLA native inbound APIs. The WOLA API modules *must* be at the 7.0.0.12 level or higher. That is the level of modules that is able to recognize they're operating in IMS and use the IMS ESAF.
Registration	Is always required and the external address space always initiates. For IMS that means using the `BBOA1REG` API. InfoCenter search `cdat_olaapis` provides details of `BBOA1REG`
Identity Assertion into WAS	`tdat_security_in` • Grant the IMS ID being asserted into WAS `READ` to the WAS server's `CBIND` class profiles. There should be two profiles: `CB.<cluster_name>` and `CB.BIND.<cluster_name>`.[4] The first controls access to the CR, the second controls access to Java EE applications in the server.
Transaction Assertion into WAS	With WAS 8.0.0.4 IMS Dependent Region applications may assert RRS transaction context into WAS. Prior to 8.0.0.4 only Sync on Return. If 8.0.0.4 then two requirements to assert RRS context into WAS: • Set WAS environment `adapter_rrs_propagate_context = 1` • IMS Control Region must be running with `RRS=Y` • Applications must set "Transaction Supported" flag on register API
Performance Considerations	**Balancing Concurrent Inbound with Defined Connection Limits** The `BBOA1REG` API specifies the minimum and maximum connections

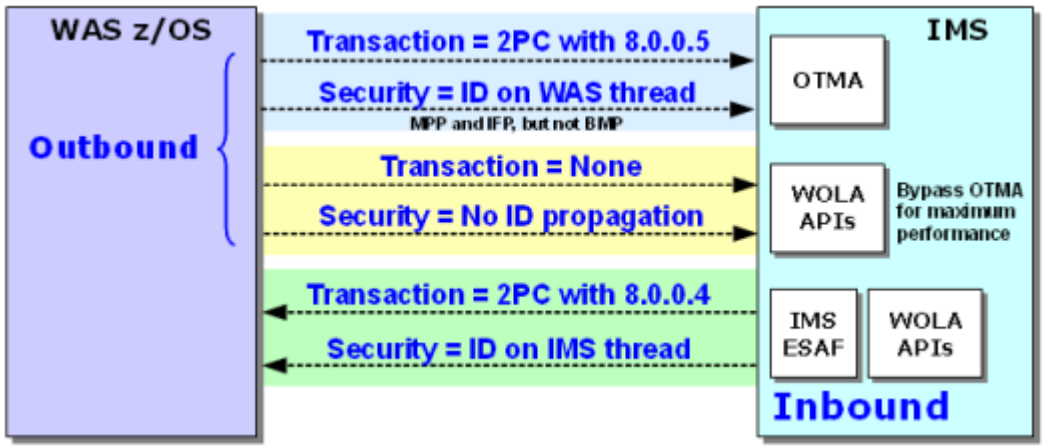
4 If the server is not clustered, then `<cluster_name>` referes to the "cluster transition name" for the server.

	permitted across the registration. The usual default for maximum is 10. The WAS environment variable `WAS_DAEMON_ONLY_adapter_max_conn` determines the maximum permitted into WAS. The default is 100. If the `BBOA1REG` API specifies more than 100 there's a chance your external program will receive errors on the "get connection" activity. That may lead to retries and lost performance. **Synchronous vs. Asynchronous APIs** The "Basic" APIs provide ease of use but limit the flexibility of operations. Specifically, they assume *synchronous* control: for example, on the `BBOA1INV` (invoke) API program control is not returned to your program until WAS returns the response. The thread and the WOLA connection is tied up during that time. The "advanced" (or "primitive") APIs allow you to operate *asynchronously*. That allows your program to receive program control immediately. That allows your program to go off and do other work. This allows for greater utilization of resources and in turn greater performance. The "The WOLA Native APIs ... a COBOL Primer" under the Techdoc WP101490 shows how outbound asynchronous operations work.

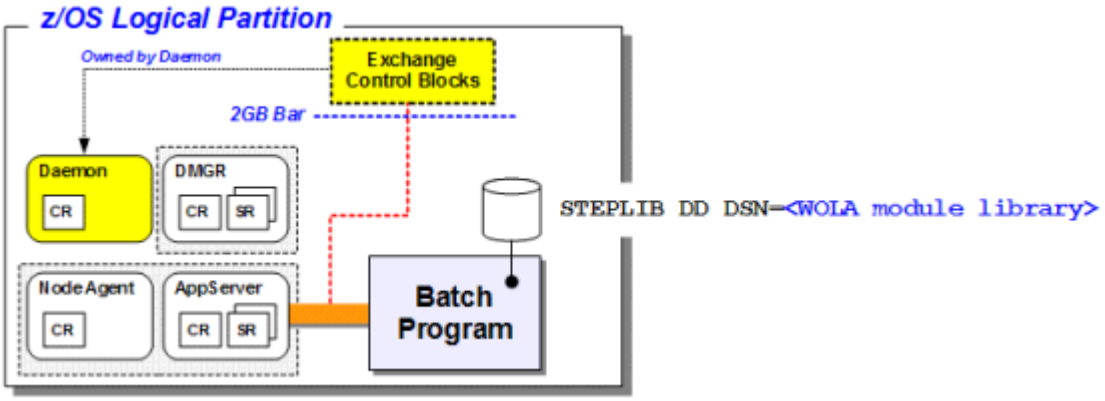
IMS DL/I Batch

See "WOLA and Batch (Including IMS DL/I)" on page 19. The concepts and specifics are identical.

Summary IMS Support



WOLA and Batch (Including IMS DL/I)



Overview

Overview	Using WOLA with batch implies writing to the WOLA APIs, both inbound and outbound.

Enabling support in batch

Overview	This is simply a matter of providing the WOLA native API module library to the `STEPLIB` concatenation of the batch program, including the

Outbound to batch

Overview	This requires use of the WOLA outbound APIs.
Registration	Is always required and the external address space always initiates. Without the Link Server task you have two ways to register: use the `BBOC REGISTER` command, or use the `BBOA1REG` API. InfoCenter search `cdat_olaapis` provides details of `BBOA1REG`
Identity and Transaction Assertion	Neither is supported with batch.
Hosting a Service	This implies having the batch program enter a "listen state" so that it can receive and handle the outbound call from WAS. Basic APIs: `BBOA1SRV`, `BBOA1SRP`, `BBOA1CNR` Primitives: `BBOA1RCA`, `BBOA1RCS`, `BBOA1CNG`, `BBOA1CNR`, `BBOA1GET` and `BBOA1SRX`.
Samples	The `OLACB03` sample described at InfoCenter `cdat_olasamples` provides an illustration of the simplest form of "hosting a service." `OLACB04` and `05` provide illustrations of more advanced usages. The "The WOLA Native APIs ... a COBOL Primer" under the Techdoc WP101490 provides many examples of using the outbound APIs. The examples there show batch, but they may be used equally well in CICS. Be sure to install the `OLASample1.ear` application into the WAS server.
Performance Considerations	**Balancing Concurrent Outbound with Defined Connection Limits** The number of servant regions for a server times the number of threads in that server determines the maximum *potential* concurrent outbound requests. The WAS environment variable `WAS_DAEMON_ONLY_adapter_max_serv` determines how many outbound services can be engaged concurrently for the WOLA registrations in the cell. The default value is 100. If your application exceeds this number there's a potential for delays and

	timeout issues. Evaluate the maximum potential against the default 100 and adjust accordingly. Further, the `minconn` and `maxconn` parameters on the `BBOA1REG` API determines how many concurrent connections may be open. The default is `minnconn=1` and `maxconn=10`. If your WAS application overdrives the limit there will be delays invoking the batch programs. **Synchronous vs. Asynchronous APIs** The "Basic" APIs provide ease of use but limit the flexibility of operations. Specifically, they assume *synchronous* control: for example, on the `BBOA1SRP` (send response) API program control is not returned to your program until WAS invokes again. The thread and the WOLA connection is tied up during that time. The "advanced" (or "primitive") APIs allow you to operate *asynchronously*. That allows your program to receive program control immediately. That allows your program to go off and do other work, or pull a request off another inbound connection. This allows for greater utilization of resources and in turn greater performance. The "The WOLA Native APIs ... a COBOL Primer" under the Techdoc WP101490 shows how outbound asynchronous operations work.

Inbound from batch

Overview	This requires use of the WOLA inbound APIs.
Registration	Is always required and the external address space always initiates. Without the Link Server task you have two ways to register: use the `BBOC REGISTER` command, or use the `BBOA1REG` API. InfoCenter search `cdat_olaapis` provides details of `BBOA1REG`
Identity and Transaction Assertion	Transaction propagation inbound to WAS is not supported. The invoked target EJB will run under the effective ID of the batch job.
Samples	The `OLACB06` sample described at InfoCenter `cdat_olasamples` provides an illustration of the simplest form of inbound invocation. `OLACB04` and `05` provide illustrations of more advanced usages. The "The WOLA Native APIs ... a COBOL Primer" under the Techdoc WP101490 provides many examples of using the inbound APIs. The examples there show batch, but they may be used equally well in CICS. Be sure to install the OLASample1.ear application into the WAS server.
Performance Considerations	**Balancing Concurrent Inbound with Defined Connection Limits** The `BBOA1REG` API specifies the minimum and maximum connections permitted across the registration. The usual default for maximum is 10. The WAS environment variable `WAS_DAEMON_ONLY_adapter_max_conn` determines the maximum permitted into WAS. The default is 100. If the `BBOA1REG` API specifies more than 100 there's a chance your external program will receive errors on the "get connection" activity. That may lead to retries and lost performance. **Synchronous vs. Asynchronous APIs** The "Basic" APIs provide ease of use but limit the flexibility of operations. Specifically, they assume *synchronous* control: for example, on the `BBOA1INV` (invoke) API program control is not returned to your program until WAS returns the response. The thread and the WOLA connection is tied up during that time. The "advanced" (or "primitive") APIs allow you to operate *asynchronously*.

	That allows your program to receive program control immediately. That allows your program to go off and do other work. This allows for greater utilization of resources and in turn greater performance. The "The WOLA Native APIs ... a COBOL Primer" under the Techdoc WP101490 shows how outbound asynchronous operations work.

Document Change History

Check the date in the footer of the document for the version of the document.

September 12, 2010	Original document.
February 3, 2012	Updated the information regarding security assertion into WAS z/OS on inbound from *batch* processing. The processing is as follows: • WOLA pulls the ID from the ACEE on the TCB at the time of the WOLA Register and validates it has access to the CBIND SAF class. This is the access check for the server. • WOLA pulls the ID from the ACEE on the TCB at the time of the WOLA Send Request/ or Invoke API call and asserts it into the EJB container. So the EJB runs under the effective ID of the batch job user. Security identity assertion for other scenarios is as originally documented.
September 11, 2012	Updated IMS section to reflect 8.0.0.4 update that allows transaction assertion into WAS from IMS for Dependent Region applications.
November 12, 2012	Updated IMS section to reflect 8.0.0.5 update that allows transaction assertion into IMS over OTMA. Updated CICS section to reflect 8.0.0.5 that allows enhanced CICS channels and containers support.

End of WP101490