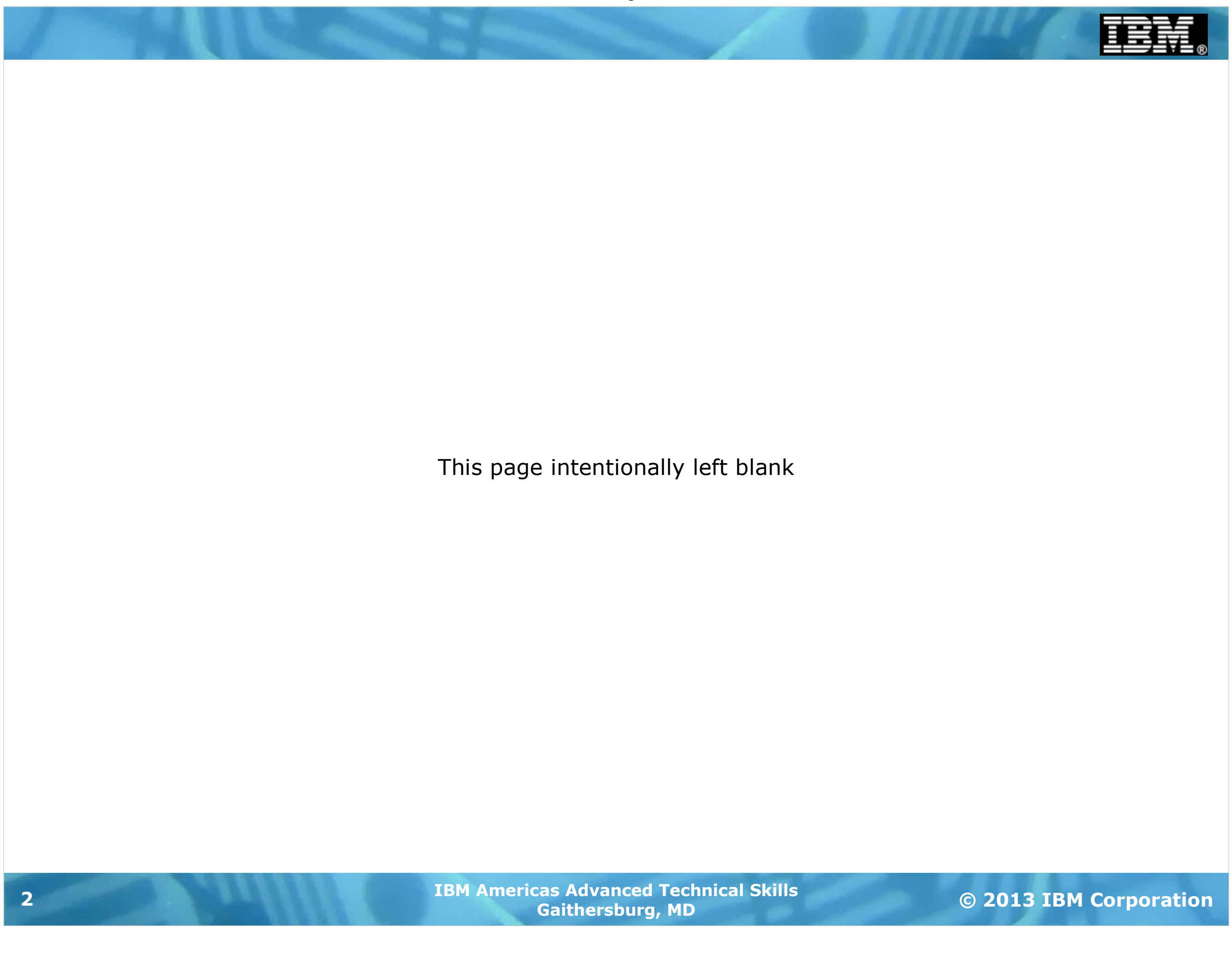
WebSphere Application Server V8.5 for z/OS

WBSR85

Unit 4 - Accessing z/OS Data


IBM
This page intentionally left blank
2
IBM Americas Advanced Technical Skills
Gaithersburg, MD
© 2013 IBM Corporation

High Level of Data Access Approaches with WAS z/OS

There are five categories of data access approaches. We'll cover three in this unit and one in the next unit. The fifth we'll leave to other workshops:

Data abstraction and open standards ...

IBM Americas Advanced Technical Skills
Gaithersburg, MD
© 2013 IBM Corporation

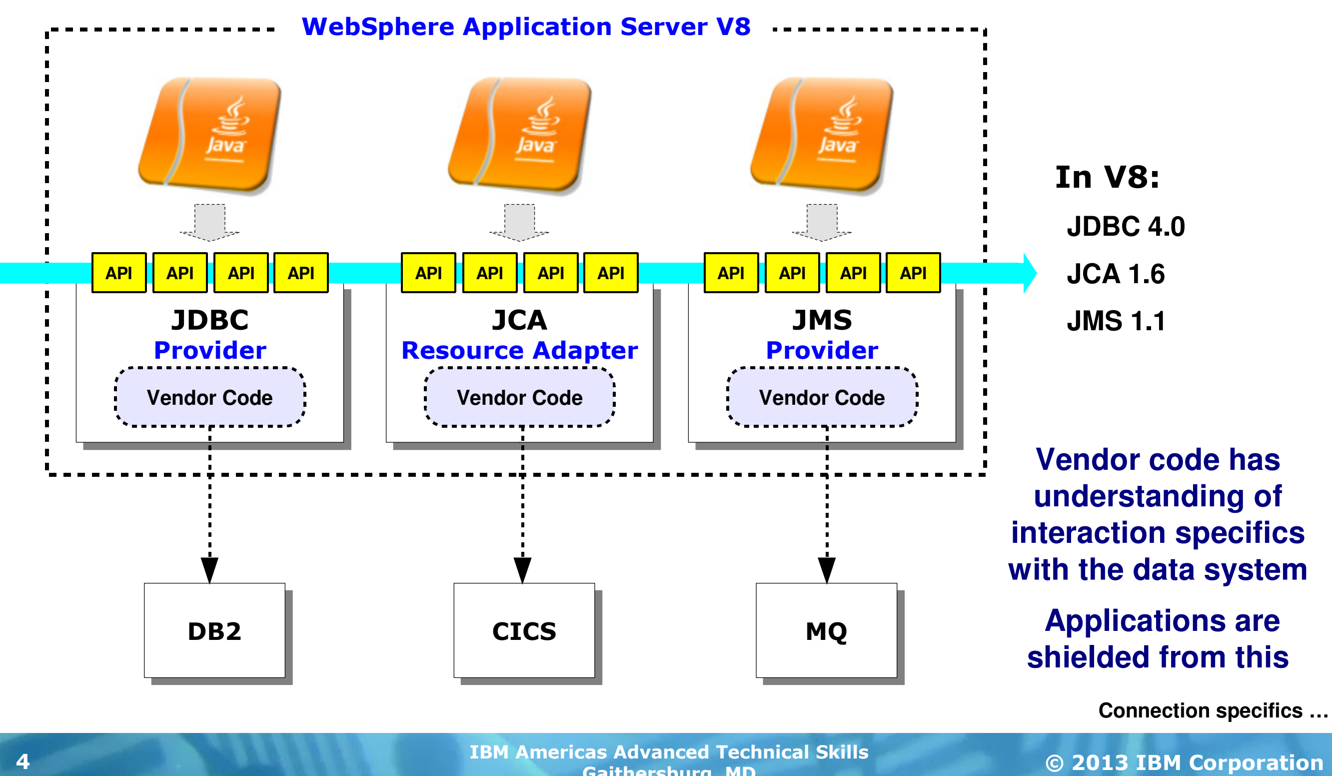
There are several categories of approaches to accessing data that resides on z/OS. For this presentation we'll focus on the first three -- relational data, non-relational data and messaging. We'll cover the memory exchange adapters in the next unit.

We won't touch on the network protocol mechanisms such as SOAP or Web 2.0 technologies, but that's not to say they're not very useful and very much a part of z/OS. All the major data systems on z/OS participate in Web Services. It's a significant component of the overall story of accessing data. We're not delving into the topic for two reasons -- (1) the focus is much more on the application and the configuration of the data subsystem (for example, how to configure a CICS region to support web services, and (2) from a WAS z/OS perspective there's not as much exploitation of z/OS directly.

So ... an interesting and useful topic, but beyond the scope of this workshop.

Data Abstraction Behind Open Standard Interfaces

The data access approaches all share a common theme -- hiding data subsystem specifics behind standard APIs, with installable code to provide lower level access:



In V8:

JDBC 4.0

JCA 1.6

JMS 1.1

Vendor code has understanding of interaction specifics with the data system

Applications are shielded from this

Connection specifics ...

IBM Americas Advanced Technical Skills
Gaithersburg, MD
© 2013 IBM Corporation

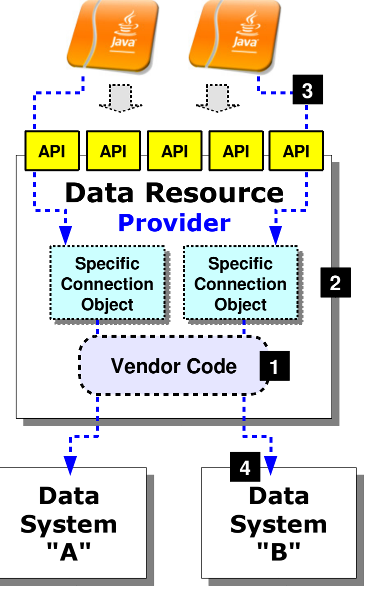
There's a common theme that runs across data access for the three categories we're focusing on ... and that is the theme of "hiding" the data access specifics behind an open standard application interface. The purpose for this is so application developers do not need to understand the nuances of CICS or DB2 access. Their focus is on the open standard application interfaces. Behind those interfaces there is vendor-written code that takes care of the lower-level specifics.

For WAS z/OS V8 -- and really WAS on all platforms since at the API layer "WAS is WAS" across all platforms -- the level of the specifications supported are JDBC 4.0, JCA 1.6 and JMS 1.1.

A good deal of this unit will focus on how the vendor-written code -- the IBM-written code in our case -- is installed and configured into the WAS z/OS runtime, and how applications then make use of the support provided.

Another Common Theme - Connection Specifics

The "Provider" supplies the vendor code that understands how to work with the data system. Another component is needed - something to tell which data system to talk to:



1. **The provider supplies the code that interacts with the specific data resource, as well as a framework for creating specific connection objects**
2. **The specific connection objects provide details about which data system to connect to and any name, port or other details required**
3. **Application do a JNDI lookup of the specific connection object**
4. **Then using that connection they access the data system named in the specific connection object**

IBM Data System	Name used to refer to the specific connection object in WAS
DB2	**"Data Source"**
CICS	**"Connection Factory"**
JMS	**"Connection Factory"**

Different names ... same concept

Local vs. remote connections ...

IBM Americas Advanced Technical Skills
Gaithersburg, MD

© 2013 IBM Corporation

There's another common theme that runs across all three categories of data access ... a definition within WAS that provides specifics about the connection to the data resource.

The "provider" is simply the vendor-supplied code that provides the mechanism for the connection, but it does not say *which* DB2 instance to connect to, or *which* CICS region to use. You would not want to hard-code that kind of information into the applications themselves, otherwise if the DB2 instance or CICS region information changes you'd have to crawl through all the affected applications and change them.

So what the architecture provides is a definition that holds the connection specifics. Applications don't code the connection specifics, they simply refer to the connection object and WAS resolves it to the actual data resource.

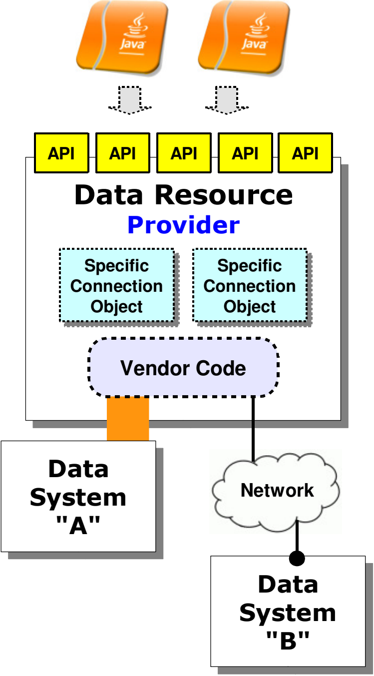
Note: in concept this is just like the role DD cards play in JCL. Applications reference the DD name, and JCL resolves that reference to the actual name of the data set.

This "specific connection object" concept has slightly different names depending on the backend data resource we're talking about. For DB2 it's called a "data source," and for CICS and MQ it's called a "connection factory." The names are different, the concept is the same.

For any given WAS environment you're very likely to have multiple such data sources or connection factories, depending on the variety of backend data resources you have. Applications are connected to the data source or connection factory it needs. WAS takes care of the connection from there.

z/OS Theme - Choice of Local or Remote Connection

On z/OS the specific connection details allow for two types of connectivity -- local, which is a cross-memory connection, or remote, which uses TCP/IP:



Cross-Memory Connection

Uses z/OS cross memory services to access the data system:

- **DB2 - Type 2 JDBC**
- **CICS - EXCI**
- **MQ - Bindings Mode**
- **Involves Java *and native code execution***
  Which means configuration will involve pointing to native libraries

Network Connection

Accesses the data system via the network and an exchange protocol mapped on TCP/IP:

- **DB2 - Type 4 JDBC**
- **CICS - CTG Gateway or IPIC**
- **MQ - Client Mode**
- **Involves Java execution only**

Relational ...

6 IBM Americas Advanced Technical Skills
Gaithersburg, MD

© 2013 IBM Corporation

On z/OS we have another theme to consider, and that's how the access from WAS to the data resource is made. The two options are cross-memory or network. When the WAS z/OS server and the data resource are on the same LPAR you may choose between them; when the WAS z/OS server is on a different LPAR from the data resource then a network connection is required.

The cross-memory mechanism carry different names depending on the data source. For instance, DB2 cross-memory connections are commonly called "Type 2 connections," which is a reference to the JDBC specification for local native-code connection to the relational database server. For CICS the mechanism that is used is the External Call Interface (EXCI) facility of CICS. For MQ the term used is "bindings mode."

All those cross-memory connections make use of the direct-connection facilities of the target data resource. As such, it's necessary to incorporate the native code (that is, non-Java) used to make this low-level connection to the data resource. DB2, CICS and MQ all supply such native libraries. This implies a configuration step to tell WebSphere about the location of these native libraries. We'll show you how that's done for DB2, CICS and MQ as we get to those sections in this unit.

A network connection makes use of the TCP/IP network to pass data requests from WAS into the data resource. Unlike the cross-memory connections, these do not require the native libraries. These are all-Java implementations. What's needed is information about the host and listener port for the target data resource.

Relational Data Access

JDBC

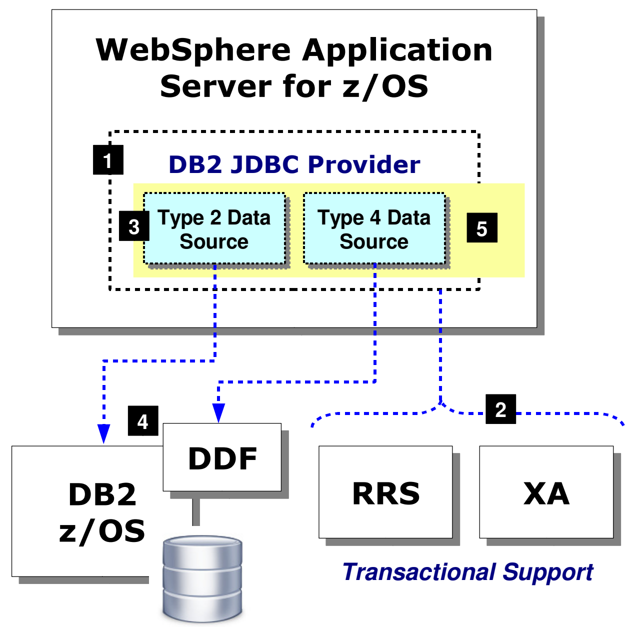
IBM Americas Advanced Technical Skills
Gaithersburg, MD

© 2013 IBM Corporation

Our first stop on this journey is relational data base access ... which uses JDBC.

Framework of This Section's Discussion

There are five major areas of discussion within this JDBC sub-section of the unit:



1. **Configuration of Provider**
   Where driver is located, Admin Console panels used to install and configure
2. **Transaction support based on "Implementation Type" selected**
   1 phase or 2 phase, RRS or XA Partner Logs
3. **Configuration of Data Sources**
   Admin Console panels used to configure
4. **Implications of Type 2 v Type 4**
   Specifically, identity assertion
5. **The new failover capabilities of WAS V8**
   Ability to automatically fail over and fail back

JDBC provider …

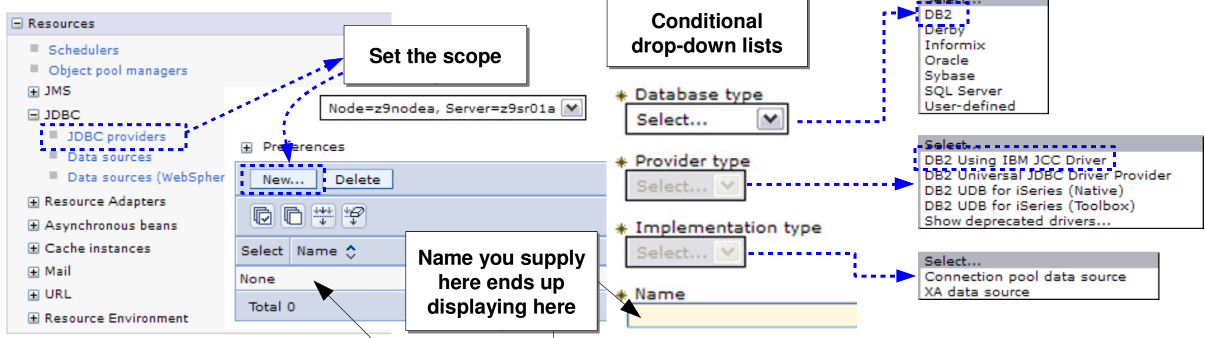
© 2013 IBM Corporation

There's a fair amount to discuss here, which is why we start off with an outline of sorts for the discussion that follows. We have five things to touch upon when discussing JDBC access to DB2 z/OS:

1. How the JDBC provider is configured inside the WAS runtime environment. As you'll soon see, this is done through the WAS administrative console by telling WAS where the DB2 installation path is located.
2. We need to have a brief discussion of the transaction support because how you configure the JDBC provider will influence whether it's capable of supporting two-phase commit processing.
3. The data sources provide WAS information about which DB2 instance to connect to. Again, this is done through the WAS administrative console. This is where you'll specify Type 2 (cross-memory) or Type 4 (network) connectivity.
4. The question of security and identity assertion comes up frequently when speaking about data access, and how identity assertion is accomplished differs between Type 2 and Type 4.
5. Finally, we wish to explore a new function of WAS Version 8, which provide a way to identify when a primary data resource is lost and failover to a defined alternate data source. This includes the ability to check for the return of the primary and failback. And with WAS z/OS there's also a new MODIFY command that influences this failover and failback.

Configuring the JDBC Provider for DB2 z/OS

This is a relatively simple process involving a few panels ... but some interesting implications are surfaced by the choices made:



DB2 Using IBM JCC Driver

Contains the JDBC 4.0 specification support

Backwards compatible so applications written to JDBC 3.0 will work with this driver

For WAS z/OS V7 and later this is recommended provider for IBM z/OS DB2, provided your DB2 has the db2jcc4.jar file (which indicates this driver is present).

DB2 Universal JDBC Driver Provider

JDBC 3.0 specification support

Connection Pool Data Source

If data source is Type 4, then 1 Phase Commit only

If data source is Type 2, then 2 Phase Commit with RRS

XA Data Source

If data source is Type 4, then 2 Phase Commit with XA

Type 2 data sources are not supported under this implementation type

Transaction support ...

 © 2013 IBM Corporation

The JDBC Provider definition is what provides WAS information about where the JDBC drivers are located. With that information WAS may load those drivers. Those drivers are what provide the open standard JDBC interface to the applications, and perform the lower-level connection to the DB2 instance.

In the Admin Console the JDBC provider information is accessed through the navigation tree under Resources, then JDBC Providers. To define a new JDBC Provider you click on the new button which brings up a set of drop down lists where you begin your configuration. The drop down lists are conditional ... that is, what you select for the first drop down list affects what will appear in the next. In our example we're going to show you DB2, so the first selection under "Database type" will be DB2.

The "Provider type" drop down then has a set of drivers to select from. The difference between the first two, which have very similar names, is the level of the JDBC specification support provided. Which you choose is really a factor of what JDBC driver your copy of DB2 supplies. Look in your DB2 `/classes` directory and choose the Provider Type accordingly:

`db2jcc.jar` only -- DB2 Universal JDBC Driver Provider

`db2jcc.jar` *and* `db2jcc4.jar` -- DB2 Using IBM JCC Driver

We're using DB2 z/OS 9.1 for the workshop lab and it includes the `db2jcc4.jar` file.

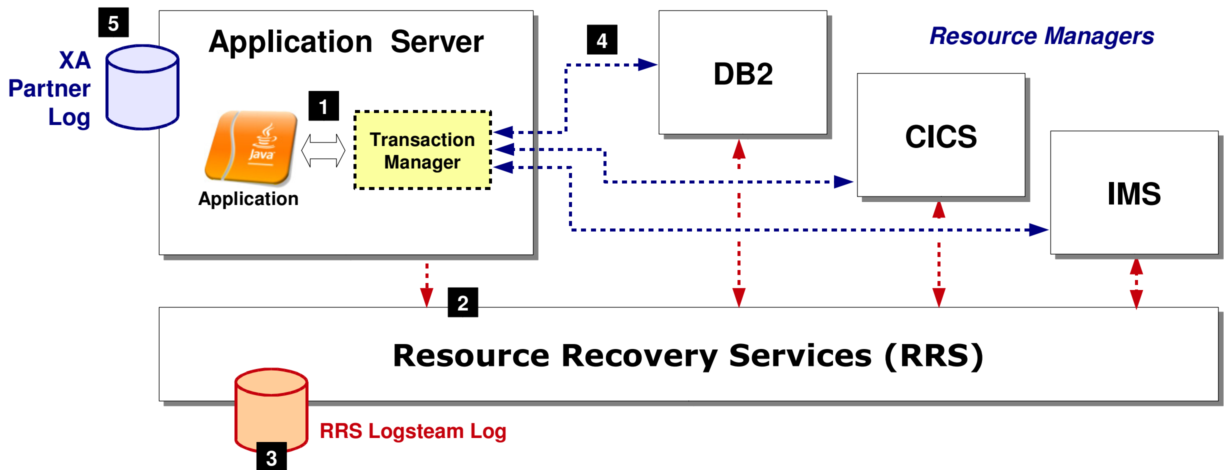
If your DB2 has the JDBC 4 driver, you should use it. Applications written to JDBC 3.0 are compatible with the newer JDBC 4.0 standard. They do not need to be re-written.

The "Implementation Type" plays a role in what transaction capabilities you'll have depending on how you configure your data sources. The "Connection Pool Data Source" implementation is what you'll select if you intend to configure Type 2 data sources. That uses RRS to provide two-phase commit processing. If you intend to use a Type 4 data source and you need two-phase commit processing, then you'll chose "XA Data Source."

All this talk about transactions implies a need to briefly cover that topic ...

Brief Discussion of Transaction Support

The previous chart mentioned RRS and XA as the two means of supporting global transactions from WAS into other resource managers ...



RRS is a facility of z/OS. It is Sysplex aware. The RRS log may be maintained in Sysplex-shared data structures. This allows cross-Sysplex Two-Phase Commit (2PC) processing across instances of WAS and resource managers.

XA is an open standard for distributed Two-Phase Commit. The transaction logs are maintained by WAS.

The "Implementation Type" setting on Provider determines which is used

Provider code supplied by DB2 ...

10 IBM Americas Advanced Technical Skills Gaithersburg, MD

© 2013 IBM Corporation

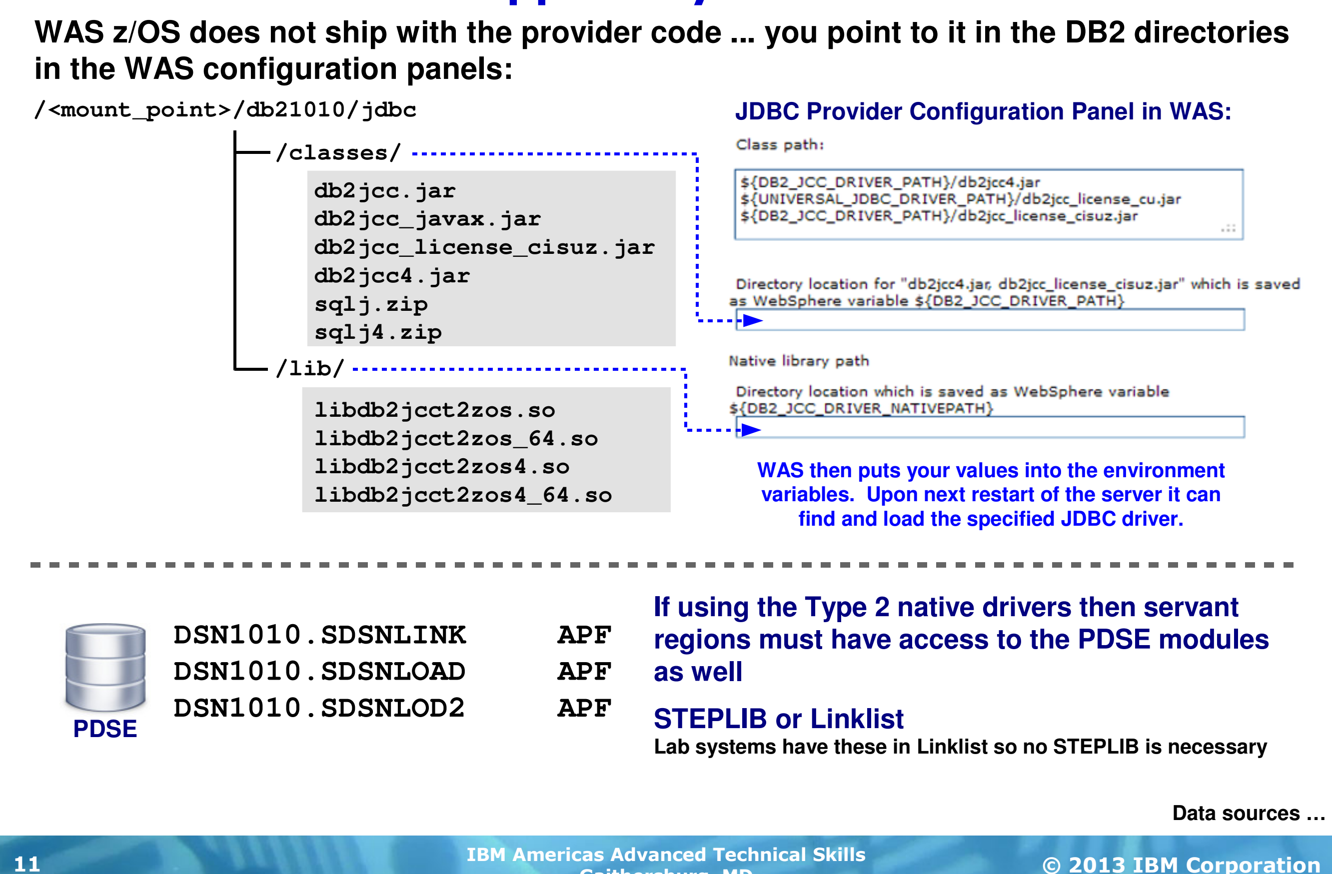
Transaction management is a fairly complex topic, but for the purposes of this workshop we can reduce the discussion to a couple of key things related to WAS z/OS. What we're striving for on this chart is an understanding of how global transactions are initiated and how they're coordinated. This is what helps explain the role of RRS versus XA, and how that relates the JDBC "implementation type" we mentioned on the previous chart.

Here's what the numbered blocks refer to:

1. Global transactions are initiated by the application based on what the application designer knew of the data requirements. In the WAS environment that means the application requests of WAS itself the start of a global transaction for which WAS will be the manager.
2. For transactions that span multiple resources something must act as the coordinator of information about each transaction participant's state. Coordinated multi-resource transaction involve a two-phase process by which the manager (WAS) asks if everyone is ready to commit, and then if everyone agrees then a final commit is issued. If even one party in the transaction says "no" then the transaction is rolled back. Resource Recovery Services (RRS) is one such "synch point coordinator." WAS and the major z/OS data resources all understand how to register into and use RRS for global transactions. RRS is used when the implementation type is "Connection Pool Data Source" and the data source type is Type 2.
3. RRS maintains its transaction information in a Sysplex-enabled logstream data structure. This is what allows RRS to be a Sysplex-wide synch point coordinator.
4. The updates to DB2 and other data resources are made based on the application. When the application requests the transaction manager of WAS to commit the transaction, it then goes through the 2PC processing discussed under #2. If RRS is used, then RRS is the synch point coordinator for WAS as it processes the 2PC.
5. If the transaction is coming from off-platform or the implementation type is "XA Data Source" with a Type 4 connector, then rather than using RRS WAS uses file system logs that maintain the state of the distributed transaction.

The "Implementation Type" influences what synch point coordinator is used.

The Provider Code Supplied by DB2



Data sources ...

© 2013 IBM Corporation

The JDBC Provider code for DB2 is provided as part of the DB2 product installation. There are two pieces to this -- files in the associated DB2 file system, and PDSE module libraries. The PDSE libraries are of interest if you plan to use the Type 2 (cross-memory) driver.

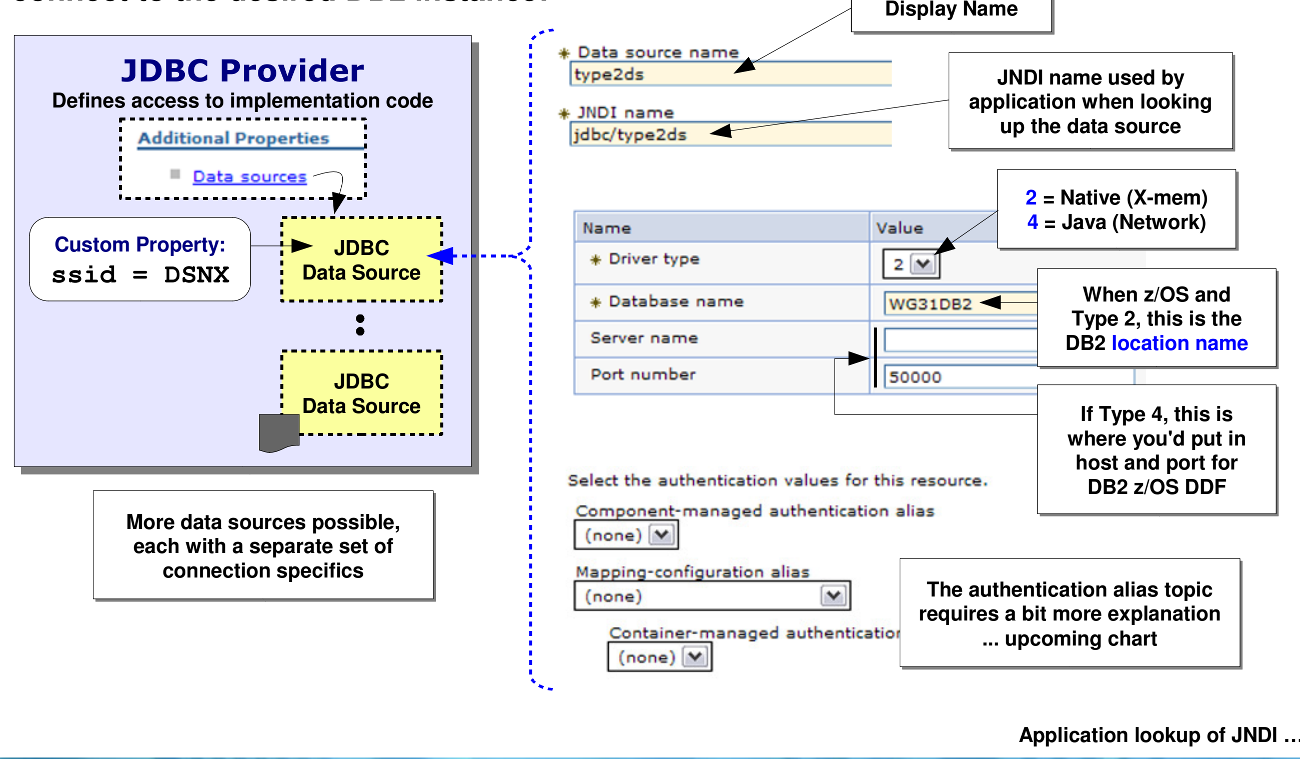
The configuration of the JDBC Provider will as you for the "driver path" and "native library path." The "driver path" is the directory in which the `db2jcc4.jar` file is located. When you specify this path in the Admin Console WAS then saves your value into the `${DB2_JCC_DRIVER_PATH}` environment variable. When the server is restarted WAS will then have knowledge of and access to the JDBC driver class files it needs to load to provide the support.

The "native library path" is needed when you intend to use the Type 2 driver. This will be the `/lib` directory located at the same level as the `/classes` directory. Again, WAS will save the value you provide to an environment variable, this time `${DB2_JCC_DRIVER_NATIVEPATH}`.

If you do plan to use the Type 2 driver then you'll also need to give the WAS *servant* region access to the PDSE libraries `SDSNLINK`, `SDSNLOAD` and `SDSNLOD2`. You may either `STEPLIB` to those (again, from the servant proc) or have them in Linklist. For this workshop they're in Linklist.

JDBC Data Sources -- Specific Connection Information

The data source is defined under the provider and has information about how to connect to the desired DB2 instance:



Application lookup of JNDI ...

© 2013 IBM Corporation

Once the Provider is defined and in place then you may define data sources. Data source definitions are associated with a Provider definition. The easiest way to accomplish this is to go to the Provider general properties page and click on the "Data Sources" link that's off to the right-hand side of the page.

On the data source definition panel there are three sections of information you'll be asked to provide:

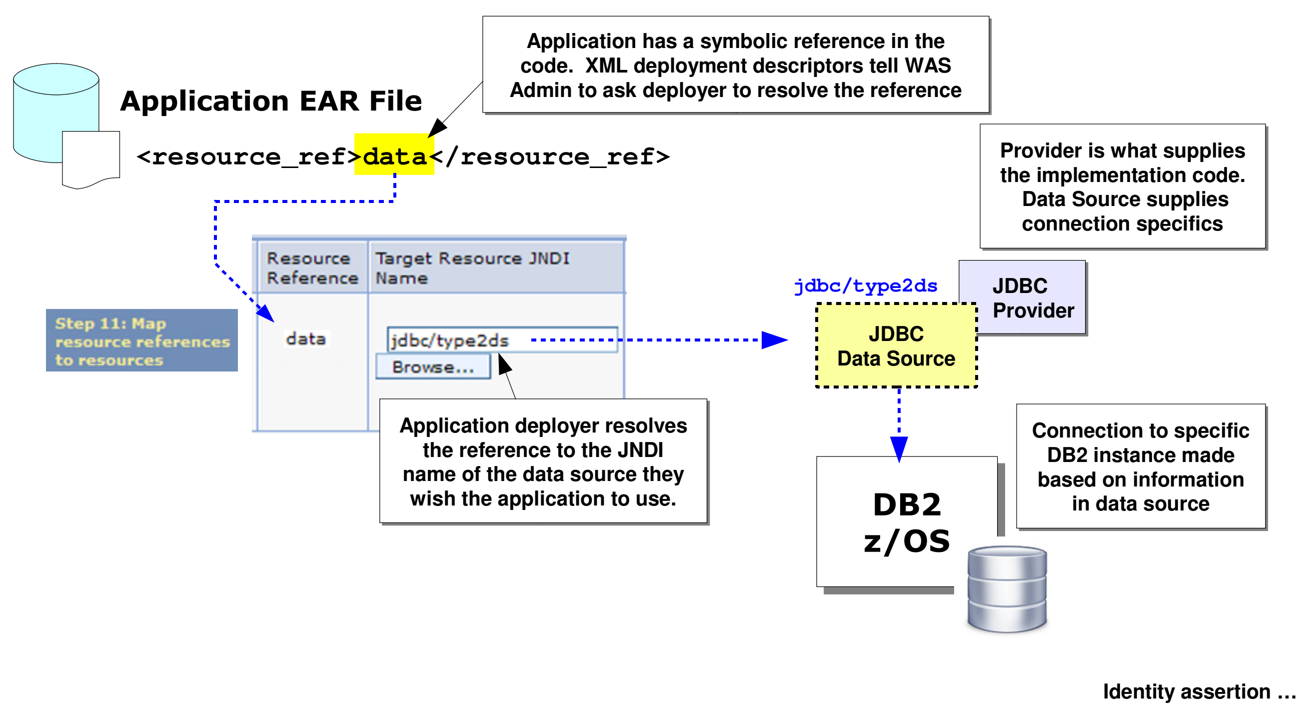
- The display name and the JNDI name for the data source. The display name is simply that -- what is displayed in the Admin Console. The JNDI name is what the application will look up to make the connection to the DB2 instance. The JNDI name may be any valid string ... typically we see people coding this as "jdbc/" and then the display name.
- You'll then specify the Type (2 or 4), the "Database name" -- which for DB2 z/OS is really the DB2 location name, and if you specified Type 4 then the "Server name" (IP host name where the DDF function is listening) and the listener port.
- The authentication alias is a way to set a userid and password pair to be passed over the connection into DB2 so DB2 has an understanding of who it is that's asking for the database services. This is a topic that requires more than a sentence or two to explain, so we'll defer this to an upcoming chart.

If you had multiple DB2 instances in your environment and you wished some applications to go to one and other applications to go to another you would need multiple data source definitions. WAS allows this. You may have as many data source definitions under a Provider definition as you need.

There is a custom property you may add to a data source definition that defines the specific Subsystem ID (SSID) to which WAS should connect when using a Type 2 connector. There are situations where this would not be required, but it is a good practice to supply it. For Type 4 the connection is based on the host and port so the SSID value is not needed.

Application Lookup of Data Source JNDI

Application "resource references" are bound to data source JNDI names ... that's the sequence of associations that ultimately provides JDBC connection



Identity assertion ...

© 2013 IBM Corporation

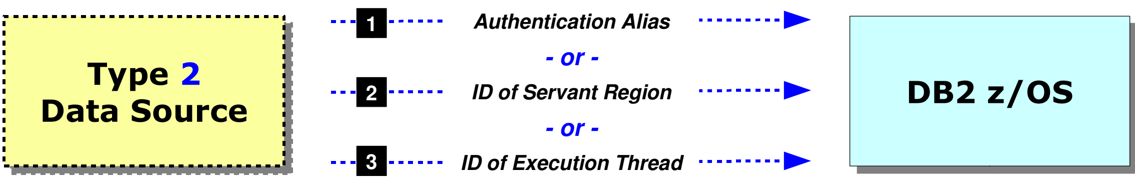
As we mentioned earlier, applications do not hard-code the connection information to the data resource. For JDBC the role of the data source is to provide that specific connection information. The application need only look up and bind to the data source to get the connection through to DB2.

The application references the data source in an abstract way -- it uses string reference in the Java code which is then called out as a resource reference in the XML deployment descriptor. When you deploy the application the WAS Admin Console function looks at all the deployment descriptors to understand what references need resolving. (It is possible to pre-resolve resource references to JNDI names so that application deployment may skip this step ... in this example we're showing the resource reference unresolved, which means the Admin Console will call it out during application deployment.)

If you want an application to use a particular JDBC data source you would resolve the resource reference to the JNDI name of the data source. Then when the application is invoked and seeks to use DB2, it will perform a JNDI lookup of the data source, which tells WAS to create the connection object. That will use the associated Provider code and connect to the specified DB2 instance.

Identity Assertion from WAS into DB2

There's a few different options depending on if Type 2 or Type 4:



1. **An alias is a hard-coded userid/password pair that WAS passes on request**
2. **If no alias then Type 2 uses the ID of the WAS servant region**
3. **Use RunAs roles and map ID of execution thread to request into DB2**

4. **An alias is a hard-coded userid/password pair that WAS passes on request**
5. **New function that allows a distributed LDAP identity to be mapped to a RACF identity**
   Function shipped in z/OS 1.13 and rolled back to 1.11. Required DB2 z/OS V10 to use.

Test Connection button …

14 IBM Americas Advanced Technical Skills Gaithersburg, MD

© 2013 IBM Corporation

Earlier we made mention of the identity that flows from WAS into DB2 for authentication and authorization purposes. What ID flows is a function of which JDBC driver type you're using as well as a few other configuration elements.

If you're using a JDBC Type 2 connector then the ID is based on the following sequence of criteria:

1. If an "authentication alias" is defined and pointed to from the data source definition, then that alias is used. An alias is a stored ID/password pair that is encrypted and maintained in the WAS configuration XML. That ID/password is then flowed over the connection to DB2 and whatever ID is defined in the alias is asserted into DB2.
2. If no alias is defined then WAS will assert the identity of the servant region into DB2.
3. If you have RunAS roles enabled then WAS will take the identity of the execution thread, which would be the identity of the authenticated user into WAS, and assert that across the connection into DB2.

However, if you're using a JDBC Type 4 connector then the ID is based on the following sequence:

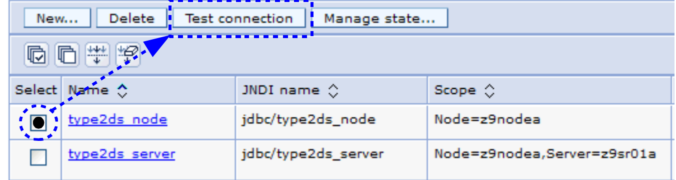
4. Again, if an alias is coded and pointed to from the data source, then the alias flows over to DB2.
5. There is a new function that allows a distributed LDAP identity to flow over to z/OS and be mapped to a RACF ID. The mapped RACF ID is what is used in DB2 (or CICS) with RACF maintaining a log of the association from LDAP-to-RACF that was made for the access to DB2.

In short, some identity will be asserted. The question is what identity and what is it based on.

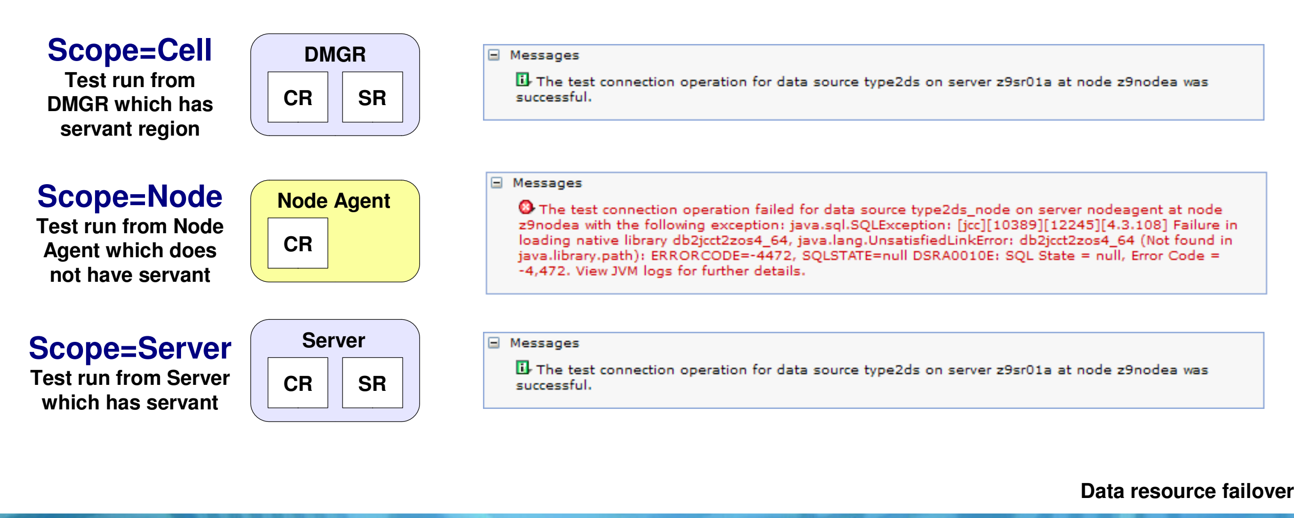
The Wildfire WAS Security workshop goes into much greater detail on these and other security considerations.

The "Test Connection" Button

Will perform a rudimentary connection test … its success depends on the scope of the JDBC Provider



The test is executed from the servant region JVM … so the question is whether the server implied from the scope has a servant region



Data resource failover …

© 2013 IBM Corporation

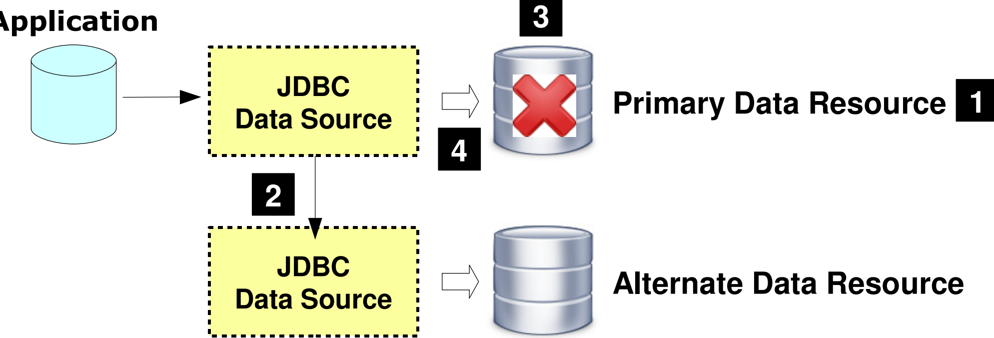
There is a "Test Connection" button associated with data sources, and there has been some confusion about this on z/OS for some time. The confusion arises from the fact that sometimes the button works, and sometimes it does not. The key is understand the "scope" of the data source.

For that "Test Connection" button to work it must have access to a servant region JVM to run the test. If the scope is "Cell" then the DMGR is used, and it has a servant region. The test will succeed, provided the connection is defined properly. If the scope is "Server" then the named server has a servant and the the test will succeed.

But if the scope is "Node" -- which is the most common scope for data resources -- then the test is attempted from the Node Agent, which has no servant region. The test fails. You get the error shown on the screen.

Data Resource Failover - Four Questions

The new function in WAS V8 (all platforms) is designed to address four questions related to data resource failover and failback:



1. **Has the primary data resource failed?**
   We'll discuss the mechanism used to trigger the failover function
2. **What alternative data resource is available?**
   This is defined with a new variable
3. **Has the primary data resource recovered?**
   A test for primary resource recovery is made
4. **Should failback to the primary be manual or automatic?**
   You may not want automatic failback … there are ways to control this

Essentials of failover support …

 © 2013 IBM Corporation

Now we'll turn our attention to a new feature of WAS V8, which is present on all platforms. Later we'll take a look at how WAS z/OS V8 extended this with a few other platform-exclusive functions.

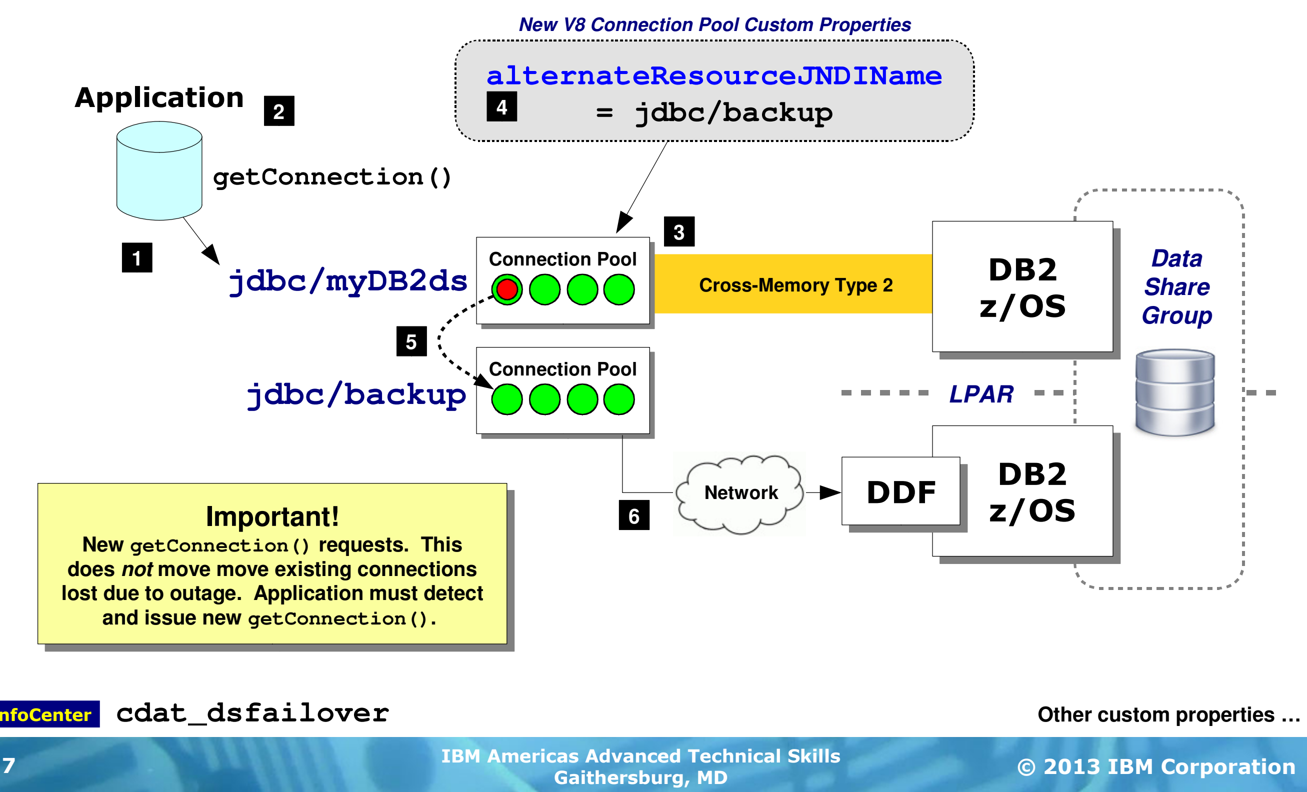
This new feature has to do with failing over to an alternative data resource when the primary resource is lost. In designing this new feature the developers had to answer four design questions:

1. *How do we determine that the primary data resource has failed?* This is done by watching for failures on application `getConnection()` requests. A failure to get a connection to the data resource behind the data source is an indication something is wrong. As you'll soon see, there's a parameter to indicate how many consecutive failed `getConnection()` requests it takes to trigger the failover.
2. *How do we know what alternate resources are available?* This is done with a new custom property that defines the alternate data source to use in the event the primary is deemed lost.
3. *How do we know when the primary data resource has recovered?* This is done by having WAS periodically issue a test connection request to the primary data resource. New custom properties define how frequently this polling is done.
4. *If the primary resource is recovered, do we fail back automatically or failback manually?* The answer is either. Controls are provided to allow automatic failback or to disable automatic failback and allow you to manually restore the primary connection with a z/OS MODIFY command.

In other words, the designers of this new function considered the cycle of events -- failure, detection of failure, failover, primary recovery, detection of primary recovery, failback -- and designed settings around that cycle.

Essentials of Resource Failover

A new environment variable is used to define an "alternate JNDI" for use when the primary JNDI experiences `getConnection()` problems:



InfoCenter **cdat_dsfailover**

Other custom properties …

© 2013 IBM Corporation

This chart provides a picture of the essential process of resource outage detection and failover. The charts that follow will add to the details and provide the complete picture.

Note: this picture is mapped onto a DB2 z/OS data sharing group scenario because that's where this new feature will really shine. In a Sysplex environment with a WAS cluster across multiple LPARs and a DB2 sharing group in the background, this allows Type 2 cross-memory access with the ability to failover to a Type 4 connection to the other LPAR.

Let's walk through the numbered blocks on the chart:

1. Applications perform a JNDI lookup of the data source as part of their initialization. Applications know which data source JNDI to use based on how the application was configured at deployment time. Resource references in the application's deployment descriptors are mapped to the configured data source JNDI names. That's what resolves the abstract reference in the application to the actual data source.
2. Applications request a connection from the data source connection pool by issuing a `getConnection()` request.
3. If the `getConnection()` is successful a connection from the connection pool is given over to the application. Here we're illustrating that connection being a Type 2 cross-memory connection to DB2.
4. A new custom property to the connection pool definition defines an alternate data source JNDI to use in the event the primary data source is lost (as determined by failed `getConnection()` requests). The JNDI name provided on this property must be associated with a properly defined data source.
5. Imagine an application issues a `getConnection()` request and it fails. (We'll soon see that there's another property to define a number of consecutive failures that triggers this.) What WAS will now do is consult the `alternateResourceJNDIName` custom property and provide the requesting application a connection object from the alternate data source connection pool.

   Note: as indicated on the chart, this is for *new* connections. *Existing* connections are lost and it's up to the application to re-establish the connections, which would then come from the alternate connection pool.
6. In this picture we're showing the alternate data source defined to use Type 4 cross-LPAR to another instance of DB2 in the data sharing group.

Other Connection Pool Custom Properties

Four other connection pool custom properties are also made available:

`failureThreshold`

Determines the number of consecutive `getConnection()` failures are needed to trigger the failover processing
Integer, Default = 5

`resourceAvailabilityTestRetryInterval`

After failover has occurred, this determines the frequency of polling to see if the primary resource has recovered
Integer, Default = 10 seconds

`enablePartialResourceAdapterFailoverSupport`

Indicates that automatic failover is permitted but automatic failback is disabled
Boolean, Default = False

`disableResourceFailOver`
`disableResourceFailBack`

Disables automatic failover or failback. Used to allow configuration of failover values, but control using z/OS `MODIFY`
Boolean, Default = False

InfoCenter `cdat_dsfailover`

z/OS MODIFY ...

© 2013 IBM Corporation

We just saw one new connection pool custom property -- `alternateResourceJNDIName`, which defines the JNDI name to fail over to in the event WAS determines it's time to perform the failover. But that custom property is not the only custom property associated with this new function. Here are the others. Note the InfoCenter search string -- these properties are spelled out in detail in that article.

- **`failureThreshold`** -- this property determines how many `getConnection()` failures in a row are needed to trigger the failover ot the defined alternate JNDI name. This is an integer value and it defaults to 5. If you set this value to 1 you create the possibility for failover due to some transient connection problem. A number higher than one insures WAS sees a persistent issue before failing over.
- **`resourceAvailabilityTestRetryInterval`** -- this is related to how frequently WAS issues the test connection request to see if the primary resource has recovered after a failover has been executed. This is an integer and defaults to 10 seconds. Setting this value to a very low number implies increased overhead due to polling; setting this value to a very high number implies infrequent polls to test for primary resource recovery.
- **`enablePartialResourceAdapterFailoverSupport`** -- this tells WAS to failover but do *not* automatically failback. There may be cases where you wish to control manually failback takes place. Failing back to the primary data resource can be accomplished with the z/OS MODIFY command which we'll cover on the next chart. So coding this tells WAS to failover but leave failing back to a manual process.
- **`disableResourceFailOver`** and **`disableResourceFailBack`** -- with either of these set to True the *automatic* failover and failback is disabled. But the MODIFY failover and failback (next chart) still applies. So these custom properties allow you to set up the alternate JNDI name but leaves triggering the failover and failback to you. This is particularly useful for cases of planned outages.

z/OS MODIFY Control of Failover and Failback

The following MODIFY commands will act upon a server where the connection pool custom property `alternateResourceJNDIName` has previously been configured:

Manual Failover to Alternate and Failback to Primary

```
F <server>,FAILOVER,'<JNDI Name>'
F <server>,FAILBACK,'<JNDI Name>'
```

The JNDI name is that of the primary data source. Never the defined alternate data source.

Note the *single quotes* enclosing the JDNI name

Manual Disable or Enable of Automatic Failover / Failback

```
F <server>,DISABLEFAILOVER,'<JNDI Name>'
F <server>,ENABLEFAILOVER,'<JNDI Name>'
```

The JNDI name is that of the primary data source. Never the defined alternate data source.

These MODIFY commands override connection pool custom properties you may have set of enable and disable of failover and failback

InfoCenter **rxml_mvsmodify**

z/OS Action Notification ...

 © 2013 IBM Corporation

We've alluded to new z/OS MODIFY commands that affects the failover and failback capabilities, and here they are:

- `F <server>,FAILOVER,'<JNDI Name>'` -- when `alternateResourceJNDIName` is coded and a properly defined alternate data source with that JNDI name exists, then this MODIFY command will invoke the failover at the time you issue this command.

  Note: in *all cases* the `<JNDI Name>` value is the *primary* data source JNDI name ... *never* the alternate resource JNDI name. `alternateResourceJNDIName` is a custom property on the *primary* data source connection pool. By providing the JNDI value for the primary you allow WAS to read the custom property for the alternate JNDI name and use it for manual failover.

- `F <server>,FAILBACK,'<JNDI Name>'` -- this tells WAS to fail back to the primary resource when this command is issued.
- `F <server>,DISABLEFAILOVER,'<JNDI Name>'` -- this will disable automatic failover. It has the same effect as the custom property `disableResourceFailOver` discussed on the previous page, but the MODIFY command allows the effect to be dynamically imposed without requiring a server restart. The MODIFY FAILOVER will invoke failover even though the automatic failover is disabled.
- `F <server>,ENABLEFAILOVER,'<JNDI Name>'` -- this will disable automatic failback. It has the same effect as the custom property `disableResourceFailBack` discussed on the previous page, but with the benefit of being dynamically imposed. Again, MODIFY FAILBACK will invoke failback even though automatic failback is disabled.

z/OS failureNotificationActionCode

These define actions to take when the primary is unreachable *and* any defined alternate JNDI resources are also unreachable:

```
failureNotificationActionCode = 1 | 2 | 3
```

1 **Issue a `BBOJ0130I` message, but take no other action**

```
BBOJ0130I: CONNECTION MANAGEMENT IN A SERVANT REGION DETECTED THAT THE
RESOURCE IDENTIFIED BY JNDI NAME jdbc/type2ds IS DISCONNECTED FROM SERVER
z9cell/z9nodea/Z9SR01/z9sr01a. ACTION TAKEN: NONE.
```

2 **Issue `PAUSELISTENERS` for the server; `RESUMELISTENERS` when resource is back**

```
ACTION TAKEN: PAUSING LISTENERS.
BBOO0222I: ZAIO0002I: z/OS asynchronous IO TCP Channel TCP_1 has stopped
listening on host * port 10065.
 :
BBOO0222I: ZAIO0002I: z/OS asynchronous IO TCP Channel TCP_4 has stopped
listening on host * port 10068.
```

Front-end routing devices will detect loss of listener ports and route to other members of a cluster

3 **Stop applications using failed resource; restart applications when resource is back**

☐	My IVT Application	➔
☐	PolicyIVPV5	✖
☐	SuperSnoop	➔

Makes affected application unavailable but leaves intact other applications in the server

Non-relational …

© 2013 IBM Corporation

The `failureNotificationActionCode` setting takes effect when WAS z/OS detects the defined primary JNDI is unreachable as well as any defined alternate JNDI resources. If WAS z/OS detects either primary or alternate *is* reachable, then this property's behavior does not take effect.

1. If the value is set to **1** then all WAS z/OS will do is issue a message but take no other action. This is one step more than occurred prior to WAS V8 where no indication was offered.
2. If the value is set to **2** then WAS z/OS will issue a `PAUSELISTENERS` command for the affected server. No additional work will flow to this server. The benefit of this is that many front-end work routing functions will detect the absence of the listener port and invoke other routing options, most notably routing to other members in a WAS cluster. When WAS z/OS detects the failed resource has recovered, it will issue a `RESUMELISTENERS` command, which opens the server back up for work. The downside to this is that all applications in the server are affected by this ... even applications not using the data source with the failed resource behind it. That's why the third option exists ...
3. If the value is set to **3** then WAS z/OS will stop the applications that are using the data source JNDI that has experienced the connection failure to the backend data resource. Other applications remain active and use whatever data sources they are using that are in good working order. Be aware that many front-end routing functions will not detect this. But some will -- the WAS z/OS Proxy Server will, as will the WAS On Demand Router.

If the value is set to something other than 1, 2 or 3 the function is ignored.

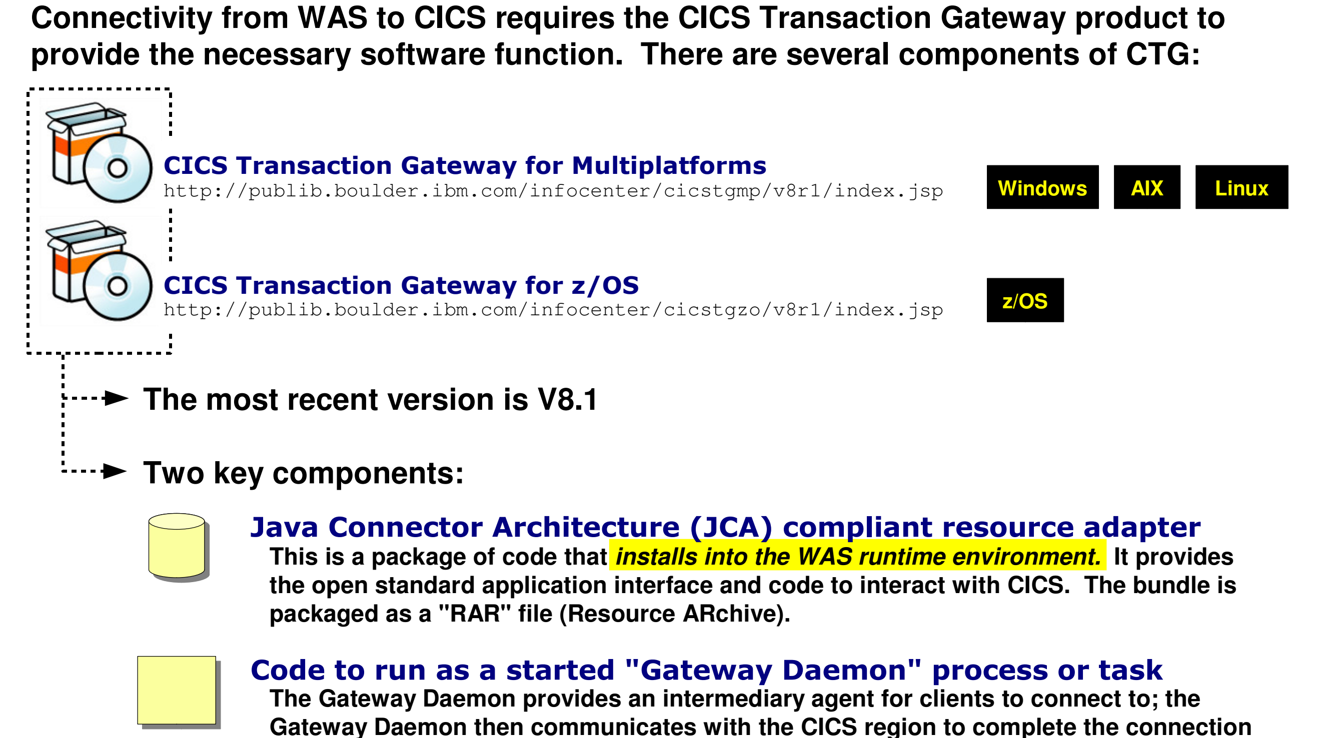
If you want these behaviors but *not* automatic failover, then omit the `alternateResourceJNDIName` property, or have that property set as well as the `disableResourceFailOver` property set. Automatic failovers won't occur but one of the three behaviors above will.

Non-Relational Data Access

CICS

© 2013 IBM Corporation

The Role of the CICS Transaction Gateway Product



Two topologies …

22 IBM Americas Advanced Technical Skills Gaithersburg, MD © 2013 IBM Corporation

We start this by bringing CICS Transaction Gateway into the discussion. It plays a key role in connecting Java EE applications (running in WAS, as an example) to CICS.

CICS Transaction Gateway (CTG for short) is a separately licensed product from either CICS or WAS. That means it must be acquired and properly licensed to use. It comes packaged for Multiplatforms (Windows, AIX, Linux) and z/OS. The most recent version is 8.1.

There are two key components to CTG, and it's important to understand this distinction to properly understand the different topologies possible using WAS and CTG.

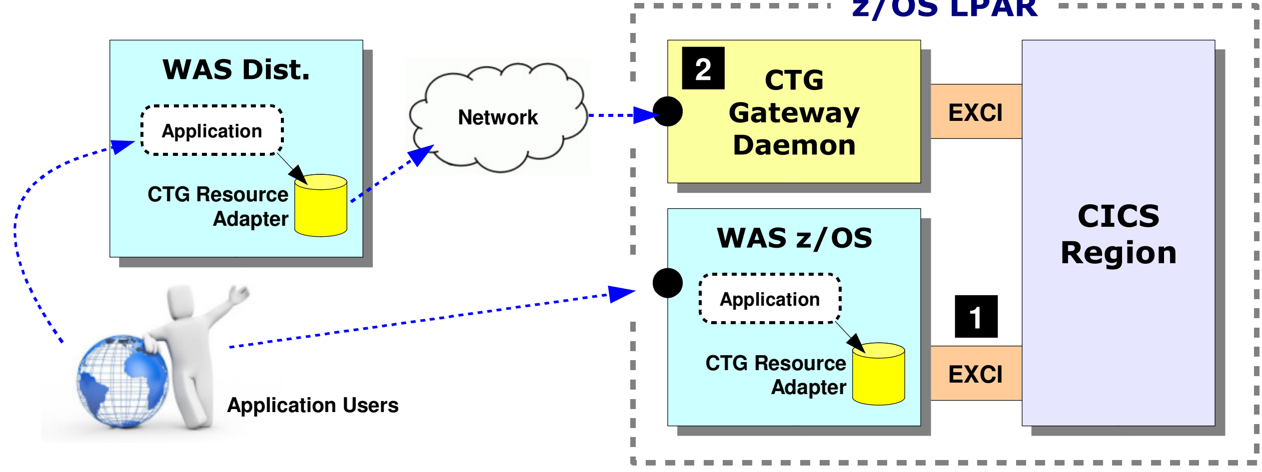
The first component is a Resource Archive file (RAR file) that contains the code for a fully-compliant Java Connector Architecture (JCA) resource adapter. This gets installed into the WAS runtime environment and provides the Java class files your application would use to interact with CICS. Those Java class files implement the Common Client Interface (CCI), an open standard programming interface. They also provide the function that understands how to communicate with CICS.

The second component is code that runs as the "Gateway Daemon" -- a started task or process that listens on a network port and acts as a "gateway" (hence the name) between the client and the CICS region.

Depending on the topology you choose to employ, you would use one or both of these components. The RAR file is *always* needed for applications to communicate from WAS to CICS. The Gateway Daemon may be used, depending on the topology you choose to use.

Two Simple Topologies … to Start the Discussion

There are several variations on topologies and it can be a bit confusing at first. Let's start with two relatively simple examples to set some context:



1. **WAS uses RAR to access CICS with EXCI**
   This is known as "Local Mode" in CTG terminology
2. **WAS uses RAR and TCP to access Gateway; Gateway uses EXCI to access the CICS region**
   This is known as "Remote Mode" in CTG terminology

We need to introduce IPIC

IPIC …

23 — IBM Americas Advanced Technical Skills, Gaithersburg, MD

© 2013 IBM Corporation

Rather than put up a chart that shows all the variations of toplogies (that gets confusing), let's focus on two relatively simple (and common) topologies.

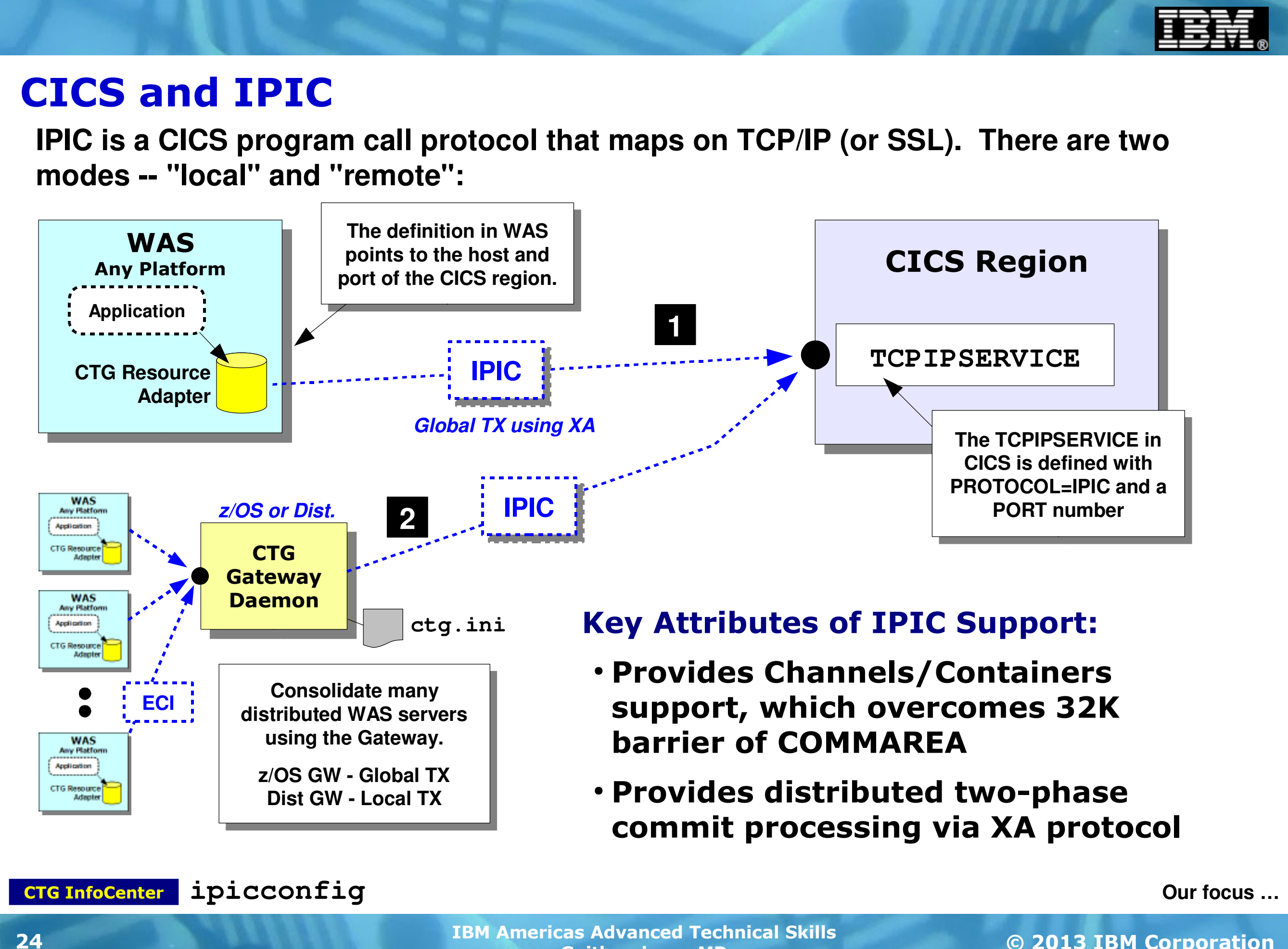
1. The first involves WAS on z/OS co-located with the target CICS region on the same LPAR. The connectivity between WAS and CICS is using the CICS External CICS Interface (EXCI). Users on the web come into the WAS server over whatever network hops are between them and WAS. WAS uses the installed CTG Resource Adapter (along with native code files not pictured on the chart) to drive the EXCI interface and gain access to the CICS region. In CTG terminology this is known as "local mode."

   There is no Gateway Daemon in this first scenario. All that's needed is the CTG Resource Adapter. This scenario supports COMMAREA but not Channels/Containers. Two Phase Commit (2PC) processing is supported using RRS as the synch point coordinator.

2. The second scenario involves WAS on a distributed platform (Windows, AIX, Linux) accessing CICS over a network. In this case a CTG Gateway Daemon on z/OS is used. The CTG Resource Adapter is also used in the WAS server on the distributed platform. The CTG Gateway Daemon listens on a defined port for incoming connection requests from the distributed WAS server. The Gateway Daemon then turns and uses EXCI to connect to the CICS region. In CTG terminology this is known as "remote mode."

   In this case there is *both* a CTG Resource Adapter (installed into the WAS environment on the distributed platform) *and* a Gateway Daemon instance.

This chart represents two relatively simple topologies. But there's a piece missing from this chart -- IPIC. IPIC is a means of transferring ECI calls over TCP/IP to a CICS region. Let's take a look at IPIC, then we'll get into how all this works with WAS z/OS.



IPIC is a protocol CICS uses for clients to interact with CICS over the network. It first came into being some years back ... in CICS 3.2 and CICS Transaction Gateway 7.2. It provides support for Channels/Containers (which permits messages larger than 32K, which is the limit for COMMAREA). It provides global transaction support with two-phase commit using XA when in "local" mode (no Gateway between WAS and CICS), or using the Gateway on z/OS.

A `TCPIPSERVICE` is defined in CICS to use IPIC as its protocol and listen on a specified port. The CTG InfoCenter has information on how this is done. It's relatively standard CICS system programmer work. It is an essential piece of this ... absent this `TCPIPSERVICE` WAS will not be able to talk IPIC to CICS.

There are two "modes" -- "local" and "remote" ... both of which are somewhat confusing, but we'll clarify the distinction here.

1. In "local" mode there's no Gateway Daemon ... the CTG Resouce Adapter installed into the WAS runtime (on any platform) is configured to use IPIC to communicate with the CICS region IPIC port. That configuration is done in the JCA Connection Factory custom properties, which we'll show in an upcoming chart. The WAS server communicates directly with the CICS region using the IPIC protocol. IPIC rides on either TCP or SSL.

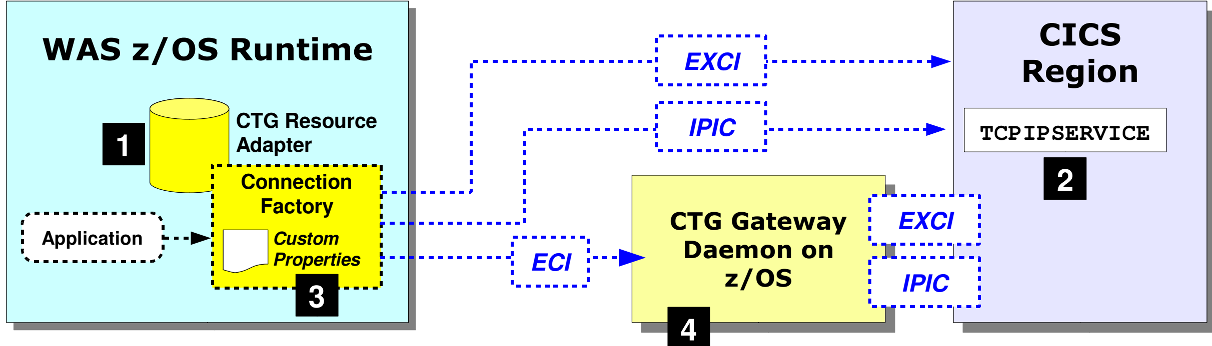
   Note: a limitation of this is that there must be an `IPCONN` definition for each WAS server connecting to the CICS region. You may wish to configure in "remote" mode (next) to overcome this.

2. In "remote" mode there *is* a Gateway Daemon. WAS runtimes (any platform) communicate with the Gateway using ECI, and the Gateway then uses IPIC to communicate with the CICS region. This topology is useful when there's a larger number of WAS server environments seeking to communicate with CICS. The Gateway serves as a consolidation point for the connection into the CICS `TCPIPSERVICE`.

   Note: global two phase commit using RRS is possible when the Gateway Daemon is located on z/OS. If the Gateway Daemon is on a distributed platform then local transaction only.

Many Options … Our Focus Will Be on WAS z/OS

This workshop is focused on WAS z/OS, so our discussion of CTG for access to CICS will be on z/OS-related topologies:



1. The installation of the CTG Resource Adapter

Which is the starting point to providing WAS-to-CICS connectivity

2. The basics of the `TCPIPSERVICE` and `IPCONN` definitions in CICS

To show the interrelationship between values there and what's coded on the connection factories

3. The configuration of JCA Connection Factories

In particular the configuration of the custom properties in support of the connection types -- EXCI, IPIC or to Gateway Daemon

4. An overview of the CTG Gateway Daemon

To give you a sense for the structure and configuration settings of the Gateway Daemon

CTG RAR file …

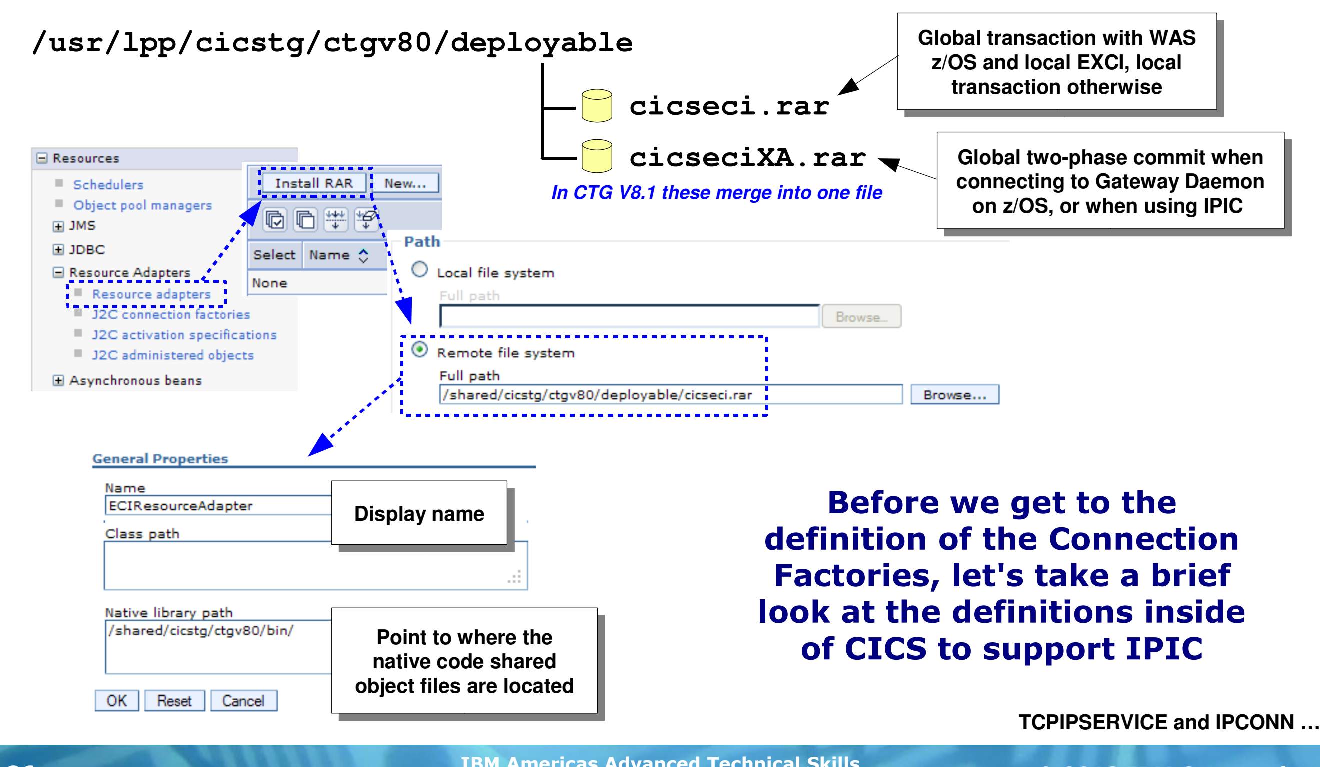
 © 2013 IBM Corporation

The combination of topologies possible with WAS across all platforms and CTG in its various uses can get large. Rather than try to perform a survey on all those combinations, we'll focus instead on those related to WAS on z/OS. That means we'll cover the following in this section:

1. *Installation of the CTG Resource Adapter into WAS z/OS* -- this is a relatively easy process done through the WAS administrative console. The resource adapter RAR file ships with CTG, and it's just a matter of pointing to that RAR file during the installation process. WAS then copies the RAR binaries into its configuration directory and you're done. But this must be done, otherwise there can be no connectivity using ECI, IPIC or EXCI to CICS.
2. *Overview of the CICS configuration in support of IPIC* -- to support IPIC connections coming in from a WAS server, the CICS region must have a `TCPIPSERVICE` definition and an `IPCONN` definition. The values provided there correspond to what you provide on the connection factory definition.
3. *Configuration of the JCA Connection Factories* -- connection factories are like JDBC data sources ... they carry information about the connection specifics. Depending on the nature of the connection (EXCI, IPIC, or to a Gateway Daemon instance) the values in the connection factory are different. We'll show you what those values are and illustrate how they correspond to the values in the CICS region.
4. *Overview of the CTG Gateway Daemon on z/OS* -- the CTG Gateway Daemon on z/OS is a started task that listens on a defined port, takes in requests from clients and turns to the CICS region and passes the requests in. It consists of a program that is launched with supplied sample JCL, and a configuration file that contains the information about what port to listen on, and how to connect to CICS.

The CTG Resource Adapter RAR File

The RAR (Resource ARchive) is the adapter in its installable packaging format. You use the Admin Console to install that RAR file into the WAS runtime environment:



Before we get to the definition of the Connection Factories, let's take a brief look at the definitions inside of CICS to support IPIC

TCPIPSERVICE and IPCONN ...

IBM Americas Advanced Technical Skills
Gaithersburg, MD

© 2013 IBM Corporation

The CTG Resource Adapter RAR file is shipped with the CTG product. When CTG is installed on z/OS you'll find the RAR file in the `/deployable` directory under the installation root. Prior to CTG V8.1 there were two RAR files -- one for non-XA connections, one for XA connections. Starting with CTG V8.1 they are combined into one RAR file with both XA and non-XA support in the same file, with a new custom property indicating XA or not.

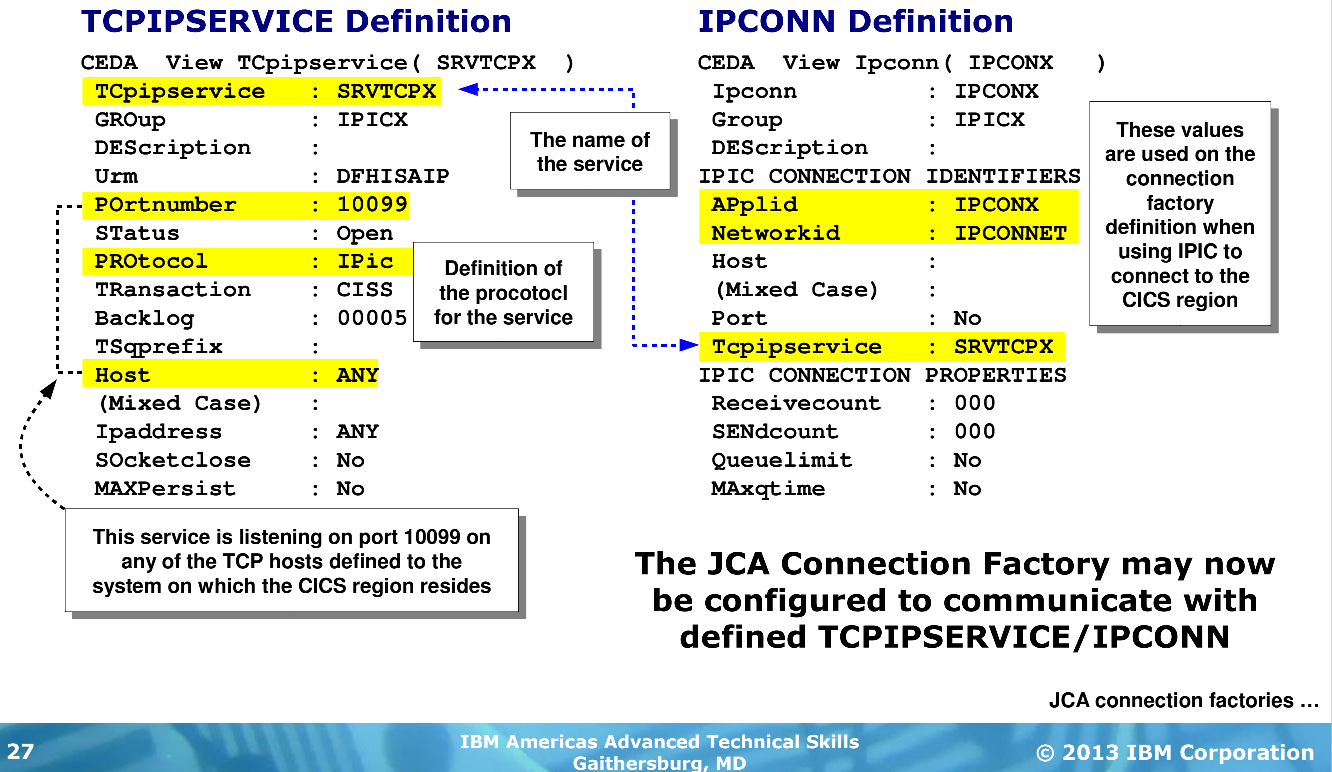
The RAR file is installed through the Admin Console by clicking on the "Resource Adapters" link, then the "Install RAR" button, then pointing to the RAR file to install. The final step is to provide the definition knowledge of where the CTG native code library is, which will be the `/bin` directory off the CTG installation root.

That's it ... a relatively simple process to install the resource adapter. This is typically done with a scope of "Node," which means you'd install the resource adapter across all nodes in which servers would need access to CICS.

The next step is to define the JCA Connection Factories to provide information about the connection to CICS.

CICS Definitions in Support of IPIC Usage

Two elements to this -- the TCPIPSERVICE definition and the IPCONN definition. Values you provide here are used in the JCA Connection Factory definition ...



The JCA Connection Factory may now be configured to communicate with defined TCPIPSERVICE/IPCONN

JCA connection factories ...

27 IBM Americas Advanced Technical Skills Gaithersburg, MD

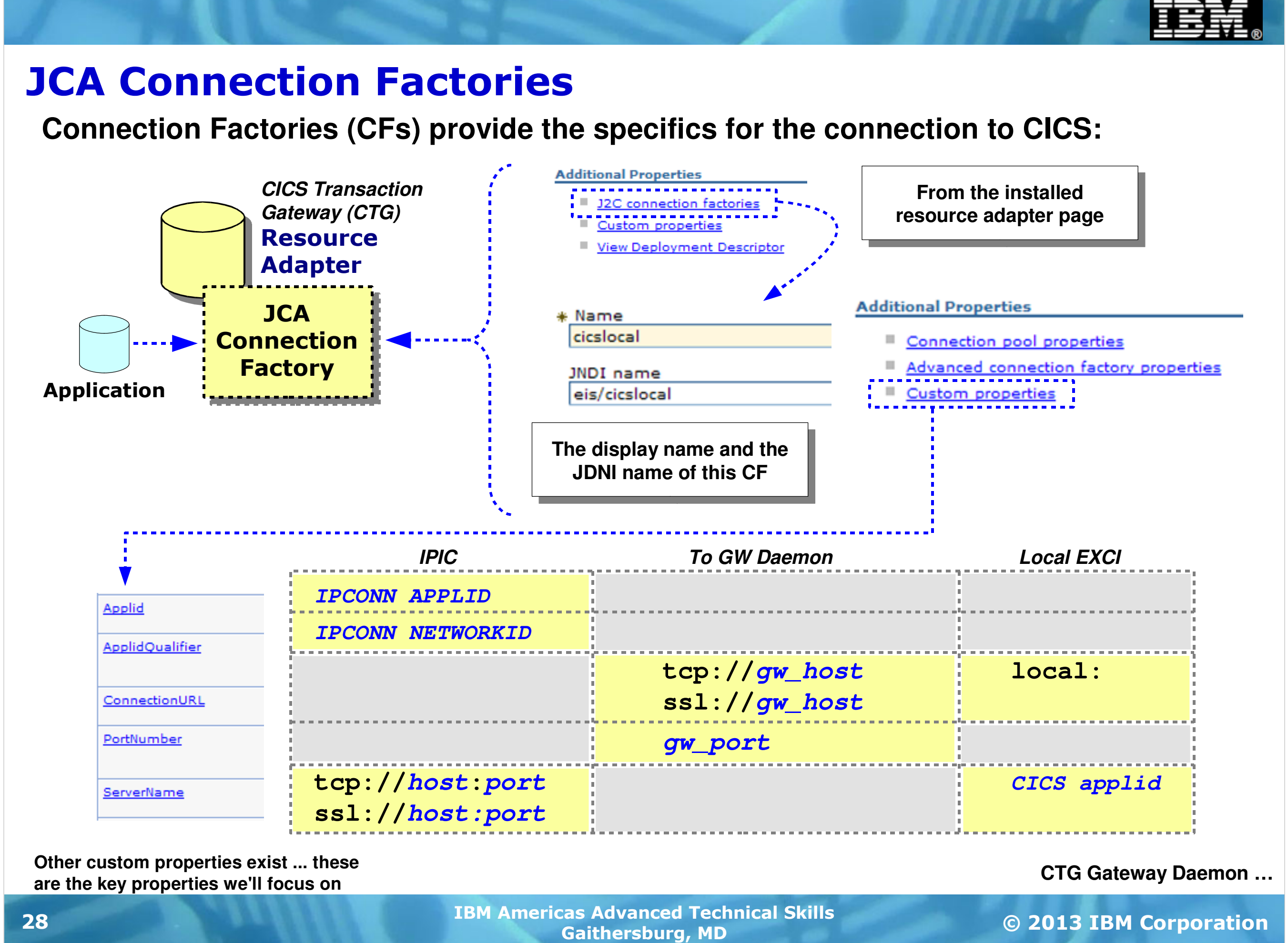
© 2013 IBM Corporation

Use of the IPIC protocol presumes there's a `TCPIPSERVICE` in the target CICS region to receive the IPIC calls. This chart shows the `TCPIPSERVICE` and `IPCONN` definitions in use on the CICS region for the upcoming labs.

`TCPIPSERVICE` -- this defines the TCP port and host the service will listen on, as well as the protocol to be used. You see the definition of IPIC as the protocol, a port value of 10099, and an indication that the host to listen on is "Any," which means any of the defined TCP stacks on the z/OS system. At the top of the definition is the name of the `TCPIPSERVICE`.

`IPCONN` --this defines a connection. We're highlighting two things here -- the pointer to the `TCPIPSERVICE` this `IPCONN` is associated with, and the `APPLID` and `NETWORKID` for this `IPCONN`. The service name here must match the name on the `TCPIPSERVICE` definition. The `APPLID` and `NETWORKID` are key values that get coded on the JCA connection factory.

Let's see what the JCA connection factory definition looks like ...



Much like the definition of JDBC data sources, JCA connection factories are associated with the installed resource adapter. If you go to the general properties page of the installed adapter, off to the right you'll see a link for "J2C connection factories." Clicking on that link will take you to the panels to define the connection factory.

The first part of this is a set of names -- one is the display name used by WAS to display the CF on the Admin Console. The other is the JNDI name that will be associated with this CF. Application resource references for CICS are resolved to the JNDI name of the CF ... that's the mechanism for connection to the CICS region.

But the real specifics get defined as a "custom property" to the connection factory. There's a rather long list of custom properties, but the ones of particular interest are shown on the chart. The values you code into each depends on what kind of connection you wish to make to the CICS region. The chart shows a matrix of what to provide into the various custom properties, depending on the connection you're wishing to make.

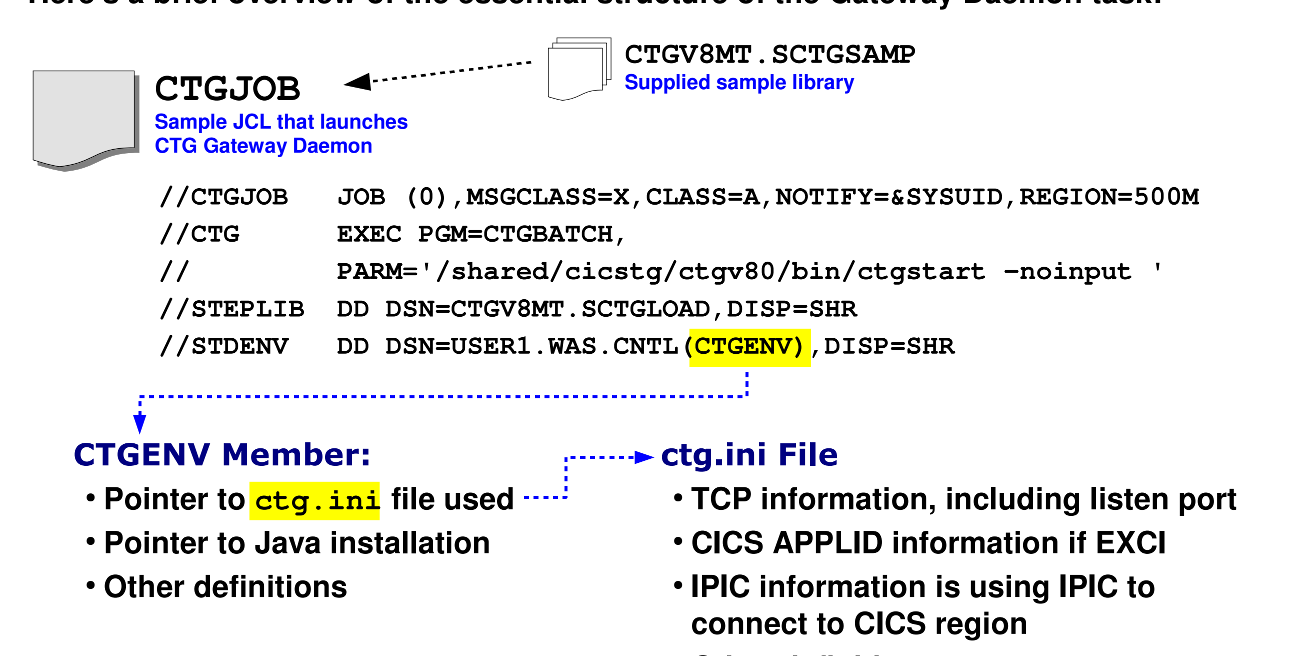
Note: black non-italic lettering indicates a keyword, blue italic indicates a value you supply.

- IPIC -- this is a direct connection using IPIC to a CICS region with a defined TCPIP service and IPCONN definition. The CICS region may be on the same LPAR or a different LPAR from the WAS z/OS server. We showed on the earlier chart the `TCPIPSERVICE` and `IPCONN` definitions in CICS so you could relate the values there to the values to code into the custom properties.
- Connection to CTG Gateway Daemon -- this provides information about the host and port where the CTG Gateway Daemon is listening. This may be on the same LPAR as the WAS z/OS server or on another LPAR.

  Note: this defines the connection from WAS to the GW Daemon. The connection from the GW Daemon to the CICS region is defined in the GW Daemon's configuration files.
- Connection to CICS region using local EXCI -- this is the case where the WAS z/OS server and the target CICS region are on the same LPAR. Here all that's needed is the keyword `local:` (with the colon, that's important) and the APPLID of the CICS region (*not* the APPLID of the IPCONN, that's different and is not used when connecting using EXCI).

CTG Gateway Daemon

Here's a brief overview of the essential structure of the Gateway Daemon task:



Consult the CTG InfoCenter for specifics on customizing CTG Gateway Daemon

Resource failover/failback ...

© 2013 IBM Corporation

The CICS Transaction Gateway Daemon is a long-running task started using a supplied JCL batch job. The JCL has more to it than it shown on this chart ... what's shown here is the essential pieces we wish to illustrate.

- The CTG Gateway Daemon is started using a supplied CTG shell script launching program called `CTGBATCH`.
- A parameter points to where the `ctgstart` shell script file is located
- The `STDENV DD` card points to an environment member you customize. The sample library provides a skeleton to use when starting out.
- In the `CTGENV` member are some key customization values, including a pointer to the `ctg.ini` file to be read in, a pointer to where a copy of the Java SDK may be found, and a handful of other definitions.

Note: The CTG InfoCenter has full details on how to customize the `CTGENV` member and the `ctg.ini` file. We won't supply that information here. The URL for the CTG InfoCenter was presented on the first page of this section.

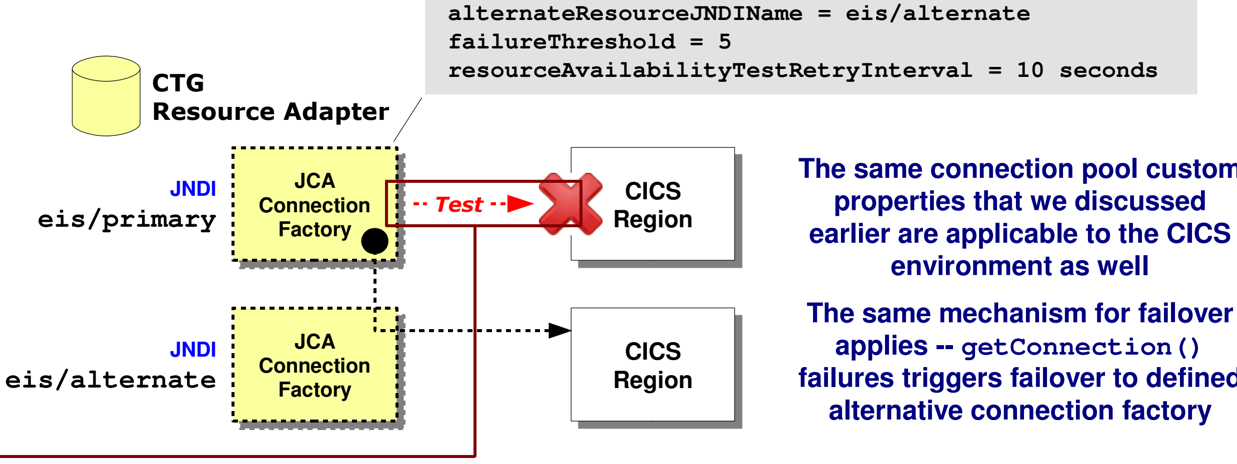
- The `ctg.ini` file is what determines the listener port for the Gateway Daemon and whether it'll use EXCI or IPIC to the backend CICS region. If EXCI then you specify the APPLID of the CICS region; if IPIC then you supply the host, port, APPLID and network ID of the IPCONN definition, much like you would do with a connection factory using IPIC.

When all the configuration settings are made and the Gateway Daemon starts, it opens up a listener port on the front end ... that's what WAS servers use to connect using ECI over the network. The Gateway Daemon then uses either EXCI or IPIC to CICS region.

Note: there's a lot we're not covering with respect to the capabilities of the CTG Gateway Daemon. For instance, we brushed right past security. There are ways to configure the CTG Gateway Daemon in a shared port, highly available configuration. The CTG InfoCenter has all this information.

Resource Failover and Failback -- Work with CICS?

The resource failover methodology we explored for JDBC applies here as well, with one notable exception -- automatic failback:



The same connection pool custom properties that we discussed earlier are applicable to the CICS environment as well

The same mechanism for failover applies -- `getConnection()` failures triggers failover to defined alternative connection factory

The issue is the test connection ... at the present time the CTG resource adapter code "test connection" process will always indicate a positive, even when CICS is not there.

Use `enablePartialResourceAdapterFailoverSupport` or `disableResourceFailBack` to prevent failback polling. Then use `MODIFY` failback when you know CICS region is truly back.

Messaging ...

IBM Americas Advanced Technical Skills
Gaithersburg, MD
© 2013 IBM Corporation

A question comes up -- does the resource failover mechanism discussed for use with JDBC work with CICS connections as well? And the answer is a qualified "yes."

CICS connections also maintain connection pools, and the connection pool properties we discussed earlier apply to CICS connection pools as well. It is quite possible to define two connection factories -- one primary, one alternate -- and use the new custom properties to define the failover properties. The same mechanism would trigger the failover: a number of failed `getConnection()` requests.

The "qualified yes" answer comes in with the *automatic* failback. WAS uses a "test connection" method to determine if the primary resource has returned. The "testRetryInterval" property determines how frequently the test connection is performed. With CICS, the CTG resource adapter code will respond affirmatively in all cases, which would result in an attempted failback even if the primary CICS wasn't there. So automatic failback should not be used.

Instead, disable automatic failback ... using either the "enablePartial" property or the property to disable failback altogether. This will allow failover, but failback will not be attempted. You may still achieve failback, but it'll be manual ... using the z/OS MODIFY command to failback the resource. The z/OS MODIFY will carry out your instructions without checking first to see if the resource is really there. Therefore, you should be certain the primary has indeed returned before issuing the MODIFY.

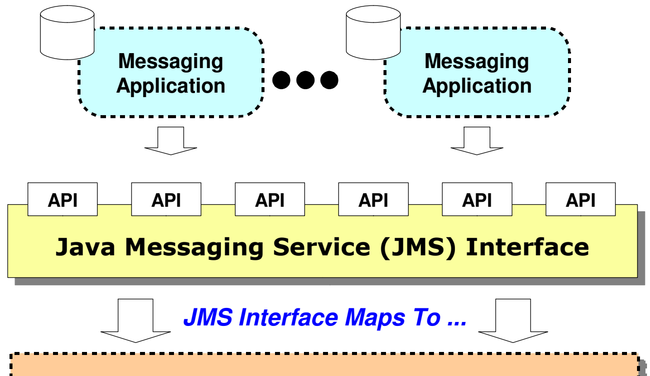
JMS and Messaging

With a focus on MQ

© 2013 IBM Corporation

The Difference Between JMS and Messaging Transport

JMS is a messaging *interface* while MQ is an example of a messaging transport that may be used under the interface:



JMS Interface Maps To ...

Messaging Transport

Middleware that implements and manages the message queues and the movement of messages to and from applications

Open standards above this line ... applications are unaware of the underlying message transport

Vendor-supplied transports below this line

IBM WebSphere MQ

The focus of this unit. An existing MQ infrastructure may be used as the transport under the JMS interface

WebSphere SIBus

The built-in all-Java messaging transport mechanism of WebSphere Application Server

Other Vendor Transports

WAS, as a Java EE server runtime, supports other vendor transports as well

Application types ...

32 IBM Americas Advanced Technical Skills Gaithersburg, MD

© 2013 IBM Corporation

The topic of messaging can be a little confusing at first because there's some terminology used that may not be properly understood. In WebSphere Application Server -- in fact, in any Java EE runtime -- messaging applications make use of something called "JMS" to handle the messaging.

Here's the key -- JMS itself is simply a programming interface. JMS itself does not have queue managers or anything that physically handles the messages. JMS relies on an underlying "transport" to do that. JMS is simply a programming interface that applications use. JMS then turns and hands the message off to the transport that is defined "below" the JMS layer.

What serves as the "messaging transport" is really up to you -- you define this as part of the setup for messaging in your WAS runtime environment. Your choices are IBM WebSphere MQ, the built in "Service Integration Bus" (SIBus) of WAS, or some other vendor messaging transport.

The JMS and Java EE environment is designed to "plug in" a transport below the JMS interface. So much of our ensuing discussion here will be to describe how that's done. Our specific focus will be on WebSphere MQ since that is a very prevalent messaging transport.

Note: as to whether you should use MQ or the built in SIBus, we prefer to frame that discussion like this -- if you have MQ and know how to use and manage MQ, then use MQ. We see little reason not to make use of a messaging infrastructure that serves your business well. If you don't have MQ today then the SIBus may well serve your needs.

Types of Messaging Applications

When discussing JMS configuration within WebSphere Application Server, it's helpful to keep straight three basic types of messaging applications:

Point-to-Point (or PUT/GET)

Very common model where an application puts a message on a queue and another application pulls the message

Message Driven Bean (MDB)

Receipt of a message in the assigned queue triggers a process to automatically get the message and invoke the Java bean

TechDocs WP101792

Pub / Sub

Applications "subscribe" to a queue. Provider "publishes" messages to the queue, and subscribers (one to many) pull from the subscription queue

Key concepts ...

© 2013 IBM Corporation

We'll soon see some terminology used that refers to different programming models for messaging applications. Therefore, it's worth investing a little time to review what those application designs are. There are three primary models that are used:

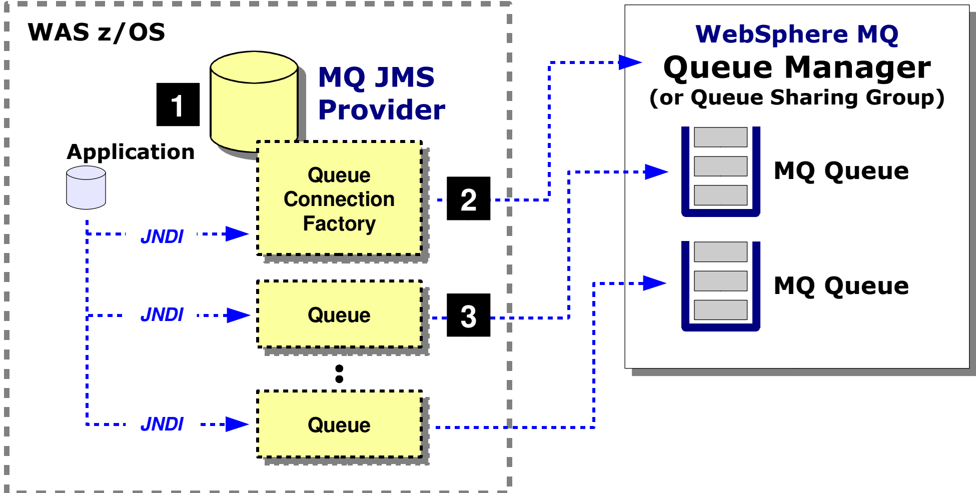
Point-to-Point -- or sometimes referred to as "simple PUT/GET" (that language goes back to the original MQ programming interface that used the verbs PUT and GET for messages) ... this model assumes a queue is used to exchange messages between two applications. One application puts a message on a queue, the other application periodically checks to see if a message is present and pulls the message off the queue. This is a very common application use for messaging.

Message Drive Beans -- on the surface this looks like point-to-point, but with this key difference: when a message is placed on the queue the receiving application gets a signal that a message is present. The application then "wakes up" and picks the message off the queue. In the Java EE world the target program is called a "message driven bean" because processing is "driven" by the arrival of a message on the queue that is being watched. In WAS V8 the use of MDBs implies the definition of an "activation spec". The Techdoc WP101792 provides a write-up and code samples for the defining and use of activation specs for MDBs.

Pub / Sub -- this stands for "publish and subscribe" and refers to a model where one application produces a message and some number of other applications -- between one and *n* -- subscribe to the message queue and receive whatever message content is placed there. The queue is referred to as a "topic queue." The key with this design is that the producer of content does not know nor care who is subscribed ... it simply knows that when content is available to be "published" it does so by placing that content on the queue. The subscribers -- one to many -- then pull a copy of the message.

Key Concepts of WAS and MQ Messaging

Some of the concepts are similar to JDBC and JCA, but there are some differences:



1 The MQ JMS Provider supplies the code needed to access MQ
Unlike JDBC or JCA, the provider code is supplied with WAS itself. It does not need to be installed.

2 Queue Connection Factory provides specifics about connecting to MQ
Typically a Queue Manager, but may be a queue sharing group. Two modes: binding and client.

3 Queue definitions provide abstraction of physical queue in QMGR
Applications may require multiple queues so it is common to have multiple JMS queue definitions

Bindings, client …

IBM Americas Advanced Technical Skills
Gaithersburg, MD
© 2013 IBM Corporation

Now it's time to take a look at some key concepts related to WAS JMS with MQ as the message transport. Our focus in on the definitions in WAS itself that relate to MQ as the transport mechanism under the JMS layer.

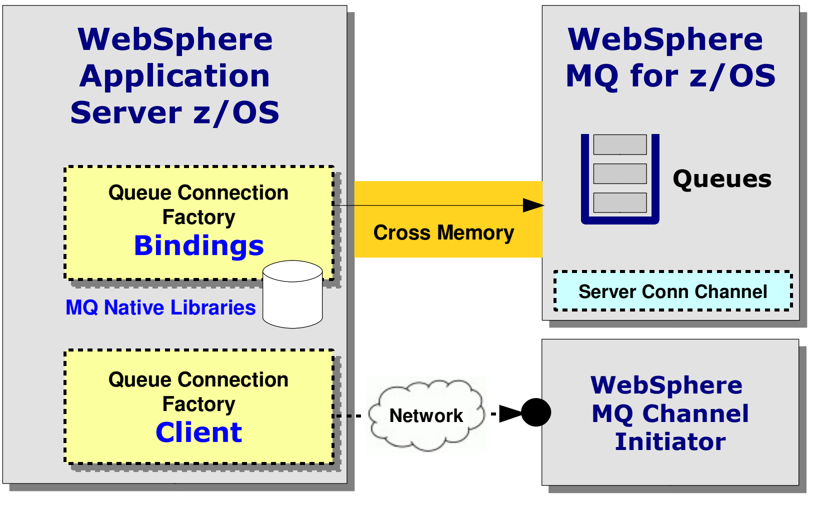
1. JMS Provider -- this is the code that makes access to MQ possible. It is analogous to the JDBC Provider and the JCA Resource Adapter. But unlike those two, the MQ Provider code is not something you install or configure into WAS ... it comes with WAS. It is simply there to be used. WAS maintenance updates the MQ provider code as needed.
2. Queue Connection Factory -- this is a definition that provides WAS knowledge of which MQ queue manager (or queue sharing group) you wish to connect to.
3. Queue -- this is a definition in WAS that provides an abstraction of a real queue in MQ. The queue definition in WAS is simply a name that resolves to a real queue name in the MQ queue manager named by the queue connection factory.

The application we'll use in the lab has a `<resource_ref>` for both the connection factory and the queue. It does a JNDI lookup of both, which connects the application to the definition in WAS. The specifics defined to the queue connection factory are what tells WAS which MQ queue manager to use; the specifics in the queue definition are what tells WAS what physical queue on that queue manager ot use.

Bindings Mode and Client Mode

The "local" and "remote" theme is carried to MQ connections as well ...

When configuring bindings you'll need to provide the path to the native shared object libraries as well as STEPLIB or LINKLIST to the SCSQANLE, SCSQANLU and SCSQAUTH data sets



When configuring client you'll need to know the host and port of the MQ channel initiator as well as the Server Connection Channel to use when connecting

JMS MQ provider ...

IBM Americas Advanced Technical Skills
Gaithersburg, MD

© 2013 IBM Corporation

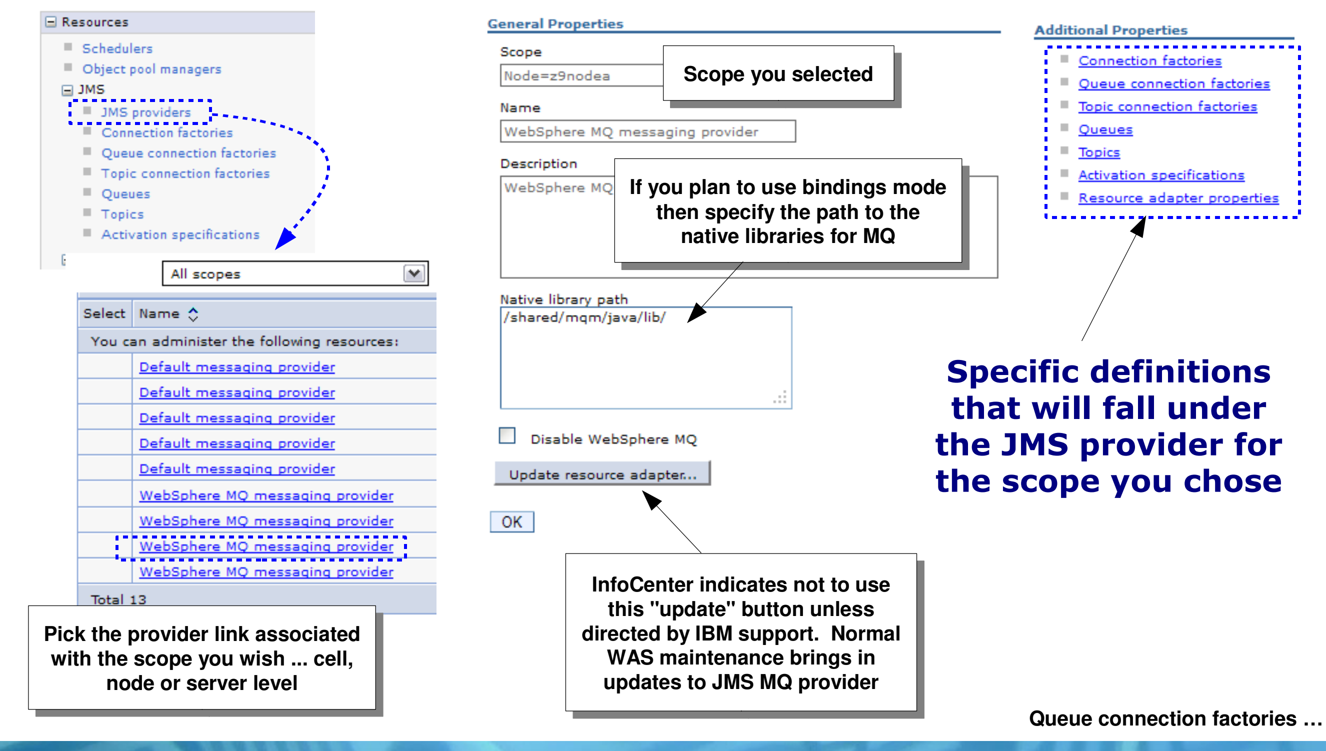
One of our common themes is the choice of a cross-memory connection or a network-based connection. We saw this with JDBC (Type 2 vs. Type 4) and we saw this with CICS (local EXCI vs. gateway or IPIC). MQ has the same -- bindings mode (cross-memory) and client mode (network).

Bindings mode uses the native interface of MQ to access the queue manager co-located on the same LPAR. That means you'll need to tell WAS about the native shared object library (a path in the MQ installation file system) and tell WAS about the native module libraries. The native module libraries may then be STEPLIBed to the servant region or put in Linklist. On our lab system they are in Linklist.

Client mode does not require access to the native libraries. The connection to the defined MQ queue manager is across a network connection ... either on the same LPAR or another LPAR. When defining this type of connection you'll need to know the host and port of the MQ channel initiator as well as the Server Connection Channel to use to access the QMGR.

Configuring the JMS Provider

The provider is very simple to configure ... the key is the connection factories and the queue definitions are access from the provider properties screen:



Queue connection factories ...

© 2013 IBM Corporation

As we just mentioned, the JMS Provider code for MQ is provided with WAS. You don't need to "install" it. You do however have to tell WAS which JMS provider you plan to use -- MQ or the built in "Default Messaging" -- and tell WAS what "scope" you want the provider to exist at.

On the Admin Console under "JMS" you'll find a link for "JMS Providers". When you click on that link you'll see a panel that shows all the JMS providers at all scopes. The page looks a little odd at first because of the layout, but once you realize it's showing "all providers, all scopes" it starts to make a bit more sense. Pick the MQ provider at the scope you desire.

On the general properties for the MQ provider there's a field for the "Native library path" -- this, along with the MQ module libraries, provides the ability to configure Bindings mode. This field is asking for the shared object files in the file system, which would be the `/java/lib` directory in the MQ install path.

There's a button on this page labeled "Update resource adapter." Be aware the InfoCenter suggests this not be used unless directed by IBM support. Updates to the MQ provider come with WAS maintanance. This button would be used when IBM support has an updated adapter it wishes you to try.

Off to the right you'll see a list of "Additonal Properties," and under there you'll see some of the things we spoke of earlier -- queue connection factories, queue definitions, activation specs, topic queues. Our focus here will be on the queue connection factory definition and the queue definition.

Queue Connection Factories

These provide the information on how to connect to the MQ queue manager:

Additional Properties
- Connection factories
- Queue connection factories
- Topic connection factories
- Queues
- Topics
- Activation specifications
- Resource adapter properties

Pub-Sub Topic or Point-to-Point

Point-to-Point

Pub-Sub

* Name

* JNDI name

Provide a display name and a JNDI name

Supply queue manager details

Enter details about the queue manager or queue sharing group that you wish to connect to.

Queue manager or queue sharing group name

Provide the name of the QMGR or the QSG

Enter connection details

Enter the details required to manager or queue sharing

Transport: Bindings, then client

Transport: Bindings / Bindings, then client / Client / Bindings

Enter host and port information in the form of separate hostname and port values

* Hostname

Port

or

Enter host and port information name list

Connection name list

Server connection channel

For client ... simple `host:port` information

For client ... connection name list: `'host1.com(1234),host2.com(4321)'`

For client ... server connection channel to use

If Bindings ...

Then just the Queue Manager name is needed. The host and port fields are grayed-out.

If Client ...

Choose either host/port or connection list, then supply the server connection channel information

If Bindings, then client ...

Choose either host/port or connection list, then supply the server connection channel information

Queue definitions ...

© 2013 IBM Corporation

One of the links under "Additional Properties" is "Queue connection factories" and that will be our focus. Two other connection factory links are present ... one is a combined point-to-point and pub-sub link, the other is for pub-sub only. Queue connection factories are used for point-to-point and MDB applications, and as such is a frequently used configuration with WAS and MQ.

When you click on that link you'll be presented with two fields asking for the display name and the JNDI name for the queue connection factory. This is just like we've seen before for JDBC data sources and JCA connection factories. One name is used by the Admin Console to display the resource; the other name is used by applications to look up the resource and make the connection.

Since this is an MQ connection factory WAS knows to ask you for the Queue Manager name or the Queue Sharing Group name. (Queue Sharing Groups are a means of having multiple Queue Managers on Parallel Sysplex share a set of common queues within the Sysplex Coupling Facility.)

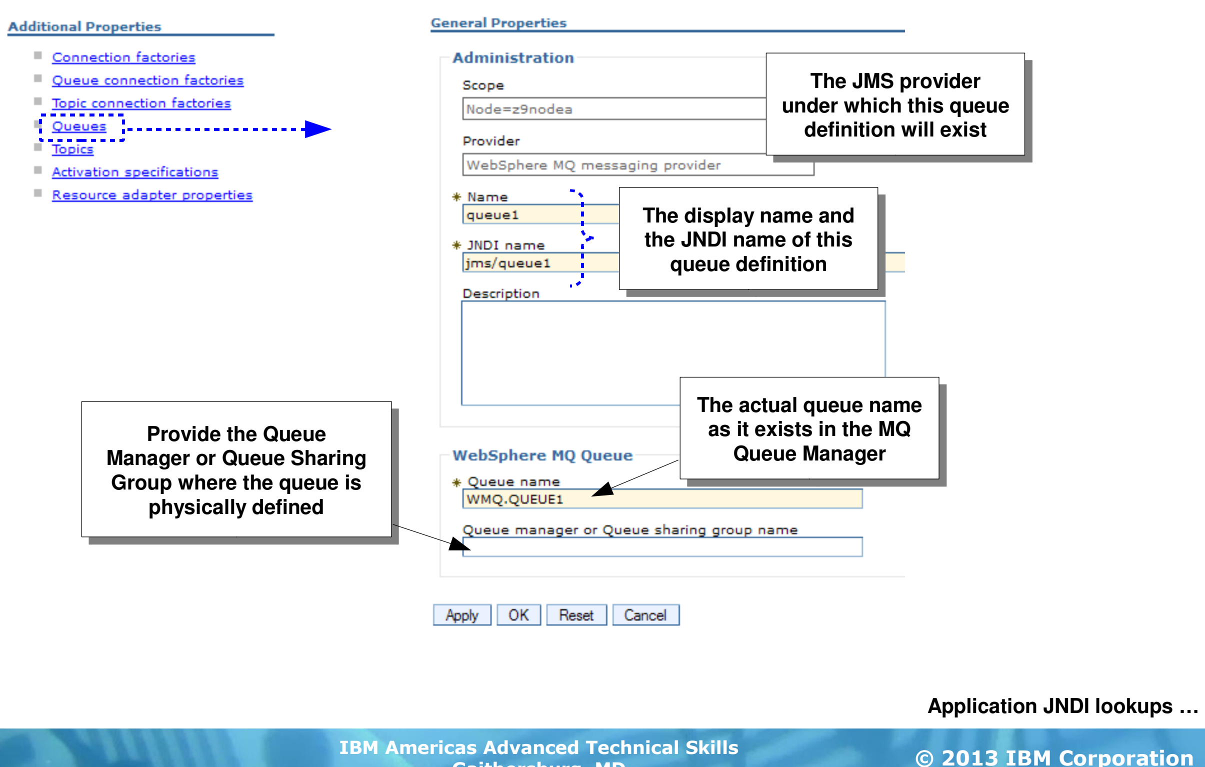
The next configuration step asks you for how WAS is to make the connection. This is the choice between Bindings or Client ... with a third twist: bindings, and if that doesn't work then client. (In a sense this is like the resource failover we've seen before.)

If you select anything other than Bindings then you have a choice for supplying the client mode specifics. One option is to supply a single host and port specification. The other option is to supply a connection name list with a format as shown on the chart.

Whether you chose the single host and port or the connection name list, in either case you'll then need to supply the MQ server connection channel this client will use when it accesses the Queue Manager. This is a value you'd get from your MQ administrator.

Queue Definitions

You would have as many queue definitions as you have queues in MQ that you wish applications to use:



Application JNDI lookups …

© 2013 IBM Corporation

Messaging applications make use of queues, and rather than hard-code the actual queue names in the application (Java EE is all about abstracting such things), you define "queues" in WAS that then resolve to the actual queue name in the QMGR specified on the queue connection factory.

Here again, a display name and a JNDI name is called for. Further down you specify the actual queue name as it is defined in the QMGR.

Application Access to JMS Resources

This involves mapping the `<resource-ref>` tags in the application deployment descriptors to the JMS definitions you've created:

SimpleJMS.ear

Messaging App Used in Upcoming Lab

web.xml

```
<resource-ref id="ResourceRef_1074045272521">
   <res-ref-name>jms/QCF</res-ref-name>
   <res-type>javax.jms.QueueConnectionFactory</res-type>
</resource-ref>
<resource-ref id="ResourceRef_1074045272531">
   <res-ref-name>jms/IncomingQueue</res-ref-name>
   <res-type>javax.jms.Queue</res-type>
</resource-ref>
<resource-ref id="ResourceRef_1074045272551">
   <res-ref-name>jms/OutgoingQueue</res-ref-name>
   <res-type>javax.jms.Queue</res-type>
</resource-ref>
```

Step 6 Map resource references to resources

Admin Console prompts you to map resource refs to the JMS JNDI definitions created

javax.jms.Queue

Select	Module	Bean	URI	Resource Reference	Target Resource JNDI Name
☐	SimpleJMSWeb		SimpleJMSWeb.war,WEB-INF/web.xml	jms/IncomingQueue	Browse...
☐	SimpleJMSWeb		SimpleJMSWeb.war,WEB-INF/web.xml	jms/OutgoingQueue	Browse...

javax.jms.QueueConnectionFactory

Select	Module	Bean	URI	Resource Reference	Target Resource JNDI Name	Login configuration
☐	SimpleJMSWeb		SimpleJMSWeb.war,WEB-INF/web.xml	jms/QCF	Browse...	

© 2013 IBM Corporation

In our earlier discussions of JDBC and JCA we saw that the application did a JNDI lookup of the data source or connection factory. That provided the application the connection specifics it needed to access the DB2 instance or CICS region it needed.

For a messaging application we have the potential of requiring several JNDI lookups -- one for the queue connection factory (which tells which MQ Queue Manager to use), and one for each queue the application uses. That's what you'll see in the upcoming lab. The application has three `<resource-ref>` definitions that need resolving when the application is deployed -- two queue resolutions and one queue connection factory resolution.

End of Unit